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`
- `min(3,x) + 20`
- `x = 7`
- `x *= 2`

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)
```

Operatoren

Unäre Operatoren:

symbol	*name*	*types*	*L/R*	*level*
++	Post-increment	(var) zahl, char	links	2
--	Post-decrement	(var) zahl, char	links	2
++	Pre-increment	(var) zahl, char	rechts	3
--	Pre-decrement	(var) zahl, char	rechts	3
+	unäres Plus	zahl, char	rechts	3
-	unäres Minus	zahl, char	rechts	3
!	Negation	boolean	rechts	3

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`
- `min(3,x) + 20`
- `x = 7`
- `x *= 2`

Achtung

Operatoren

Unäre Operatoren:

symbol	*name*	*types*	*L/R*	*level*
++	Post-increment	(var) zahl, char	links	2
--	Post-decrement	(var) zahl, char	links	2
++	Pre-increment	(var) zahl, char	rechts	3
--	Pre-decrement	(var) zahl, char	rechts	3
+	unäres Plus	zahl, char	rechts	3
-	unäres Minus	zahl, char	rechts	3
!	Negation	boolean	rechts	3

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-Increment).
- x++ tut das, nachdem der Wert ermittelt wurde (Post-Increment).
- `b = x++;` entspricht:

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

- `b = ++x;` entspricht:

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


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-Increment).
- x++ tut das, nachdem der Wert ermittelt wurde (Post-Increment).
- `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

symbol	*name*	*types*	*L/R*	*level*
*	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

symbol	*name*	*types*	*L/R*	*level*
+	Konkatenation	string	links	5

Operatoren

Vergleichsoperatoren:

symbol	*name*	*types*	*L/R*	*level*
>	größer	zahl, char	links?	7
>=	größergleich	zahl, char	links?	7
<	kleiner	zahl, char	links?	7
<=	kleinergleich	zahl, char	links?	7
==	gleich	primitiv	links	8
!=	ungleich	primitiv	links	8

Operatoren

Binäre arithmetische Operatoren:

`byte`, `short`, `char` werden nach `int` konvertiert

symbol	*name*	*types*	*L/R*	*level*
*	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

symbol	*name*	*types*	*L/R*	*level*
+	Konkatenation	string	links	5

Operatoren

Vergleichsoperatoren:

symbol	*name*	*types*	*L/R*	*level*
>	größer	zahl, char	links?	7
>=	größergleich	zahl, char	links?	7
<	kleiner	zahl, char	links?	7
<=	kleinergleich	zahl, char	links?	7
==	gleich	primitiv	links	8
!=	ungleich	primitiv	links	8

Operatoren

Boolsche Operatoren:

symbol	*name*	*types*	*L/R*	*level*
&&	Und-Bedingung	boolean	links	12
||	Oder-Bedingung	boolean	links	13

Operatoren

Zuweisungsoperatoren:

symbol	*name*	*types*	*L/R*	*level*
=	Zuweisung	var, wert	rechts	15
*=,/=, %=, +=, -=	Zuweisung	var, wert	rechts	15

Für die letzte Form gilt:

$$v \circ\!= a \iff v = (type(v))\ (v \circ a)$$

Operatoren

Boolsche Operatoren:

symbol	*name*	*types*	*L/R*	*level*
&&	Und-Bedingung	boolean	links	12
||	Oder-Bedingung	boolean	links	13

Operatoren

Zuweisungsoperatoren:

symbol	*name*	*types*	*L/R*	*level*
=	Zuweisung	var, wert	rechts	15
*=,/=, %=, +=, -=	Zuweisung	var, wert	rechts	15

Für die letzte Form gilt:

$$v \circeq a \iff v = (type(v))\ (v \circ a)$$

Operatoren

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

Operatoren

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

Operatoren

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

Operatoren

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 ...");
```

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

Operatoren

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 ...");
```

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

Operatoren

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

Operatoren

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

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

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 all Operanden aus (Kurzschlussauswertung).

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

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 all 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 all 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:

symbol	*name*	*types*	*L/R*	*level*
( )	Klammerung	*	links	0

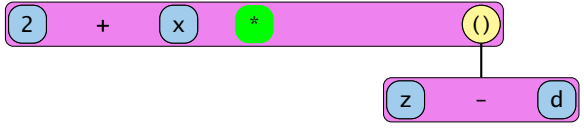
Beispiel: 2 + x * (z - d)

Beispiel: 2 + x * (z - d)

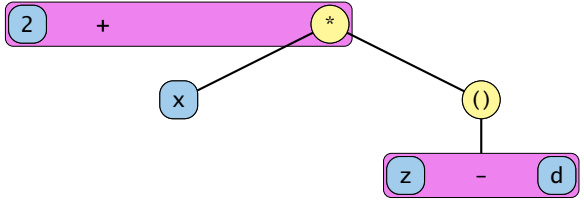
Beispiel: 2 + x * (z - d)

Beispiel: 2 + x * (z - d)

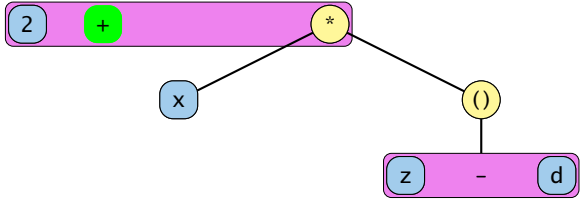
Beispiel: 2 + x * (z - d)



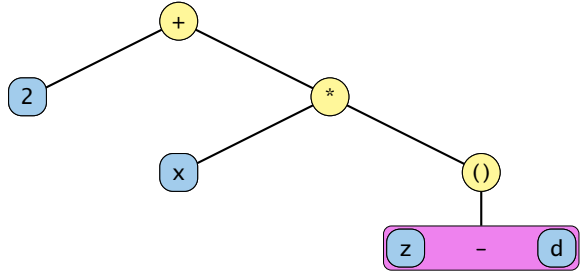
Beispiel: 2 + x * (z - d)



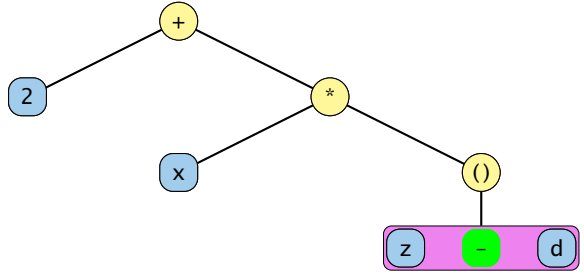
Beispiel: 2 + x * (z - d)



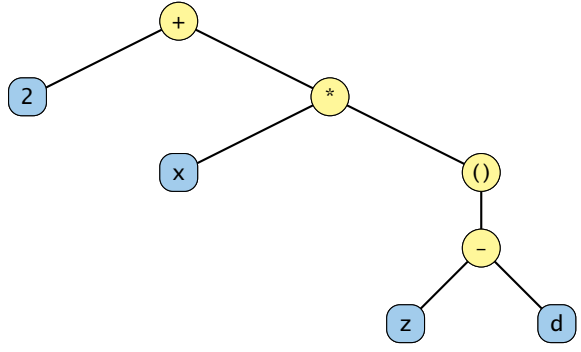
Beispiel: 2 + x * (z - d)



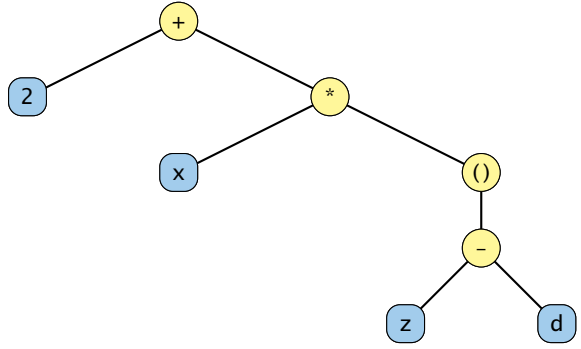
Beispiel: 2 + x * (z - d)



Beispiel: 2 + x * (z - d)

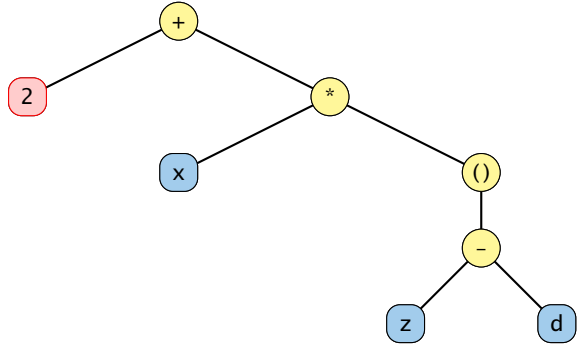


Beispiel: 2 + x * (z - d)



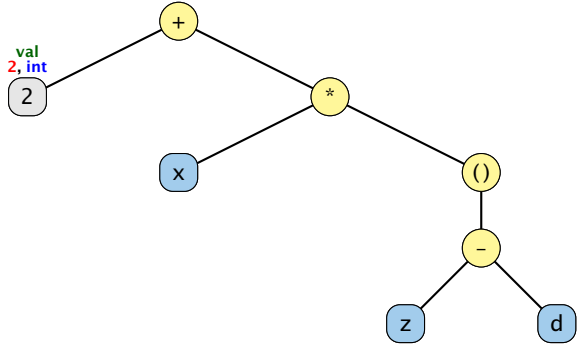
x -3 d 7 z 5

Beispiel: 2 + x * (z - d)



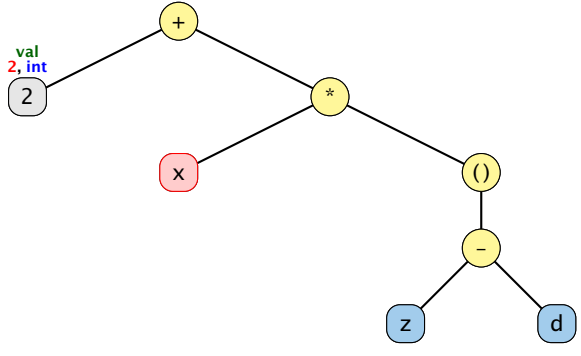
x -3 d 7 z 5

Beispiel: 2 + x * (z - d)



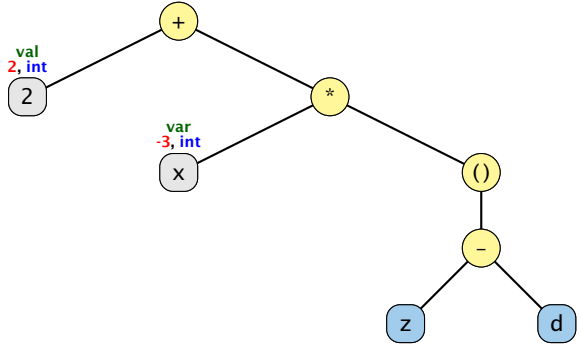
x -3 d 7 z 5

Beispiel: 2 + x * (z - d)



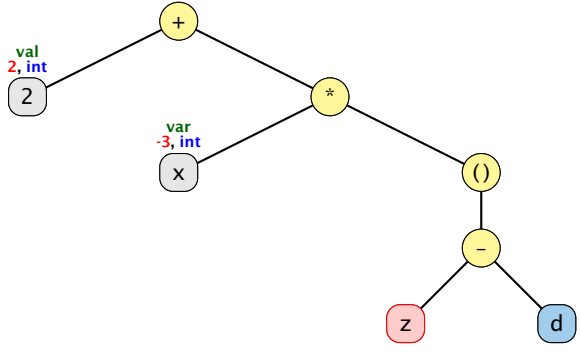
x -3 d 7 z 5

Beispiel: 2 + x * (z - d)



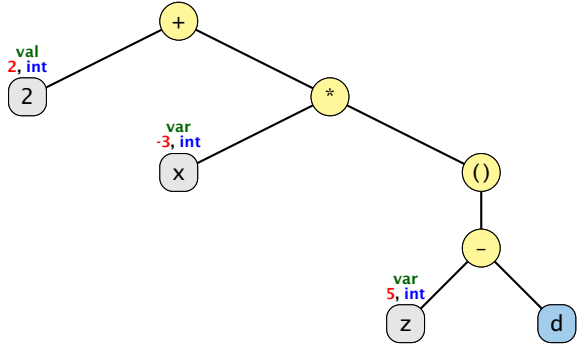
x -3 d 7 z 5

Beispiel: 2 + x * (z - d)



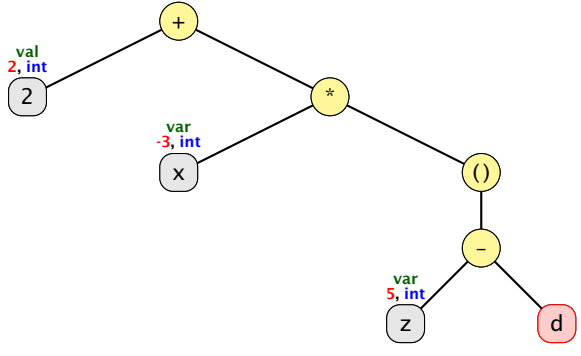
x -3 d 7 z 5

Beispiel: 2 + x * (z - d)



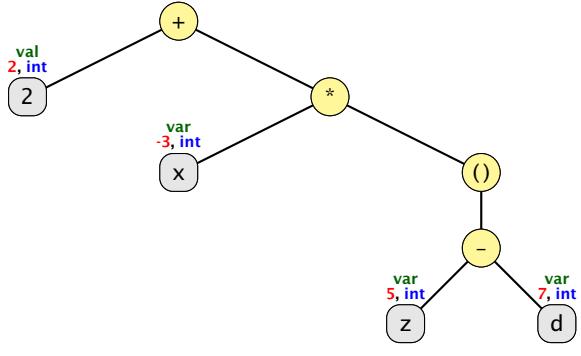
x -3 d 7 z 5

Beispiel: 2 + x * (z - d)



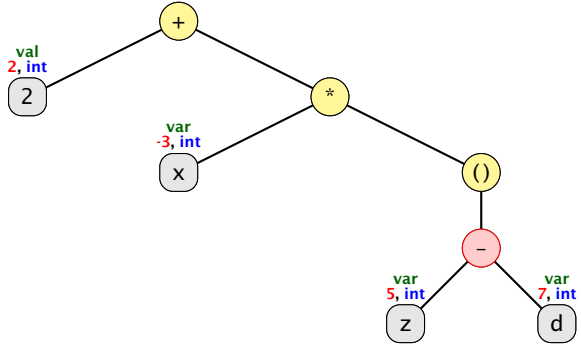
x -3 d 7 z 5

Beispiel: 2 + x * (z - d)



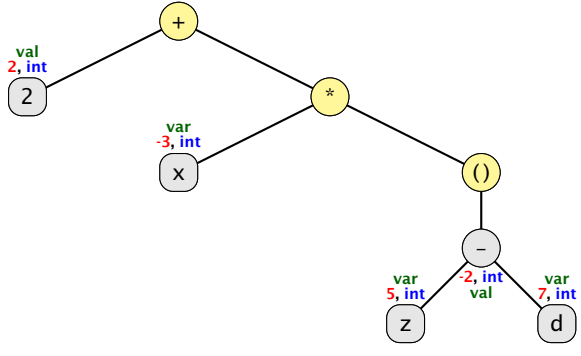
x -3 d 7 z 5

Beispiel: 2 + x * (z - d)



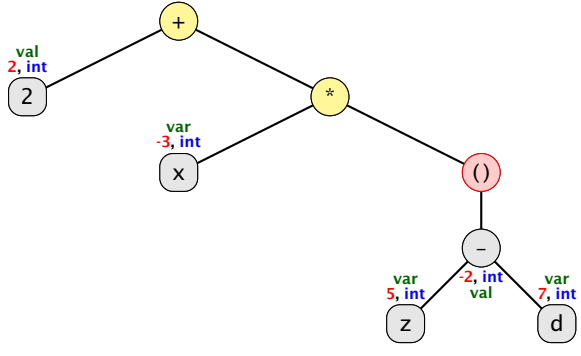
x -3 d 7 z 5

Beispiel: 2 + x * (z - d)



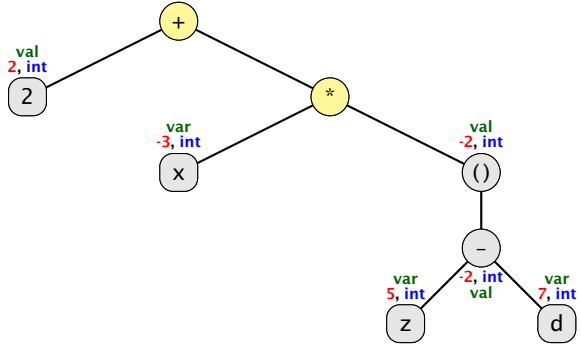
x -3 d 7 z 5

Beispiel: 2 + x * (z - d)



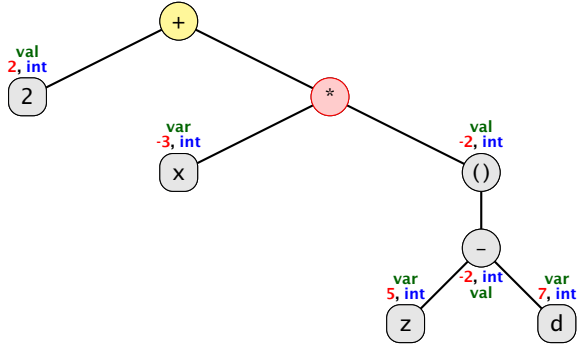
x -3 d 7 z 5

Beispiel: 2 + x * (z - d)



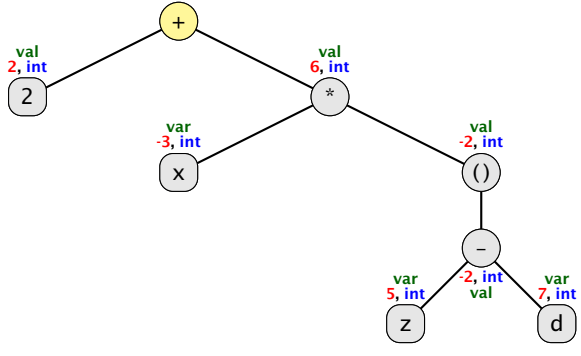
x -3 d 7 z 5

Beispiel: 2 + x * (z - d)



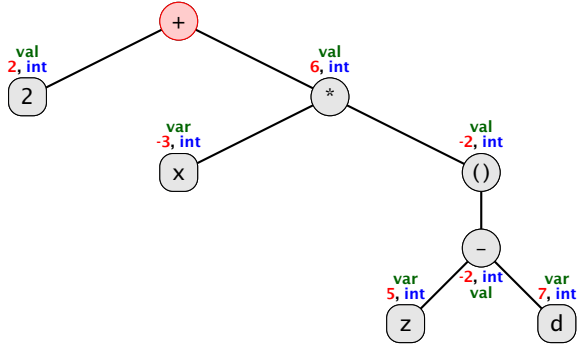
x -3 d 7 z 5

Beispiel: 2 + x * (z - d)



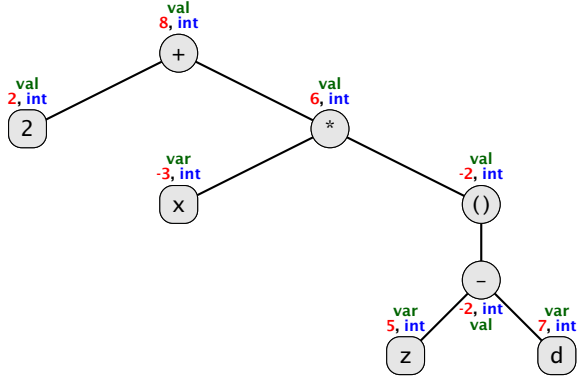
x -3 d 7 z 5

Beispiel: 2 + x * (z - d)



x -3 d 7 z 5

Beispiel: 2 + x * (z - d)

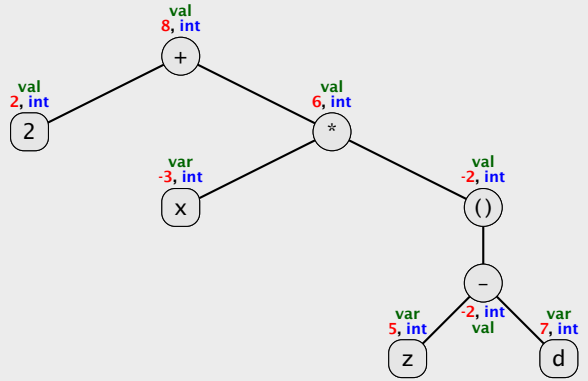


x -3 d 7 z 5

Beispiel: a = b = c = d = 0

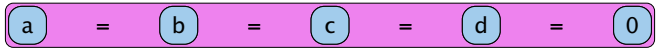
a	=	b	=	c	=	d	=	0

Beispiel: 2 + x * (z - d)

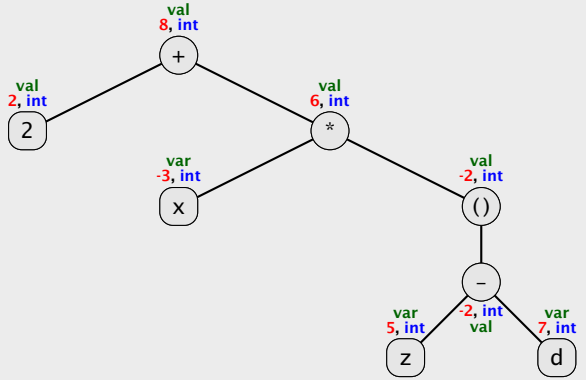


x -3 d 7 z 5

Beispiel: a = b = c = d = 0

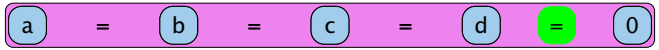


Beispiel: 2 + x * (z - d)

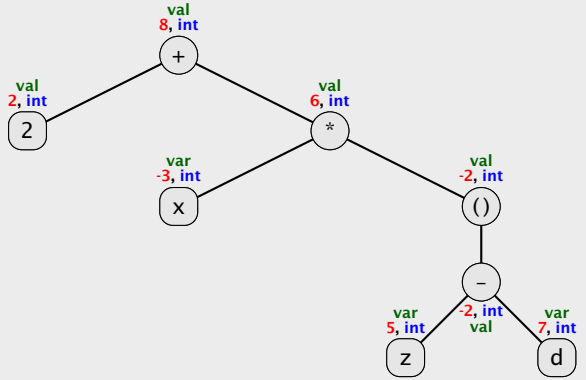


x -3 d 7 z 5

Beispiel: a = b = c = d = 0

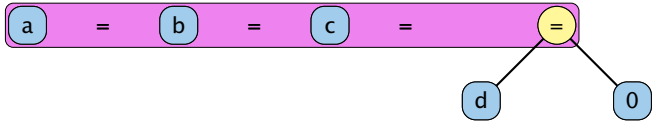


Beispiel: 2 + x * (z - d)

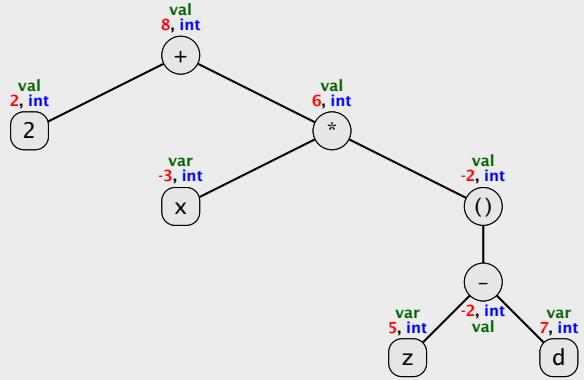


x -3 d 7 z 5

Beispiel: a = b = c = d = 0

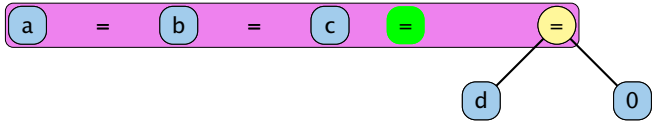


Beispiel: 2 + x * (z - d)

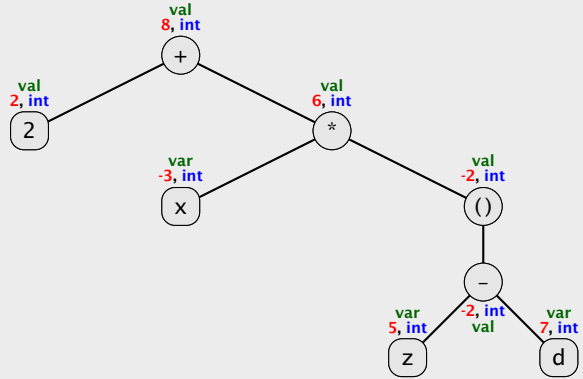


x -3 d 7 z 5

Beispiel: a = b = c = d = 0

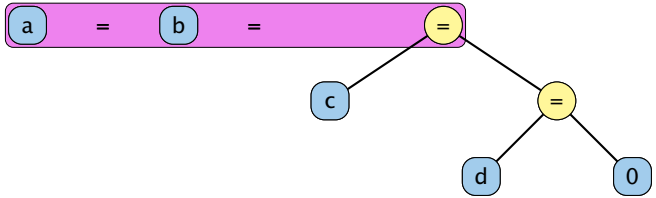


Beispiel: 2 + x * (z - d)

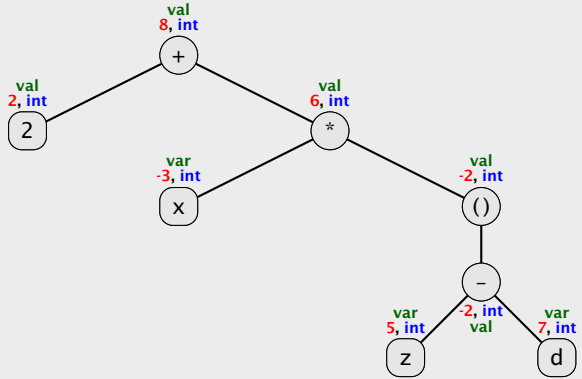


x -3 d 7 z 5

Beispiel: a = b = c = d = 0

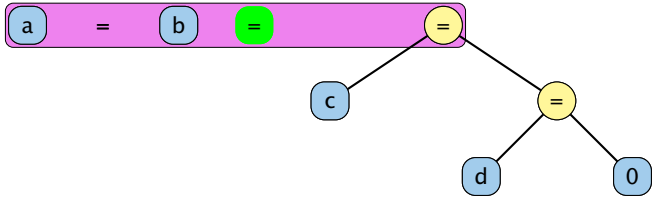


Beispiel: 2 + x * (z - d)

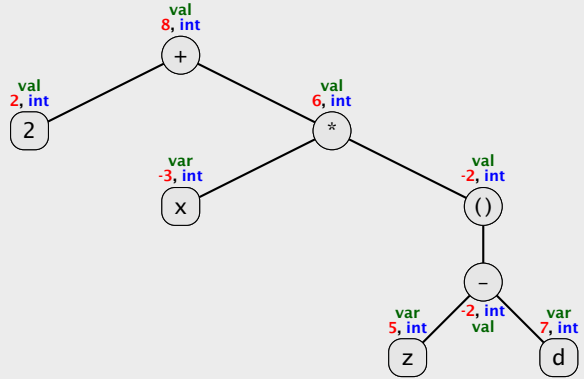


x -3 d 7 z 5

Beispiel: a = b = c = d = 0

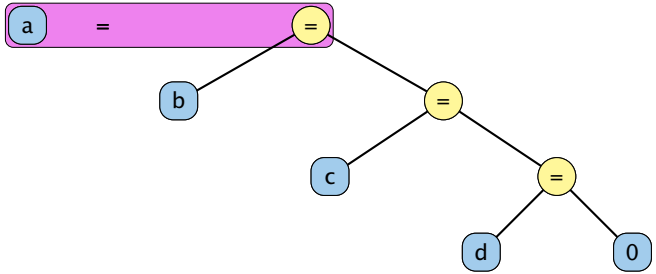


Beispiel: 2 + x * (z - d)

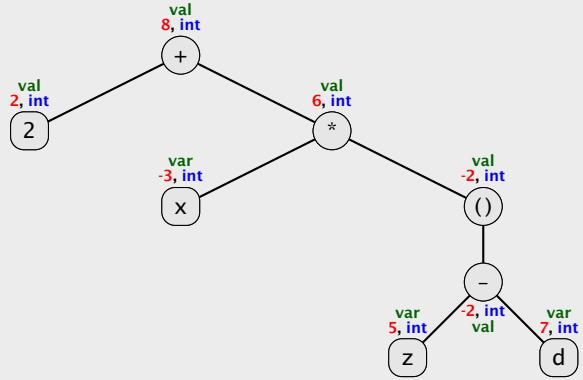


x -3 d 7 z 5

Beispiel: a = b = c = d = 0

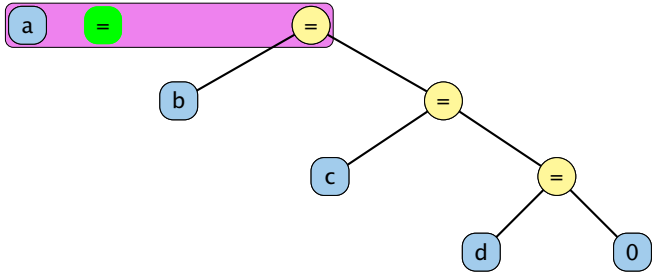


Beispiel: 2 + x * (z - d)

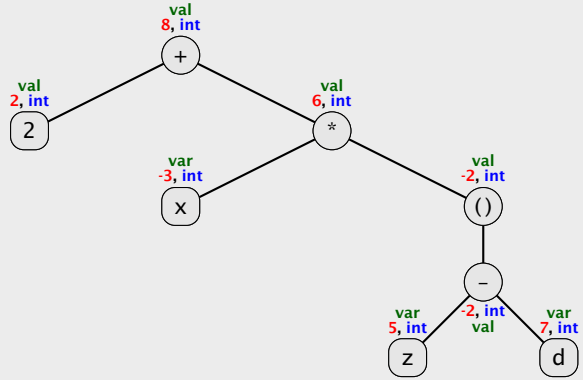


x -3 d 7 z 5

Beispiel: a = b = c = d = 0

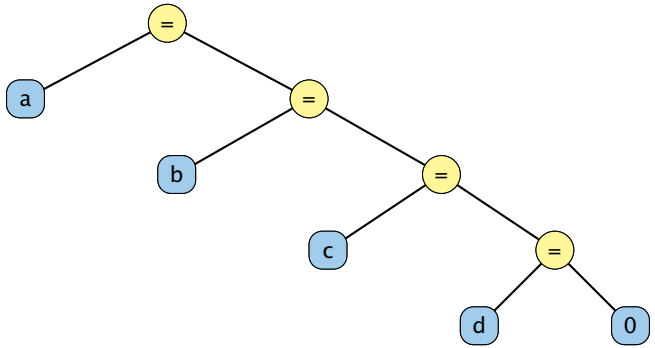


Beispiel: 2 + x * (z - d)

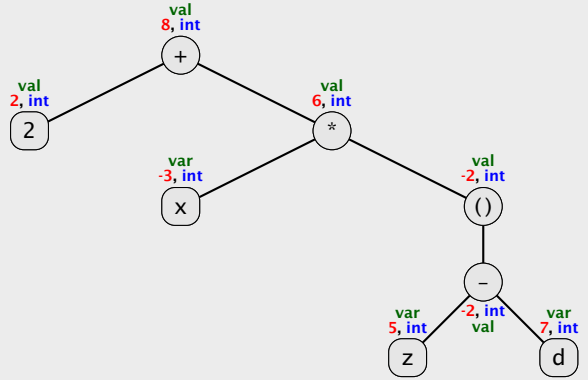


x -3 d 7 z 5

Beispiel: a = b = c = d = 0

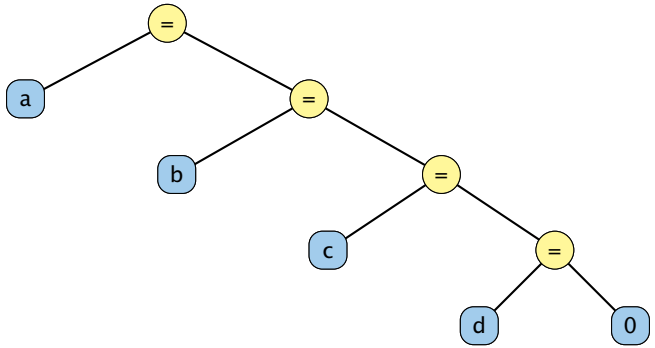


Beispiel: 2 + x * (z - d)



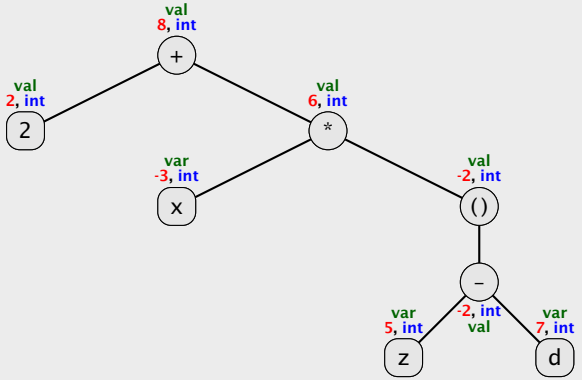
x -3 d 7 z 5

Beispiel: a = b = c = d = 0



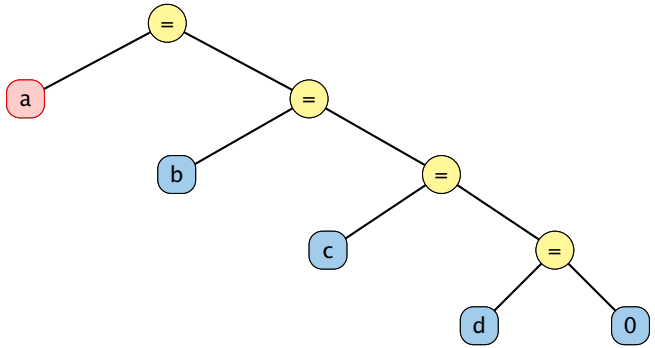
a -3 b 7 c 5 d 2

Beispiel: 2 + x * (z - d)



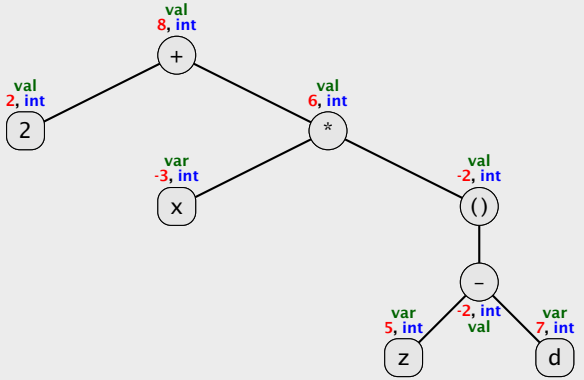
x -3 d 7 z 5

Beispiel: a = b = c = d = 0



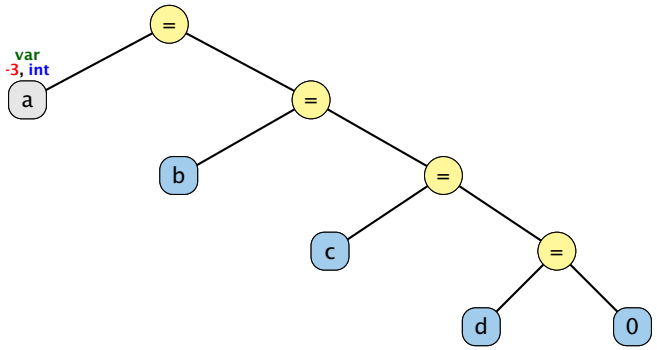
a -3 b 7 c 5 d 2

Beispiel: 2 + x * (z - d)



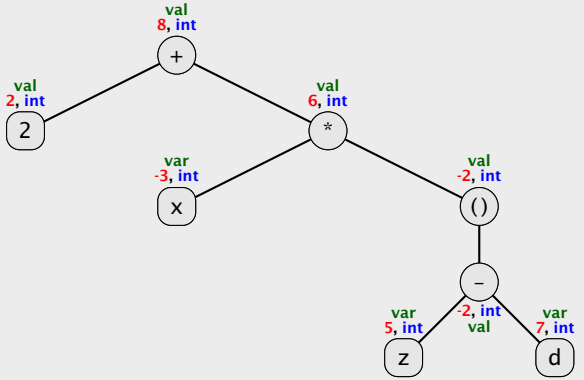
x -3 d 7 z 5

Beispiel: a = b = c = d = 0



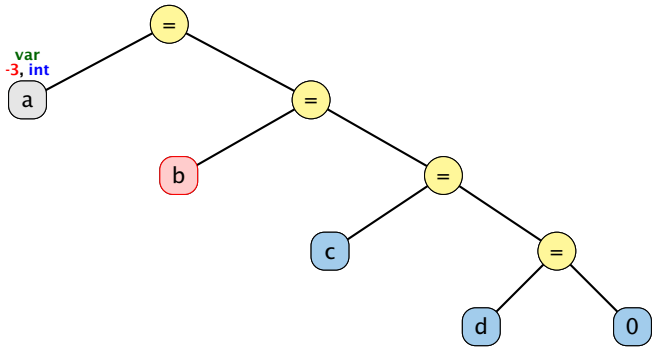
a -3 b 7 c 5 d 2

Beispiel: 2 + x * (z - d)



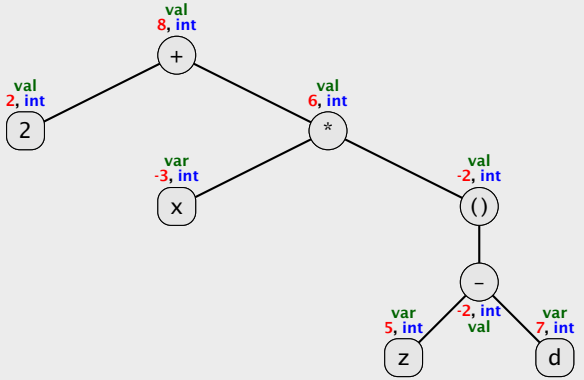
x -3 d 7 z 5

Beispiel: a = b = c = d = 0



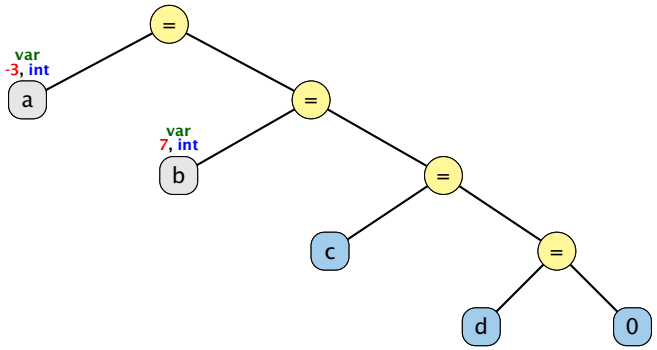
a -3 b 7 c 5 d 2

Beispiel: 2 + x * (z - d)



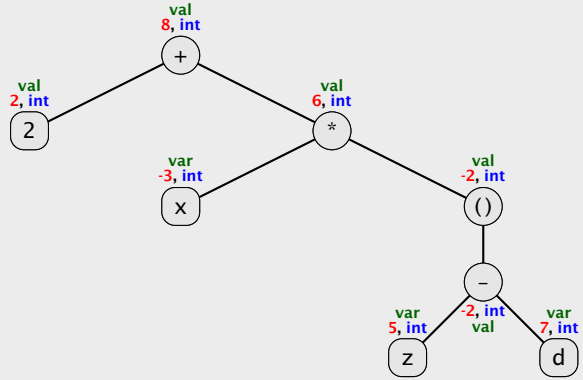
x -3 d 7 z 5

Beispiel: a = b = c = d = 0



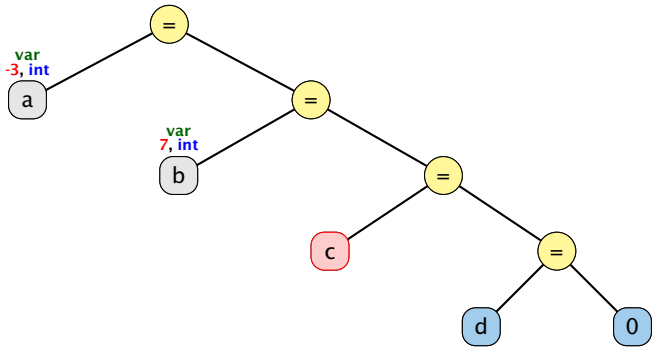
a -3 b 7 c 5 d 2

Beispiel: 2 + x * (z - d)



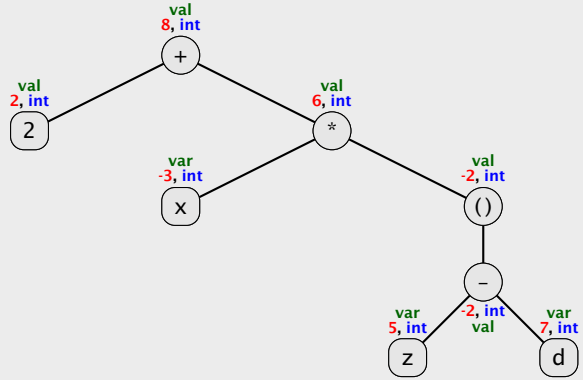
x -3 d 7 z 5

Beispiel: a = b = c = d = 0



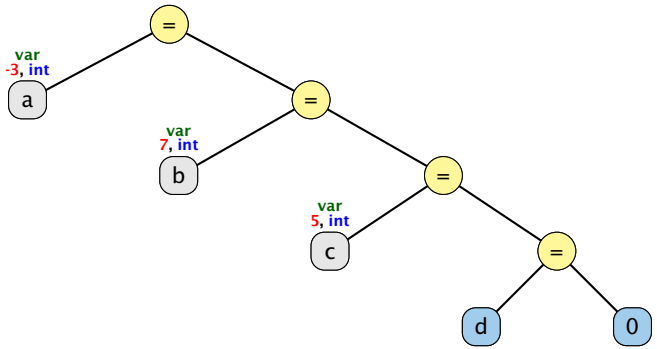
a -3 b 7 c 5 d 2

Beispiel: 2 + x * (z - d)



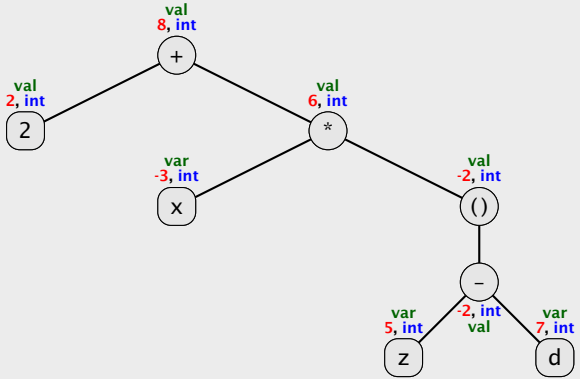
x -3 d 7 z 5

Beispiel: a = b = c = d = 0



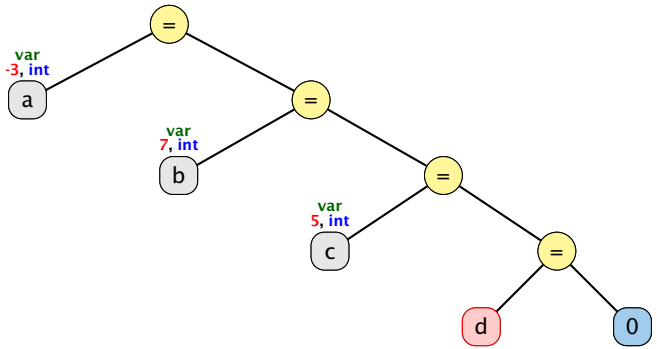
a -3 b 7 c 5 d 2

Beispiel: 2 + x * (z - d)



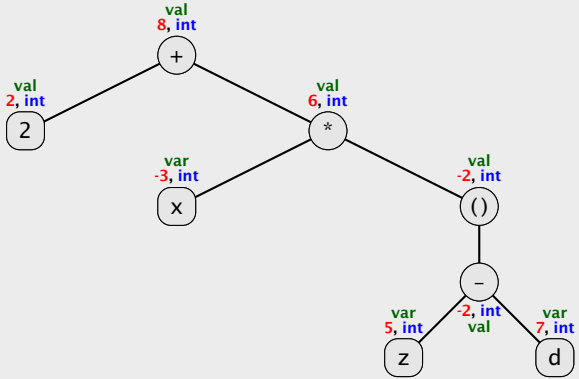
x -3 d 7 z 5

Beispiel: a = b = c = d = 0



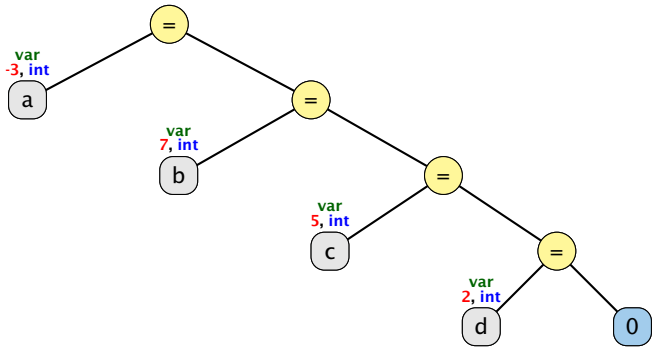
a -3 b 7 c 5 d 2

Beispiel: 2 + x * (z - d)



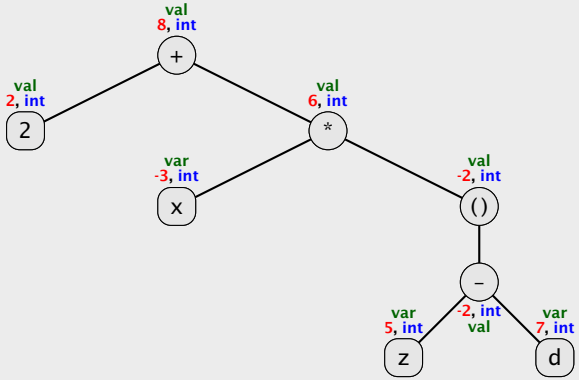
x -3 d 7 z 5

Beispiel: a = b = c = d = 0



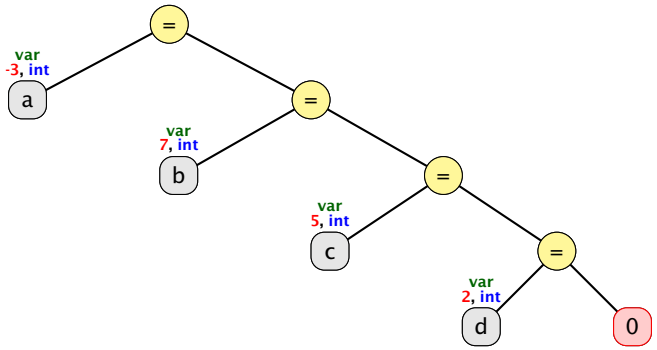
a -3 b 7 c 5 d 2

Beispiel: 2 + x * (z - d)



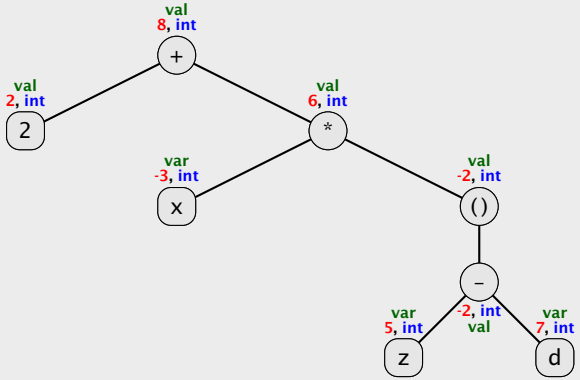
x -3 d 7 z 5

Beispiel: a = b = c = d = 0



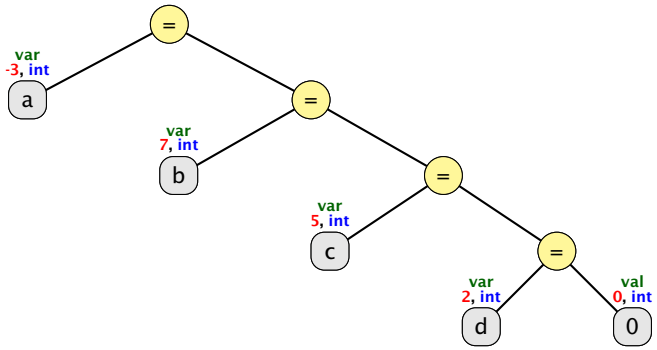
a -3 b 7 c 5 d 2

Beispiel: 2 + x * (z - d)



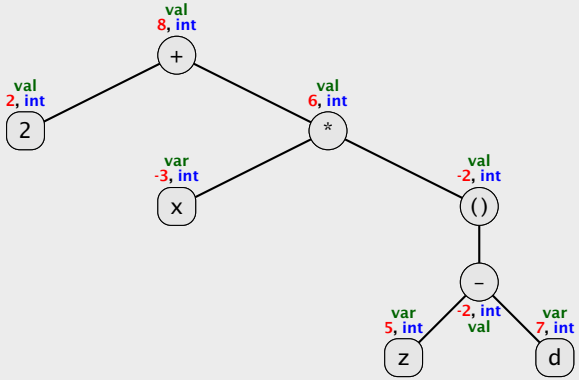
x -3 d 7 z 5

Beispiel: a = b = c = d = 0



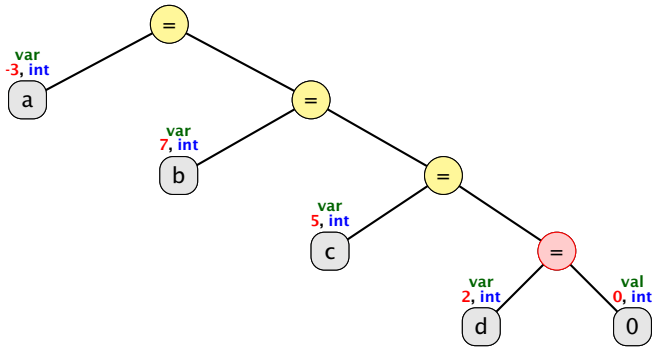
a -3 b 7 c 5 d 2

Beispiel: 2 + x * (z - d)



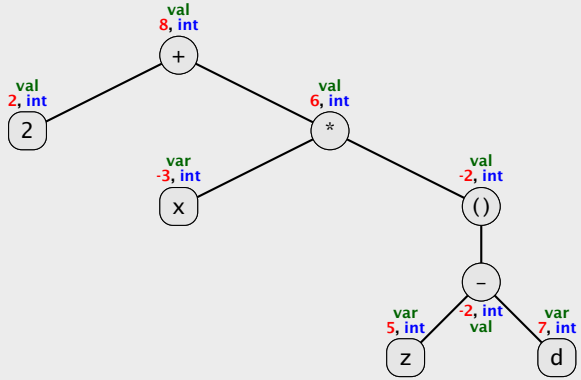
x -3 d 7 z 5

Beispiel: a = b = c = d = 0



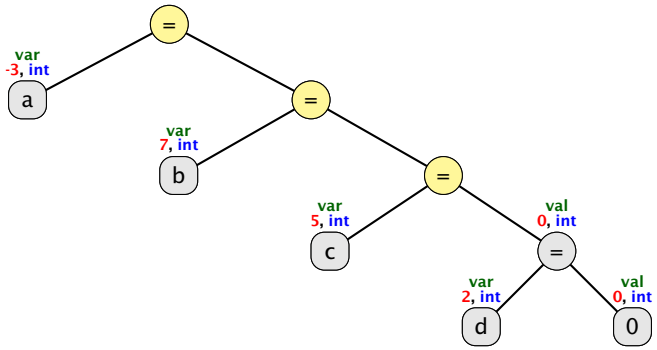
a -3 b 7 c 5 d 2

Beispiel: 2 + x * (z - d)



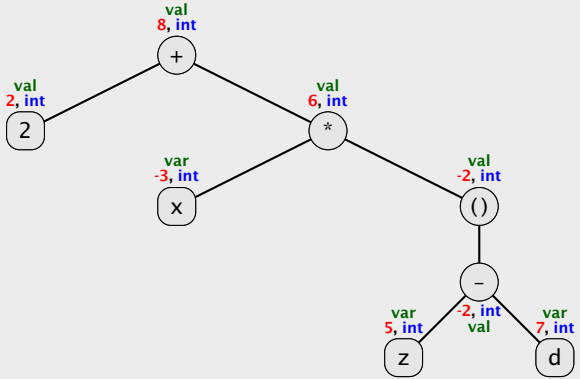
x -3 d 7 z 5

Beispiel: a = b = c = d = 0



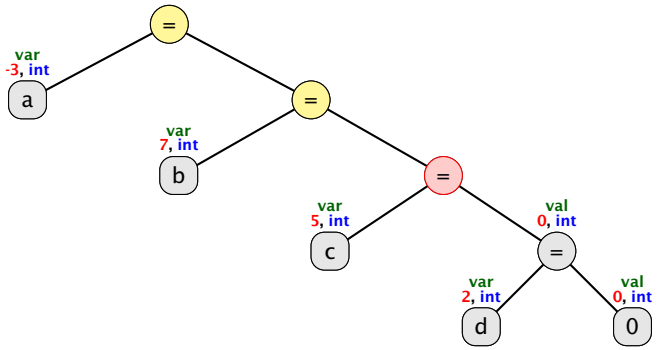
a -3 b 7 c 5 d 0

Beispiel: 2 + x * (z - d)



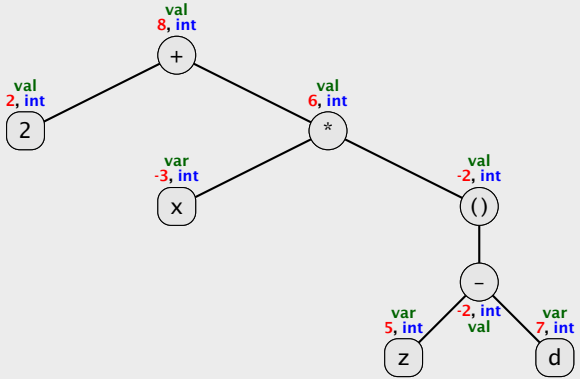
x -3 d 7 z 5

Beispiel: a = b = c = d = 0



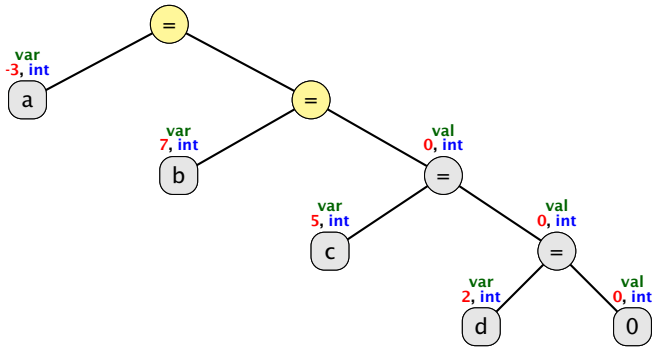
a -3 b 7 c 5 d 0

Beispiel: 2 + x * (z - d)



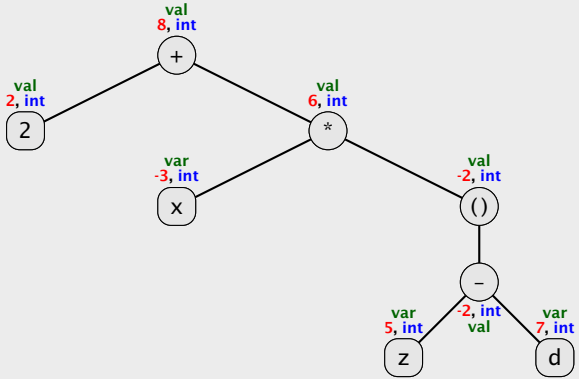
x -3 d 7 z 5

Beispiel: a = b = c = d = 0



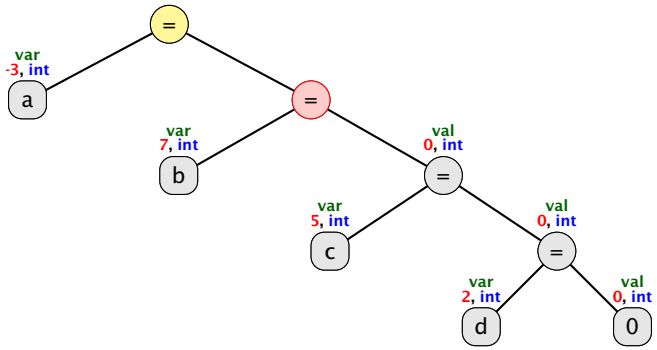
a -3 b 7 c 0 d 0

Beispiel: 2 + x * (z - d)



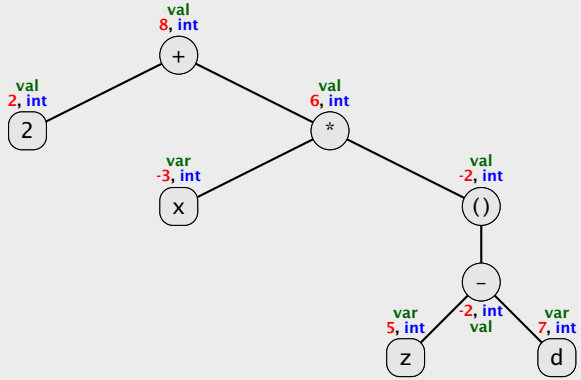
x -3 d 7 z 5

Beispiel: a = b = c = d = 0



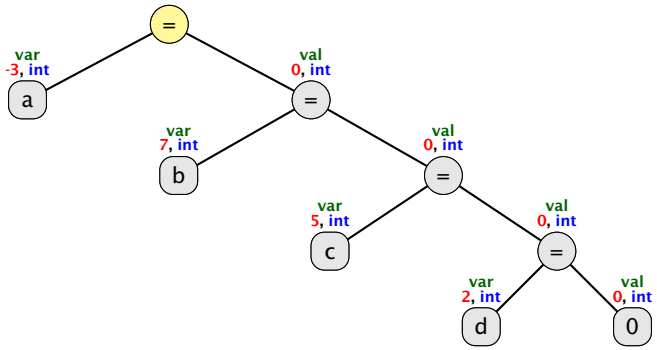
a -3 b 7 c 0 d 0

Beispiel: 2 + x * (z - d)



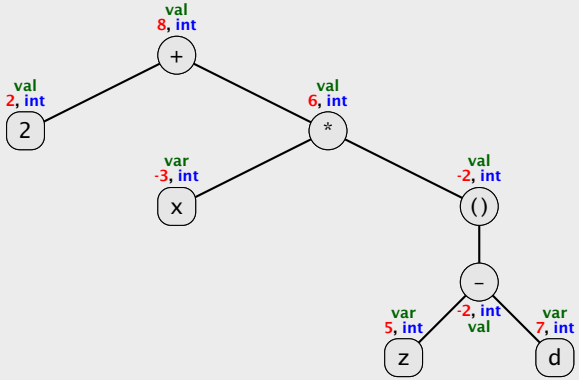
x -3 d 7 z 5

Beispiel: a = b = c = d = 0



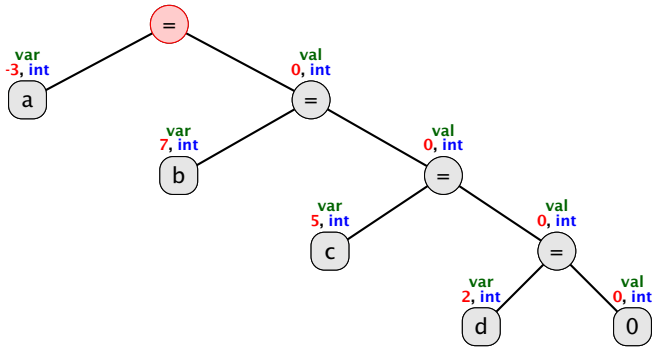
a -3 b 0 c 0 d 0

Beispiel: 2 + x * (z - d)



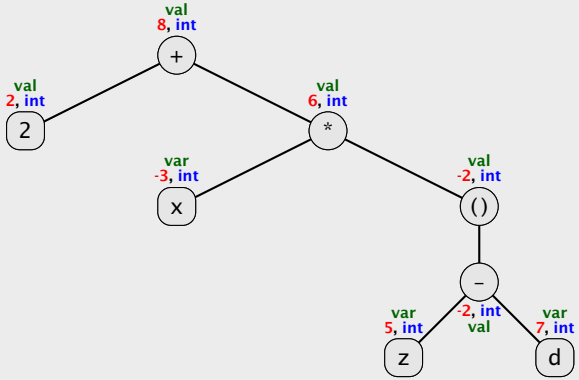
x -3 d 7 z 5

Beispiel: a = b = c = d = 0



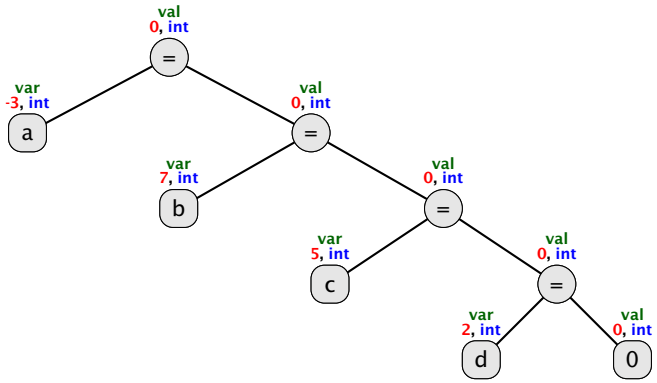
a -3 b 0 c 0 d 0

Beispiel: 2 + x * (z - d)



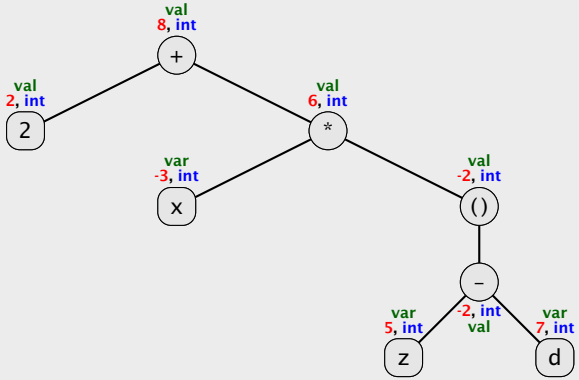
x -3 d 7 z 5

Beispiel: a = b = c = d = 0



a 0 b 0 c 0 d 0

Beispiel: 2 + x * (z - d)

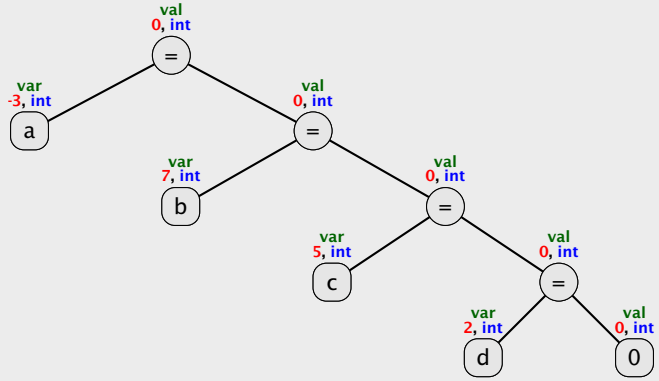


x -3 d 7 z 5

Beispiel: a != 0 && b/a < 10

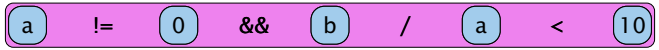
a	!=	0	&&	b	/	a	<	10

Beispiel: a = b = c = d = 0

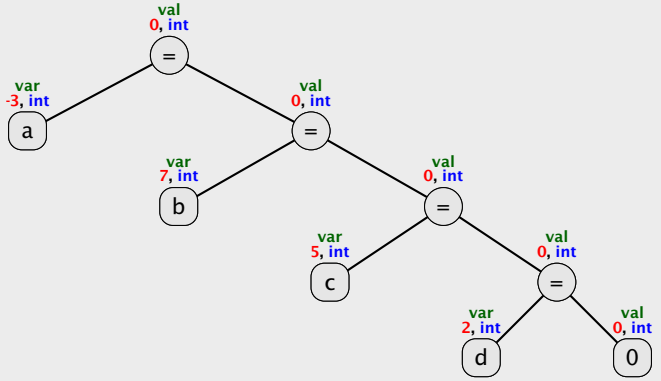


a 0 b 0 c 0 d 0

Beispiel: a != 0 && b/a < 10

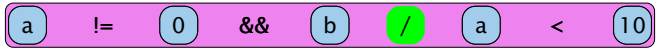


Beispiel: a = b = c = d = 0

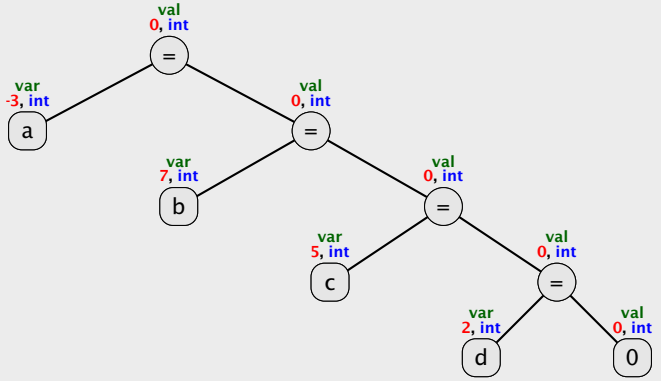


a 0 b 0 c 0 d 0

Beispiel: a != 0 && b/a < 10

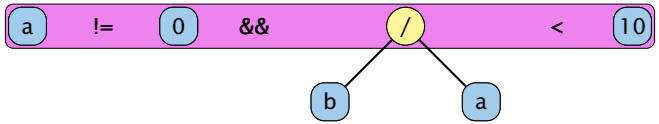


Beispiel: a = b = c = d = 0

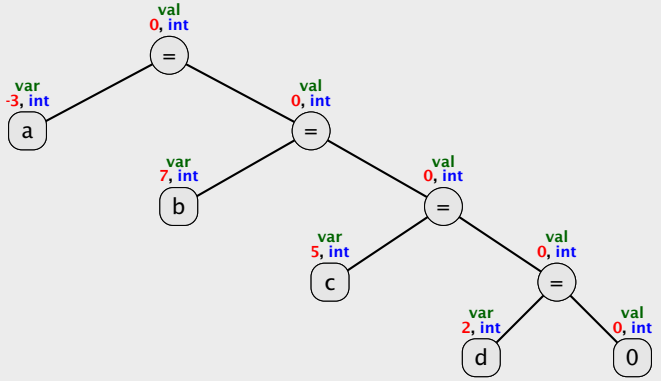


a 0 b 0 c 0 d 0

Beispiel: a != 0 && b/a < 10

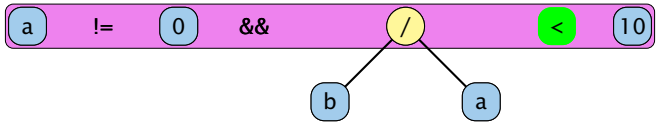


Beispiel: a = b = c = d = 0

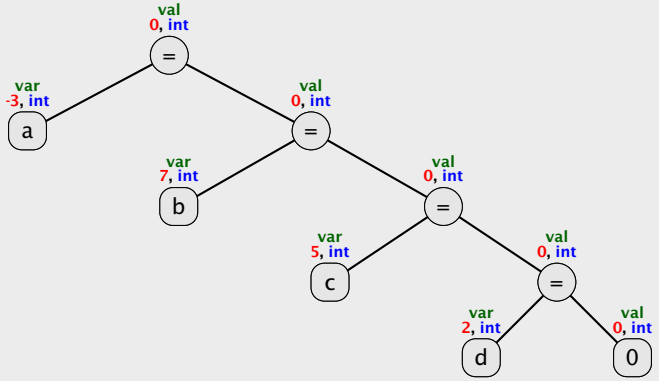


a 0 b 0 c 0 d 0

Beispiel: a != 0 && b/a < 10

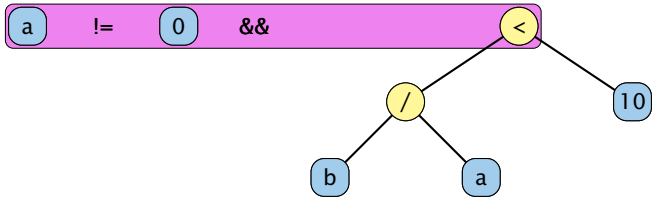


Beispiel: a = b = c = d = 0

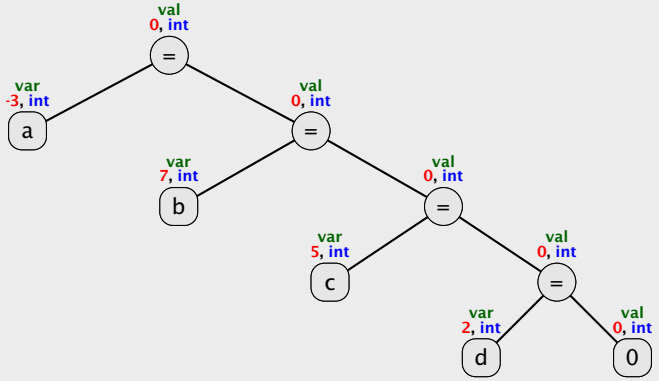


a 0 b 0 c 0 d 0

Beispiel: a != 0 && b/a < 10

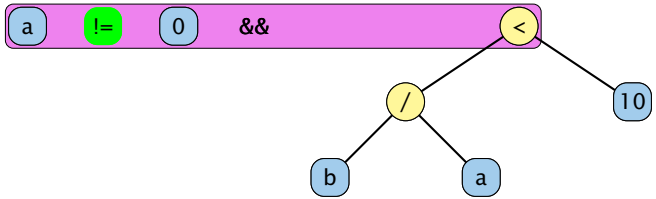


Beispiel: a = b = c = d = 0

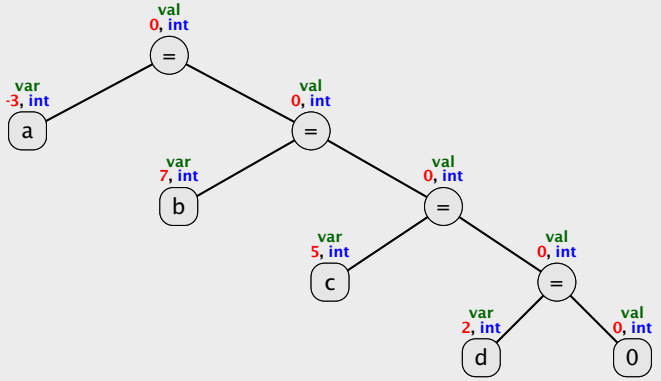


a 0 b 0 c 0 d 0

Beispiel: a != 0 && b/a < 10

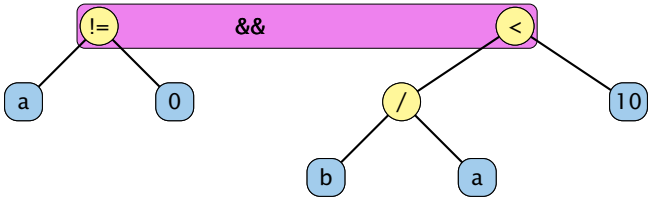


Beispiel: a = b = c = d = 0

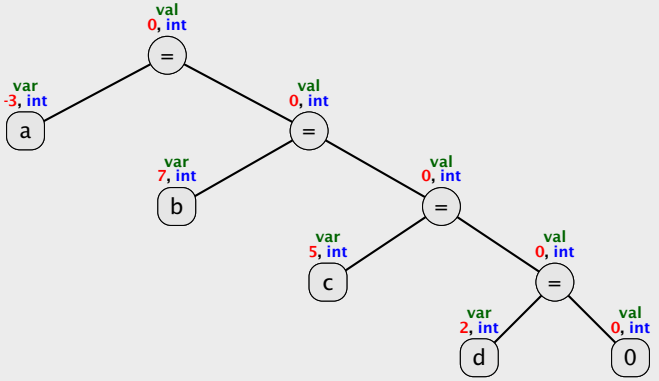


a 0 b 0 c 0 d 0

Beispiel: a != 0 && b/a < 10

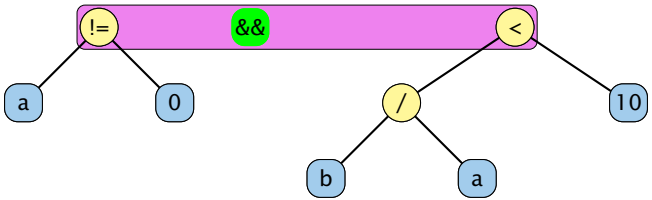


Beispiel: a = b = c = d = 0

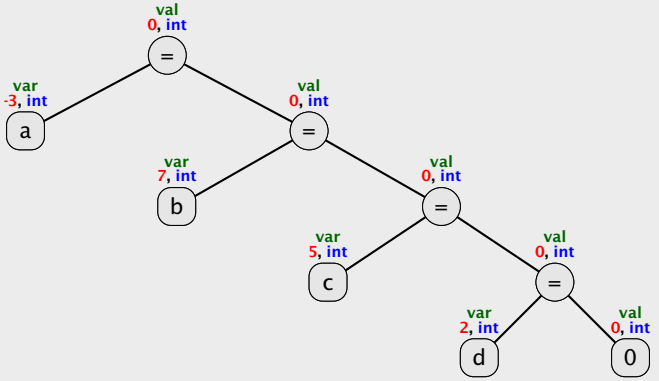


a 0 b 0 c 0 d 0

Beispiel: a != 0 && b/a < 10

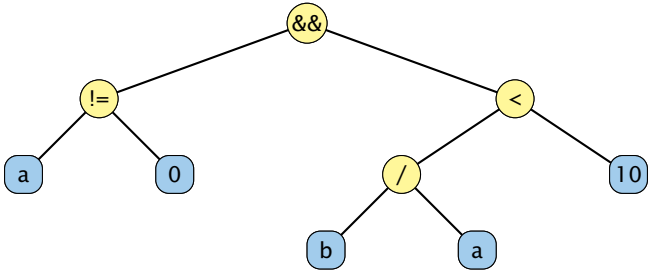


Beispiel: a = b = c = d = 0

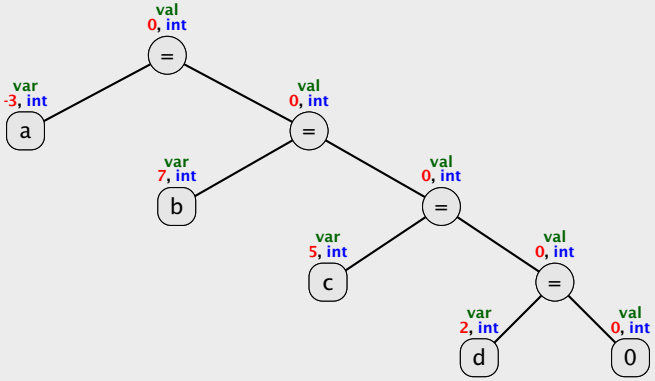


a 0 b 0 c 0 d 0

Beispiel: a != 0 && b/a < 10

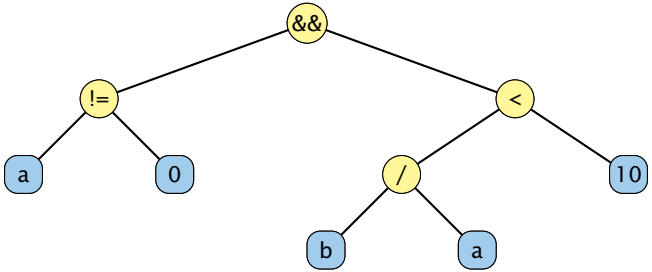


Beispiel: a = b = c = d = 0



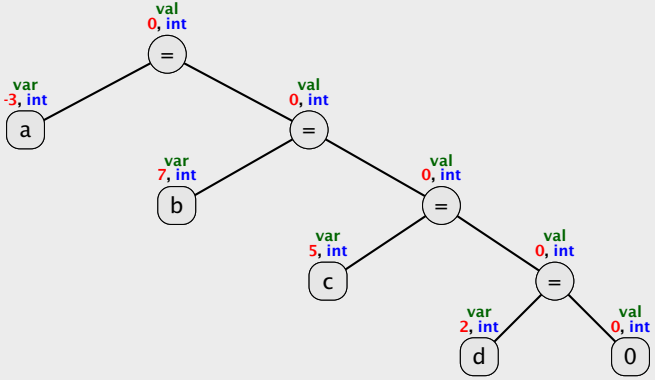
a 0 b 0 c 0 d 0

Beispiel: a != 0 && b/a < 10



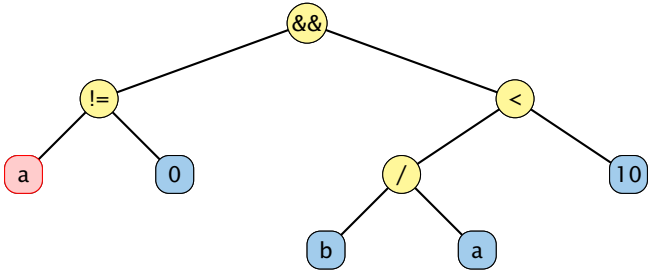
a 0 b 4

Beispiel: a = b = c = d = 0



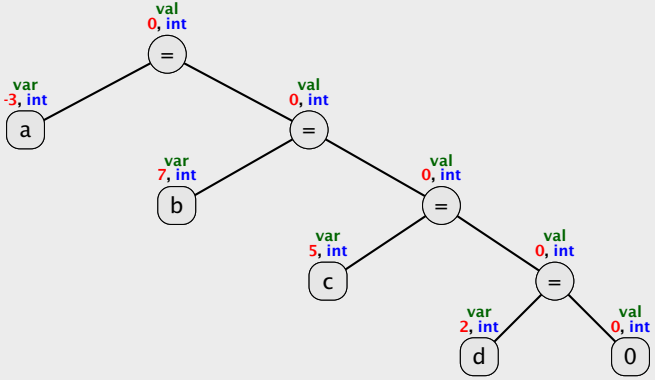
a 0 b 0 c 0 d 0

Beispiel: a != 0 && b/a < 10



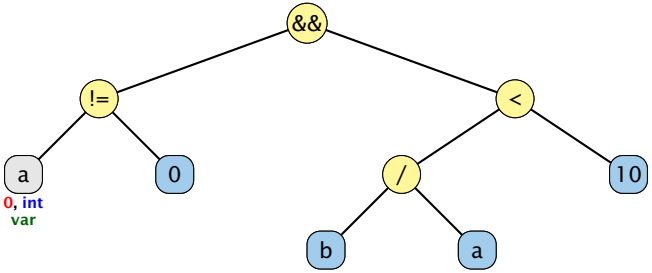
a 0 b 4

Beispiel: a = b = c = d = 0



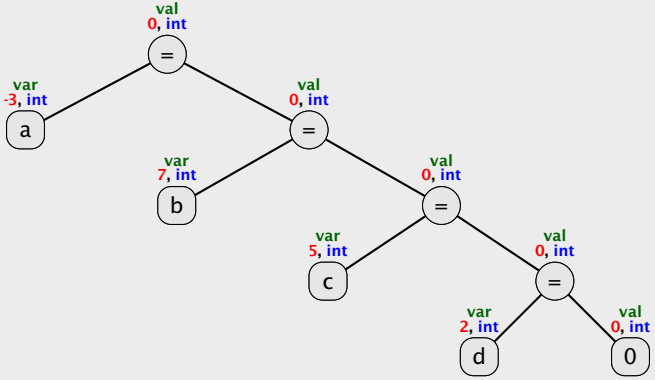
a 0 b 0 c 0 d 0

Beispiel: a != 0 && b/a < 10



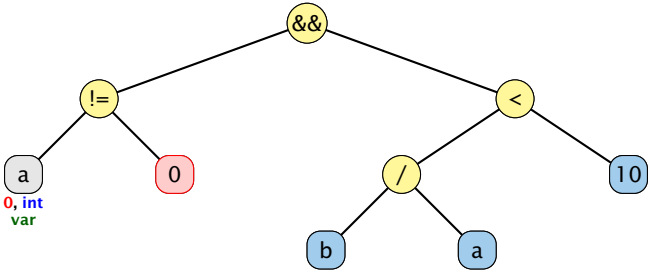
a 0 b 4

Beispiel: a = b = c = d = 0



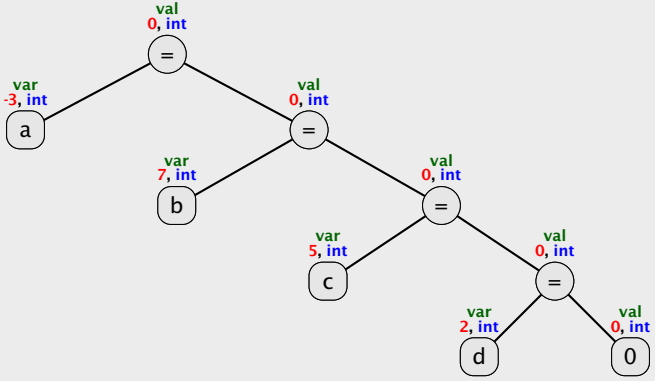
a 0 b 0 c 0 d 0

Beispiel: a != 0 && b/a < 10



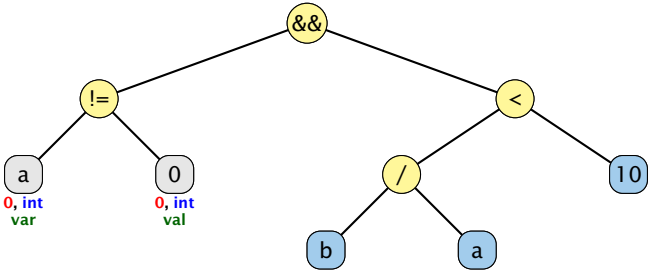
a 0 b 4

Beispiel: a = b = c = d = 0



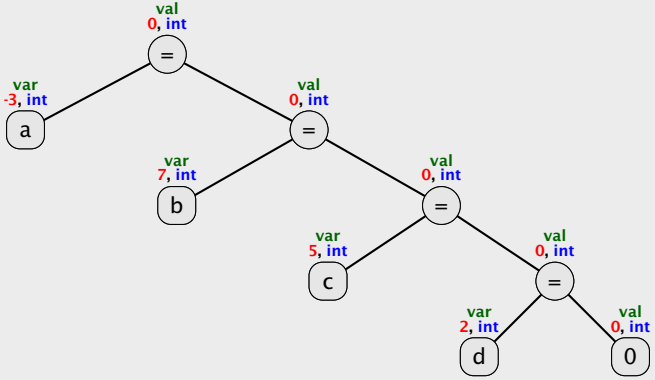
a 0 b 0 c 0 d 0

Beispiel: a != 0 && b/a < 10



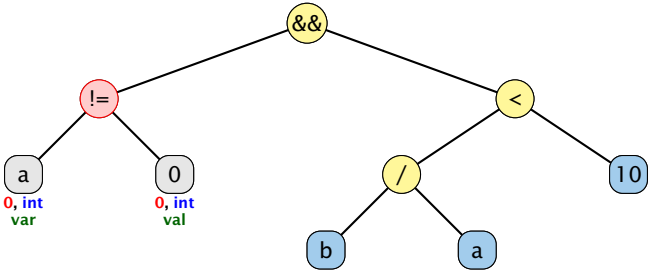
a 0 b 4

Beispiel: a = b = c = d = 0



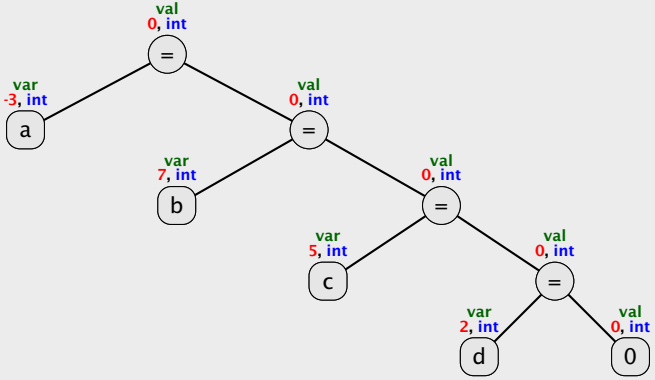
a 0 b 0 c 0 d 0

Beispiel: a != 0 && b/a < 10



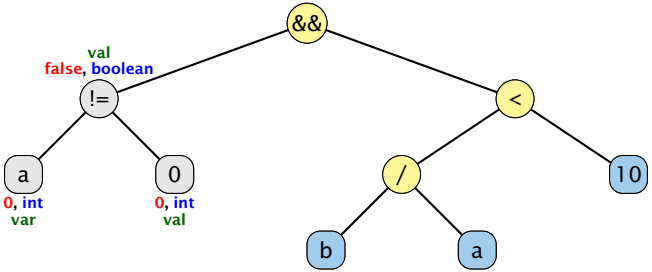
a 0 b 4

Beispiel: a = b = c = d = 0



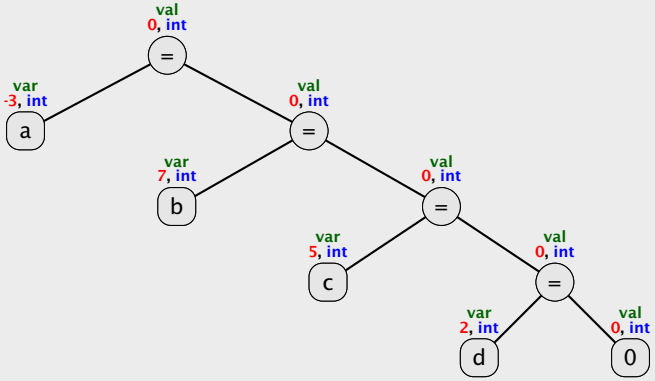
a 0 b 0 c 0 d 0

Beispiel: a != 0 && b/a < 10



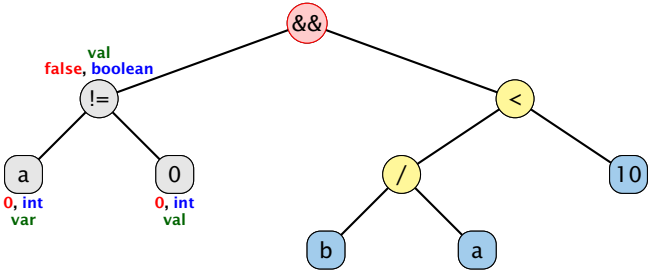
a 0 b 4

Beispiel: a = b = c = d = 0



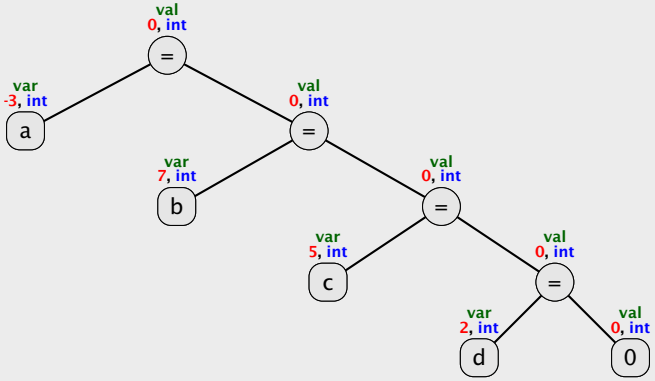
a 0 b 0 c 0 d 0

Beispiel: a != 0 && b/a < 10



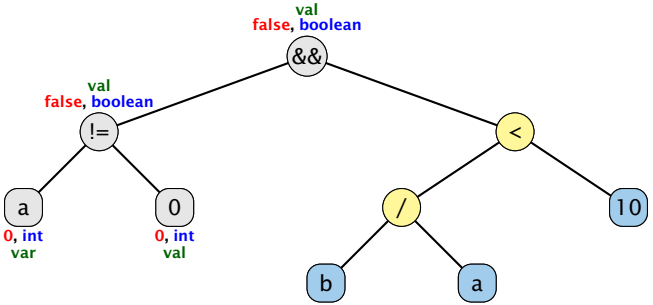
a 0 b 4

Beispiel: a = b = c = d = 0



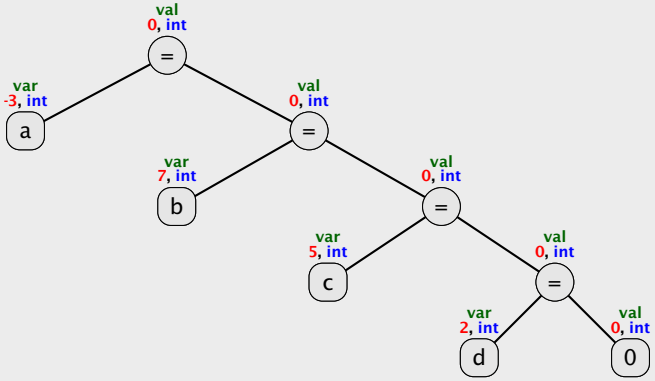
a 0 b 0 c 0 d 0

Beispiel: a != 0 && b/a < 10



a 0 b 4

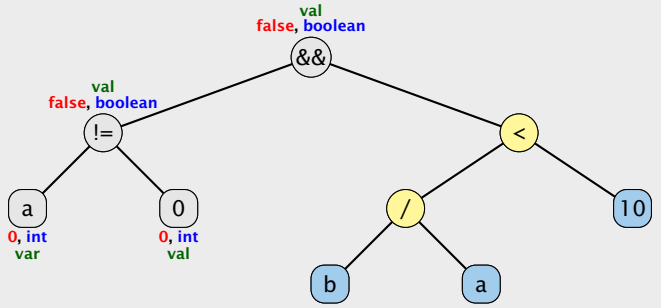
Beispiel: a = b = c = d = 0



a 0 b 0 c 0 d 0

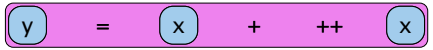
Beispiel: y = x + ++x

Beispiel: a != 0 && b/a < 10

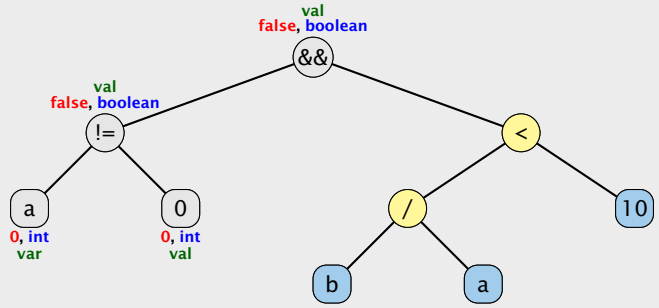


a 0 b 4

Beispiel: y = x + ++x

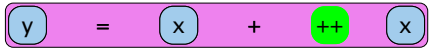


Beispiel: a != 0 && b/a < 10

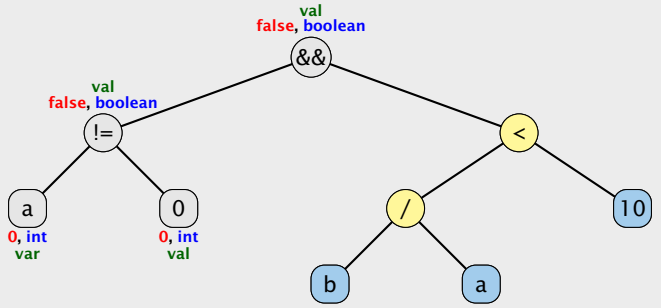


a 0 b 4

Beispiel: y = x + ++x

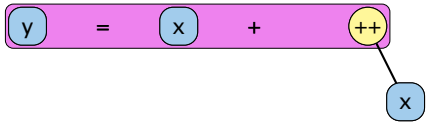


Beispiel: a != 0 && b/a < 10

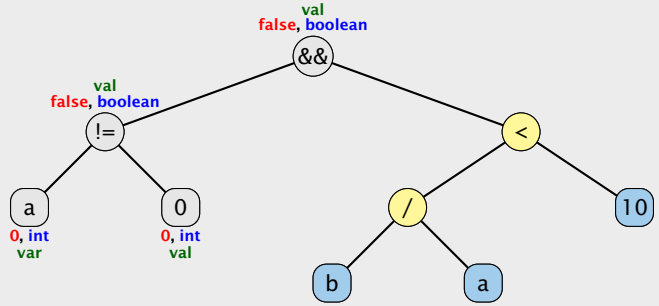


a 0 b 4

Beispiel: y = x + ++x

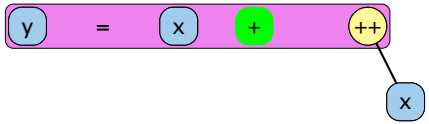


Beispiel: a != 0 && b/a < 10

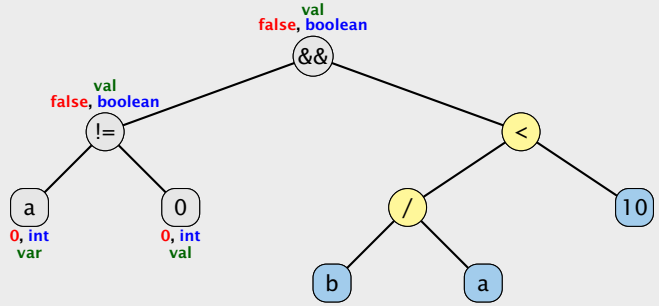


a 0 b 4

Beispiel: y = x + ++x

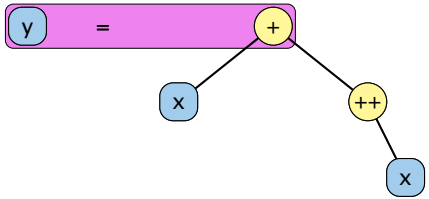


Beispiel: a != 0 && b/a < 10

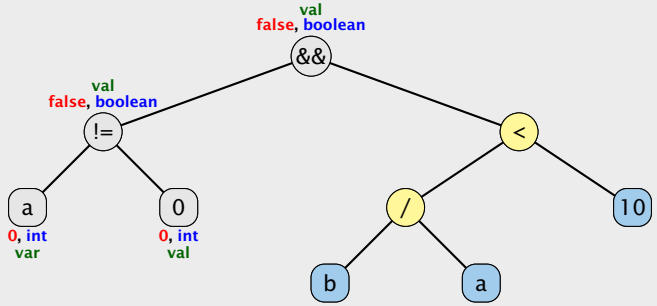


a 0 b 4

Beispiel: y = x + ++x

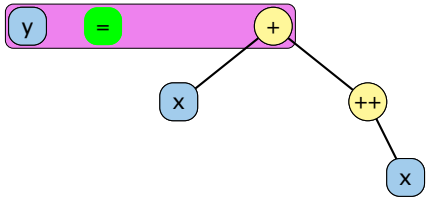


Beispiel: a != 0 && b/a < 10

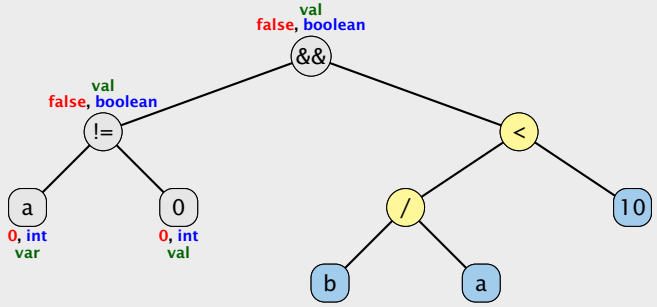


a 0 b 4

Beispiel: y = x + ++x

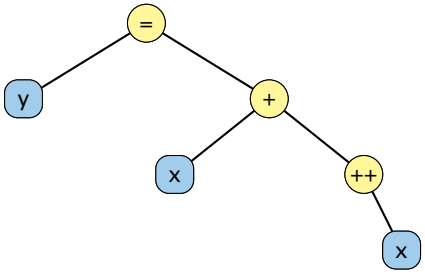


Beispiel: a != 0 && b/a < 10

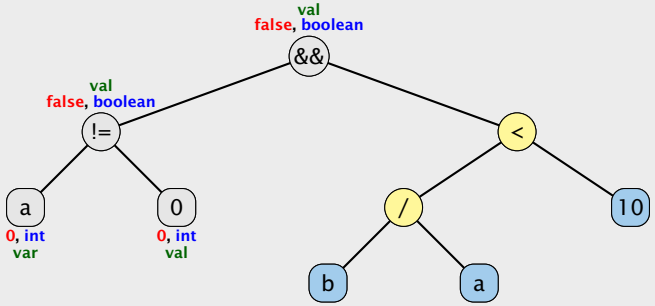


a 0 b 4

Beispiel: y = x + ++x

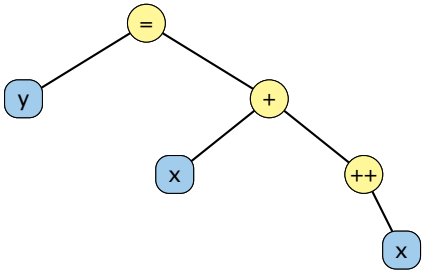


Beispiel: a != 0 && b/a < 10



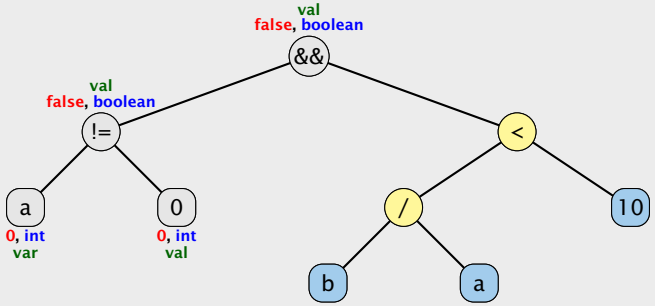
a 0 b 4

Beispiel: y = x + ++x



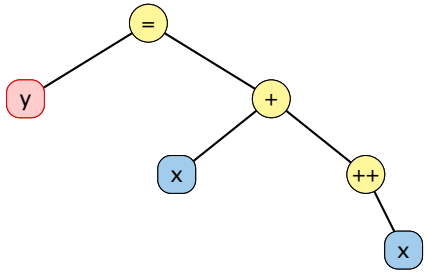
x 4 y 0

Beispiel: a != 0 && b/a < 10



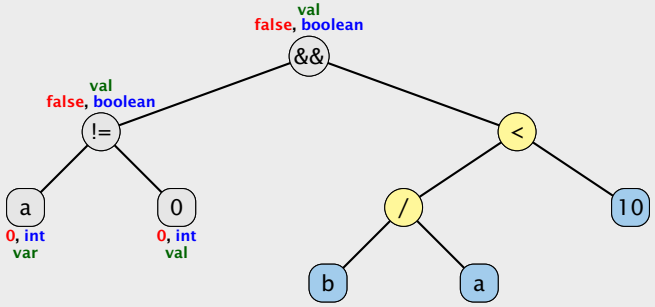
a 0 b 4

Beispiel: y = x + ++x



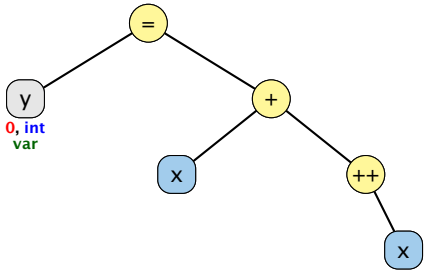
x 4 y 0

Beispiel: a != 0 && b/a < 10



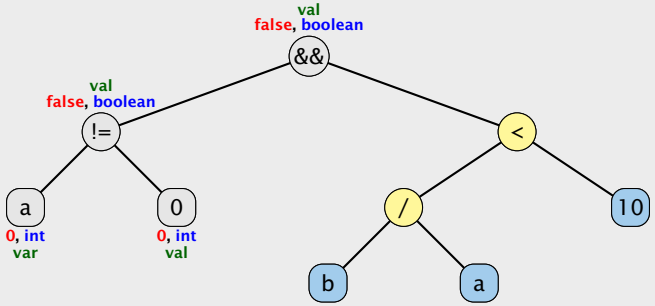
a 0 b 4

Beispiel: y = x + ++x



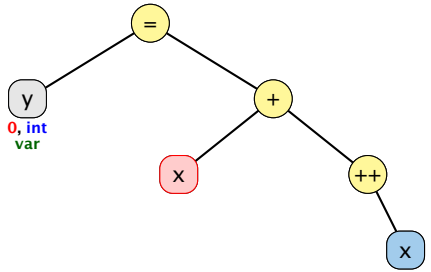
x 4 y 0

Beispiel: a != 0 && b/a < 10



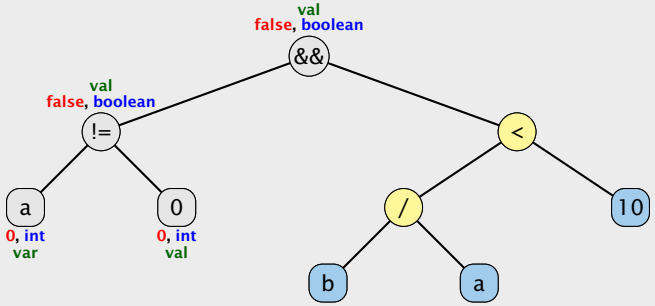
a 0 b 4

Beispiel: y = x + ++x



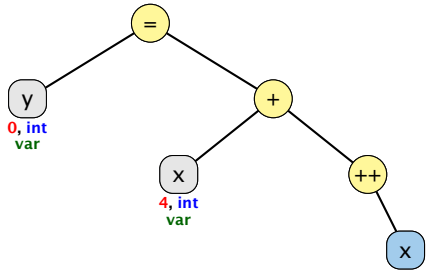
x 4 y 0

Beispiel: a != 0 && b/a < 10



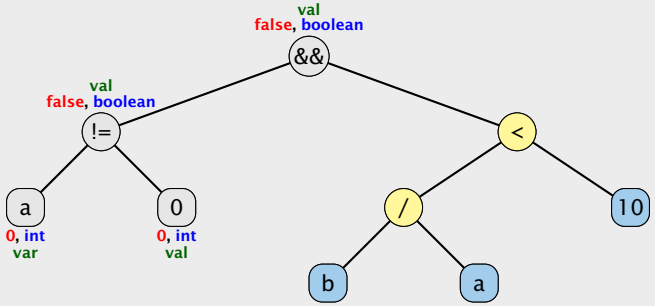
a 0 b 4

Beispiel: y = x + ++x



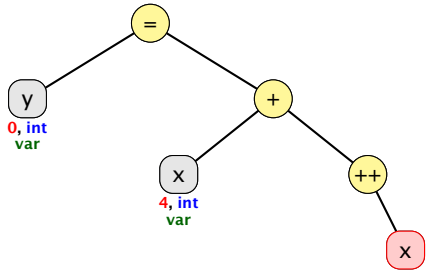
x 4 y 0

Beispiel: a != 0 && b/a < 10



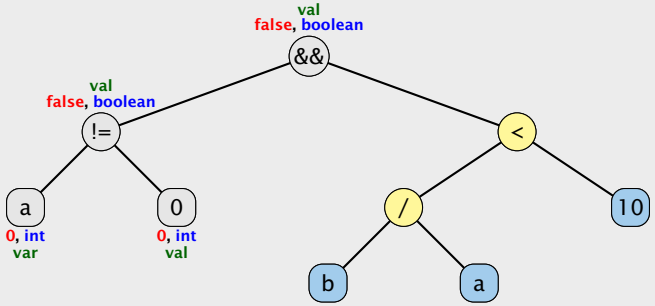
a 0 b 4

Beispiel: y = x + ++x



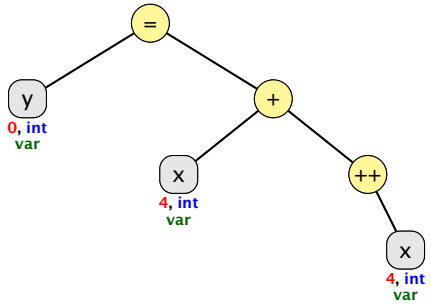
x 4 y 0

Beispiel: a != 0 && b/a < 10



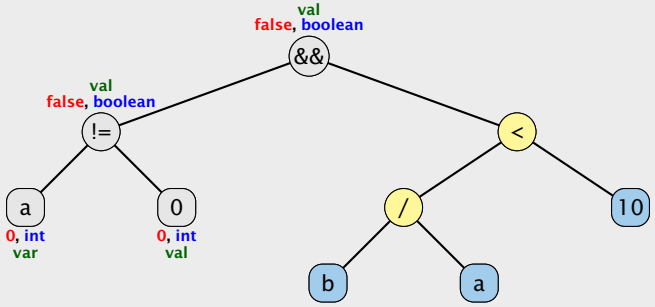
a 0 b 4

Beispiel: y = x + ++x



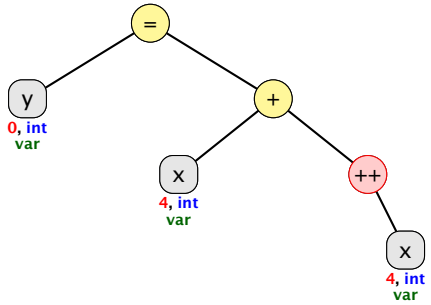
x 4 y 0

Beispiel: a != 0 && b/a < 10



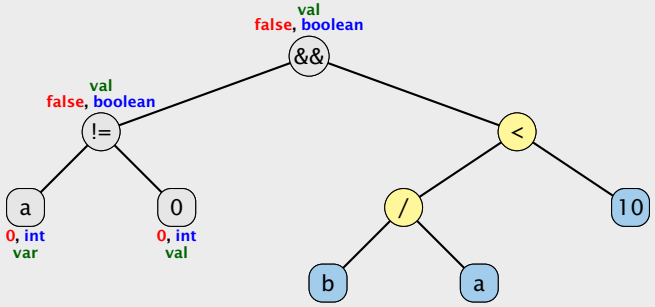
a 0 b 4

Beispiel: y = x + ++x



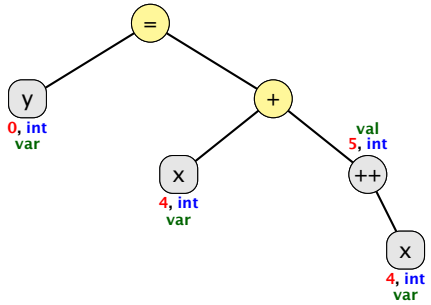
x 4 y 0

Beispiel: a != 0 && b/a < 10



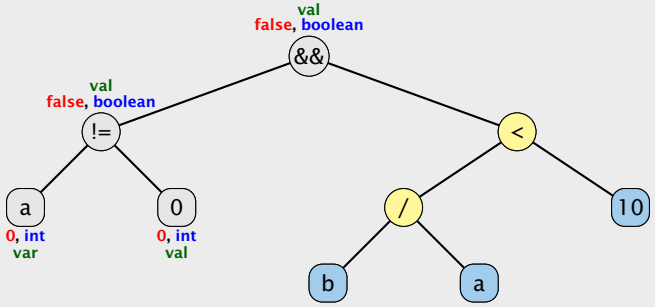
a 0 b 4

Beispiel: y = x + ++x



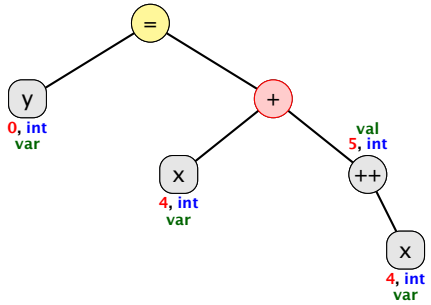
x 5 y 0

Beispiel: a != 0 && b/a < 10



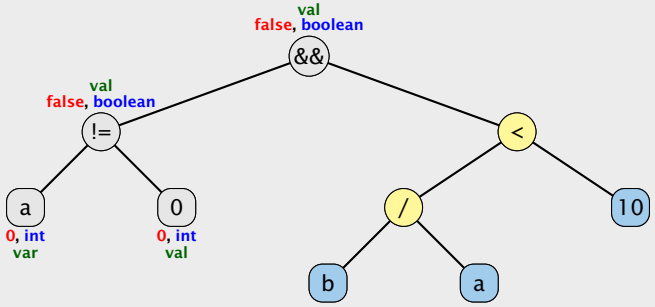
a 0 b 4

Beispiel: y = x + ++x



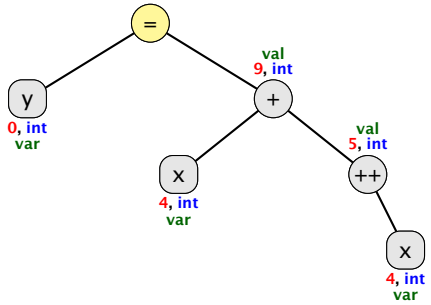
x 5 y 0

Beispiel: a != 0 && b/a < 10



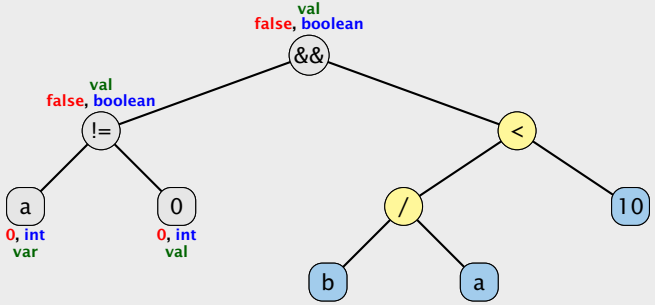
a 0 b 4

Beispiel: y = x + ++x



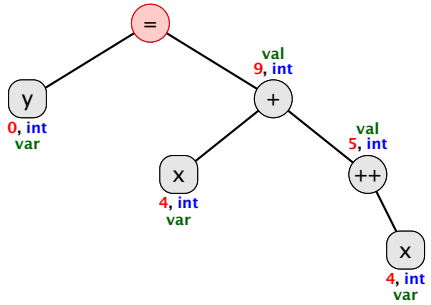
x 5 y 0

Beispiel: a != 0 && b/a < 10



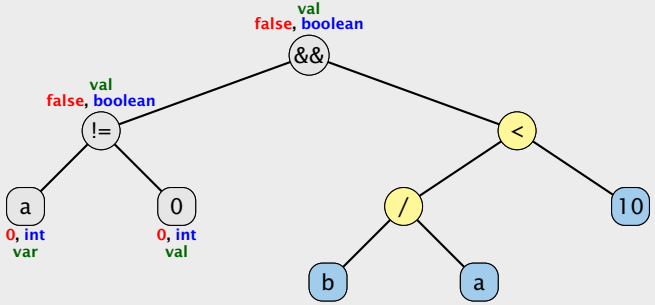
a 0 b 4

Beispiel: y = x + ++x



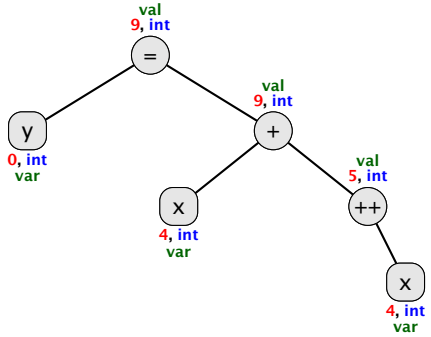
x 5 y 0

Beispiel: a != 0 && b/a < 10



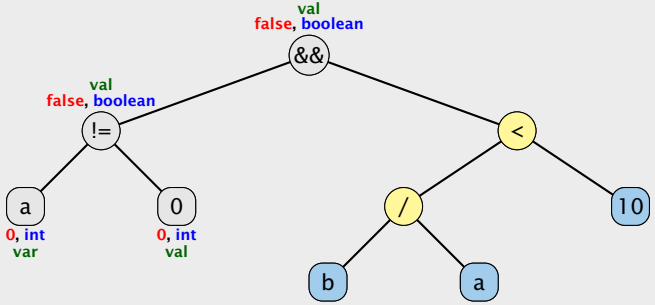
a 0 b 4

Beispiel: y = x + ++x



x 5 y 9

Beispiel: a != 0 && b/a < 10

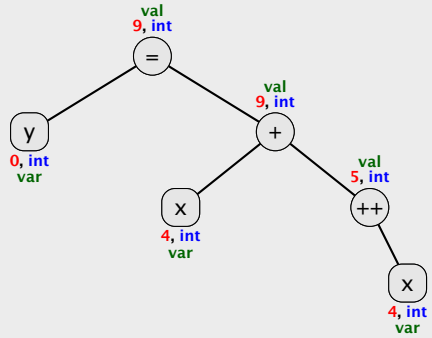


a 0 b 4

Beispiel: y = x++ + x

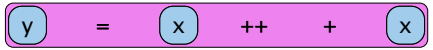
y	=	x	++	+	x

Beispiel: y = x + ++x

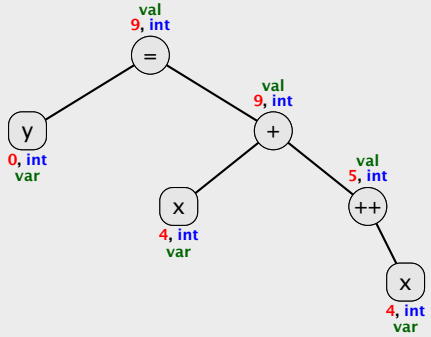


x 5 y 9

Beispiel: y = x++ + x



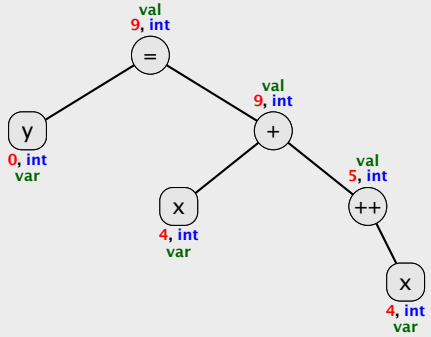
Beispiel: y = x + ++x



x 5 y 9

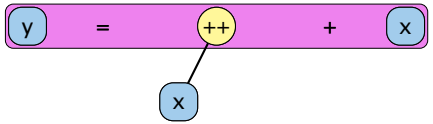
Beispiel: y = x++ + x

Beispiel: y = x + ++x

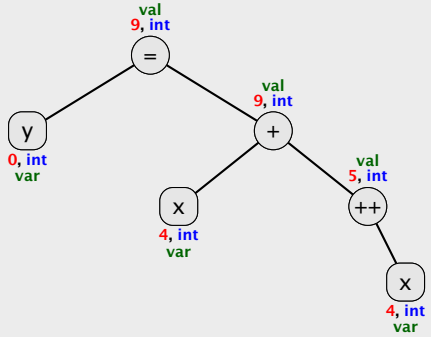


x 5 y 9

Beispiel: y = x++ + x

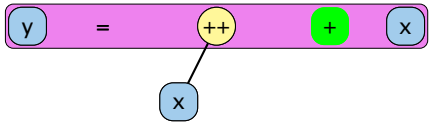


Beispiel: y = x + ++x

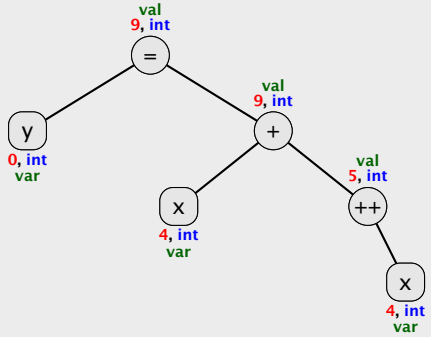


x 5 y 9

Beispiel: y = x++ + x

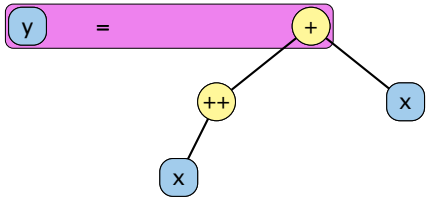


Beispiel: y = x + ++x

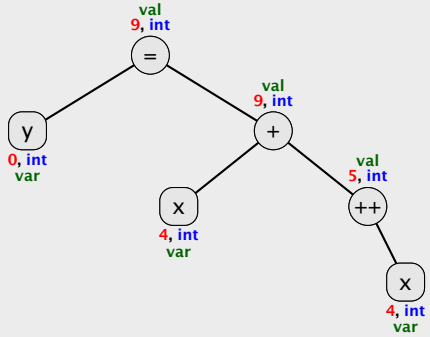


x 5 y 9

Beispiel: y = x++ + x

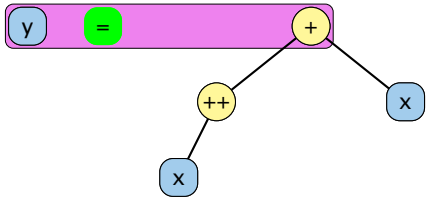


Beispiel: y = x + ++x

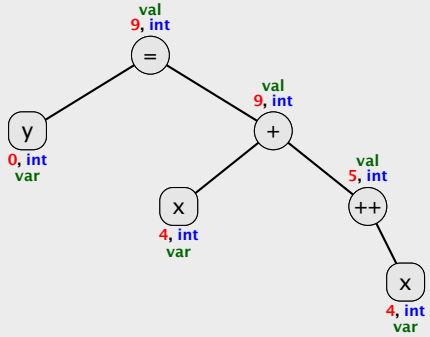


x 5 y 9

Beispiel: y = x++ + x

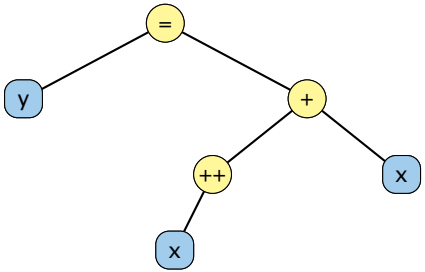


Beispiel: y = x + ++x

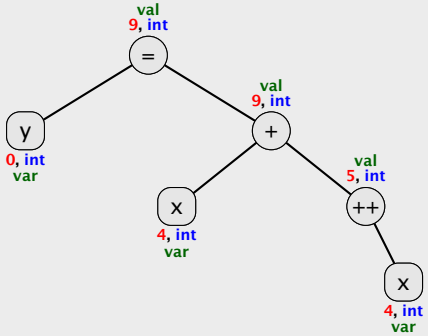


x 5 y 9

Beispiel: y = x++ + x

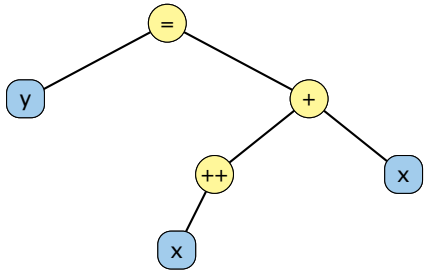


Beispiel: y = x + ++x



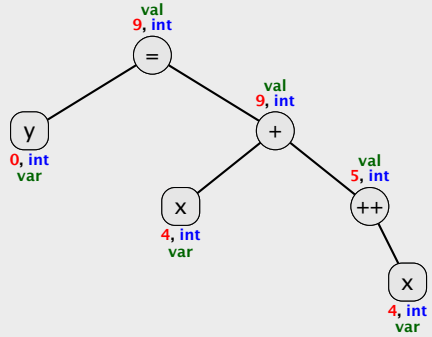
x 5 y 9

Beispiel: y = x++ + x



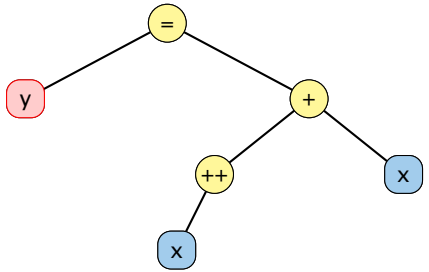
x 4 y 0

Beispiel: y = x + ++x



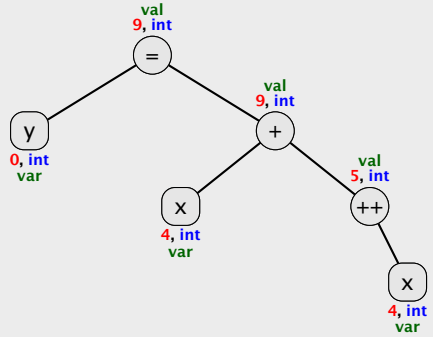
x 5 y 9

Beispiel: y = x++ + x



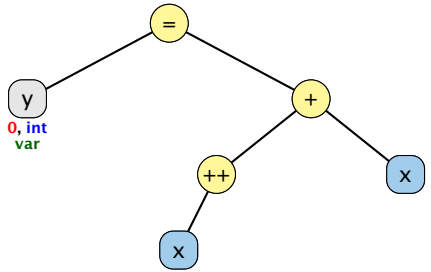
x 4 y 0

Beispiel: y = x + ++x



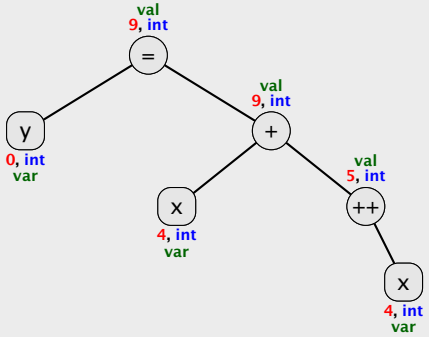
x 5 y 9

Beispiel: y = x++ + x



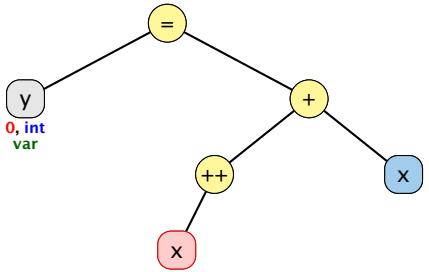
x 4 y 0

Beispiel: y = x + ++x



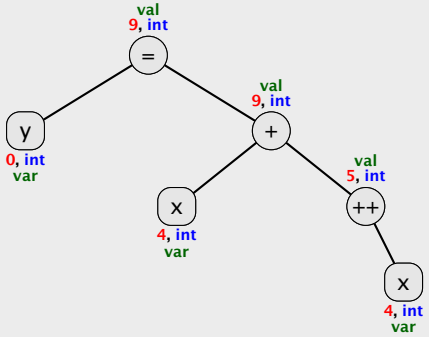
x 5 y 9

Beispiel: y = x++ + x



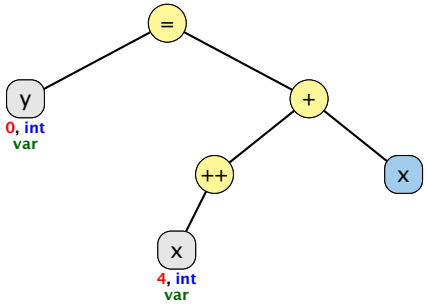
x 4 y 0

Beispiel: y = x + ++x



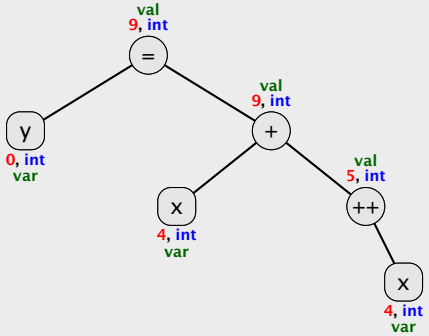
x 5 y 9

Beispiel: y = x++ + x



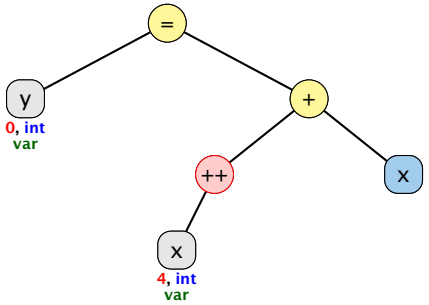
x 4 y 0

Beispiel: y = x + ++x



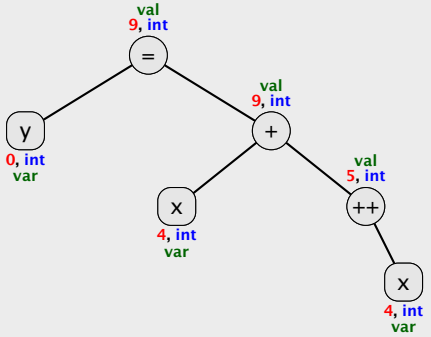
x 5 y 9

Beispiel: y = x++ + x



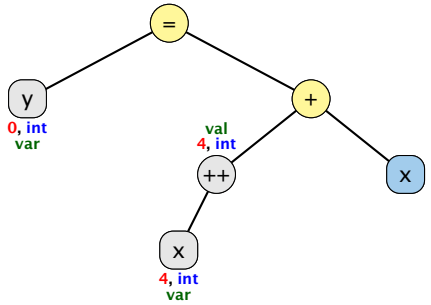
x 4 y 0

Beispiel: y = x + ++x



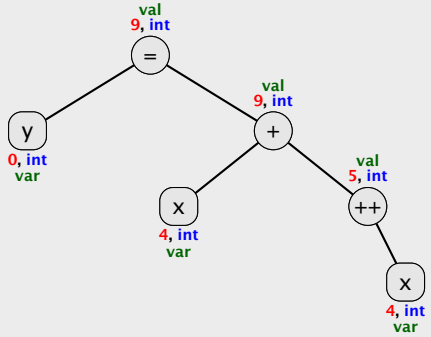
x 5 y 9

Beispiel: y = x++ + x



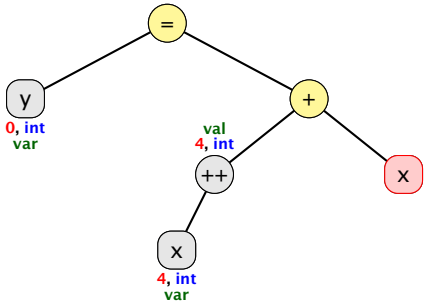
x 5 y 0

Beispiel: y = x + ++x



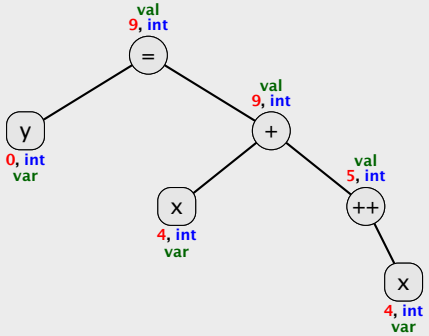
x 5 y 9

Beispiel: y = x++ + x



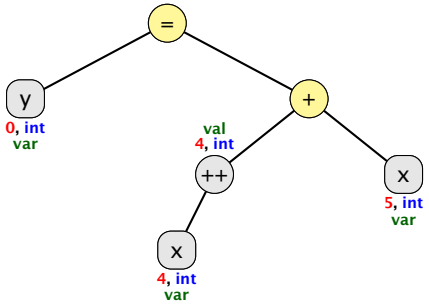
x 5 y 0

Beispiel: y = x + ++x



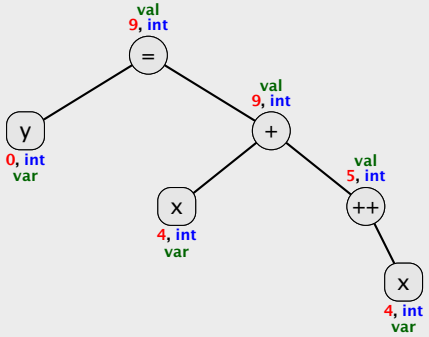
x 5 y 9

Beispiel: y = x++ + x



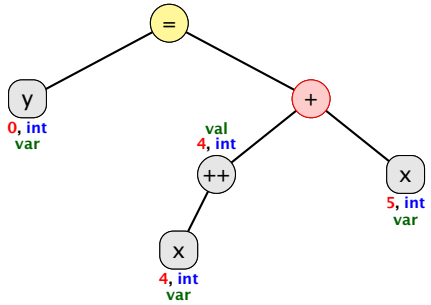
x 5 y 0

Beispiel: y = x + ++x



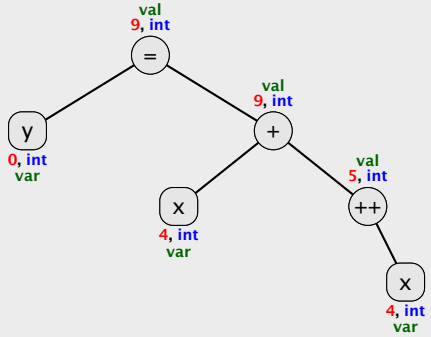
x 5 y 9

Beispiel: y = x++ + x



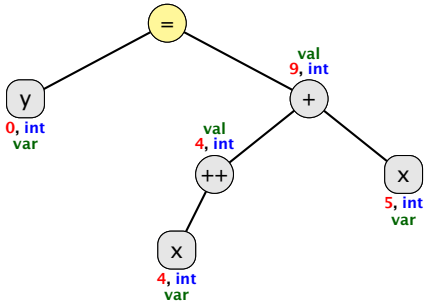
x 5 y 0

Beispiel: y = x + ++x



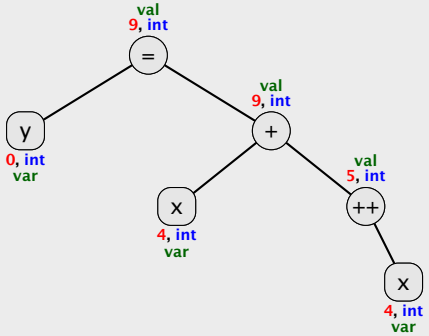
x 5 y 9

Beispiel: y = x++ + x



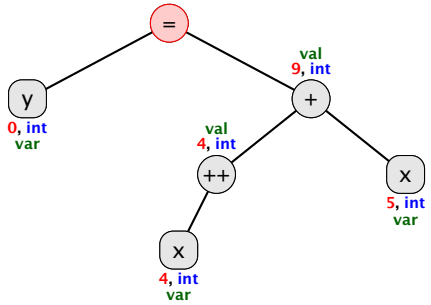
x 5 y 0

Beispiel: y = x + ++x



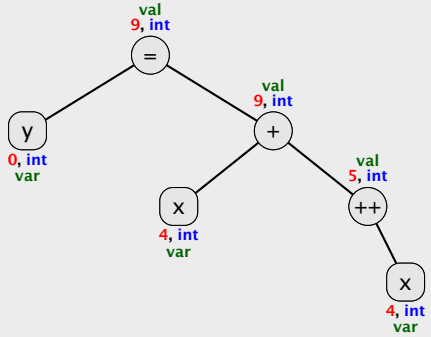
x 5 y 9

Beispiel: y = x++ + x



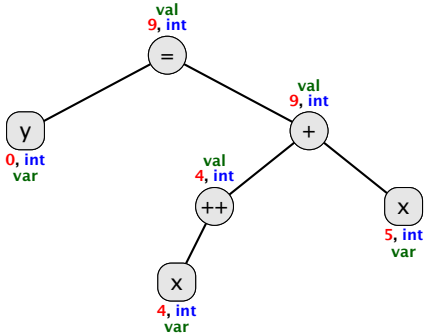
x 5 y 0

Beispiel: y = x + ++x



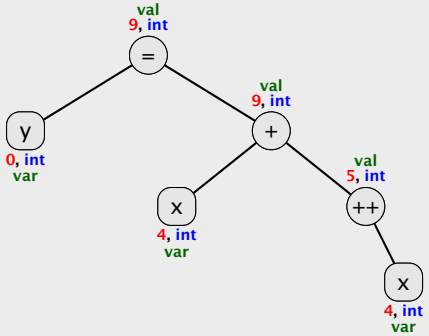
x 5 y 9

Beispiel: y = x++ + x



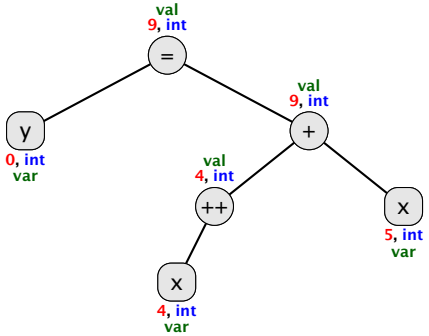
x 5 y 9

Beispiel: y = x + ++x



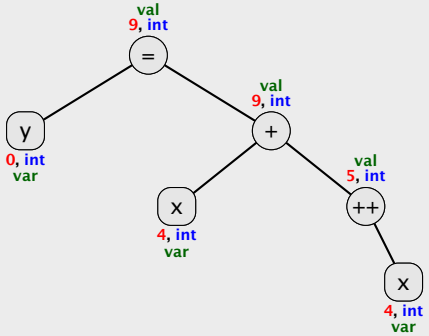
x 5 y 9

Beispiel: y = x++ + x



x 5 y 9

Beispiel: y = x + ++x



x 5 y 9

Impliziter Typecast

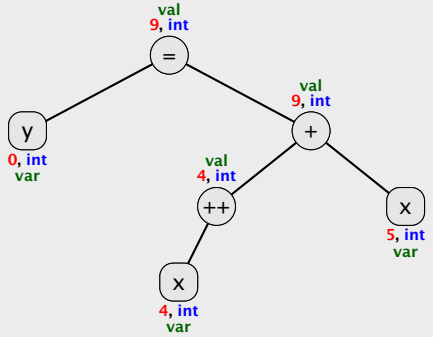
Wenn ein Ausdruck vom TypA an einer Stelle verwendet wird, wo ein Ausdruck vom TypB erforderlich ist, wird

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

Beispiel: Zuweisung

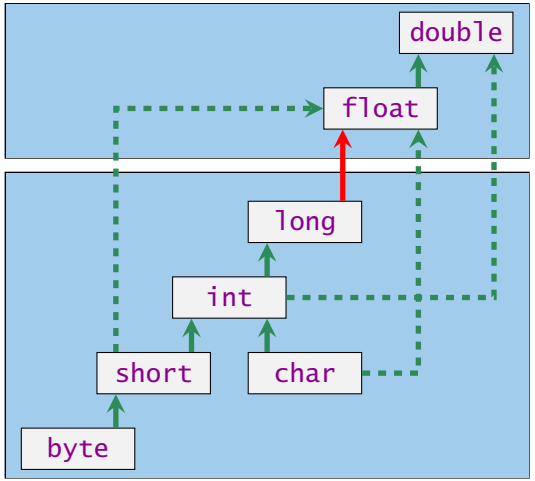
```
long x = 5;
int  y = 3;
x = y; // impliziter Cast von int nach long
```

Beispiel: y = x++ + x



x 5 y 9

Erlaubte Implizite Typecasts – Numerische Typen



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

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 gecasted (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
```

Welcher Typ wird benötigt?

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

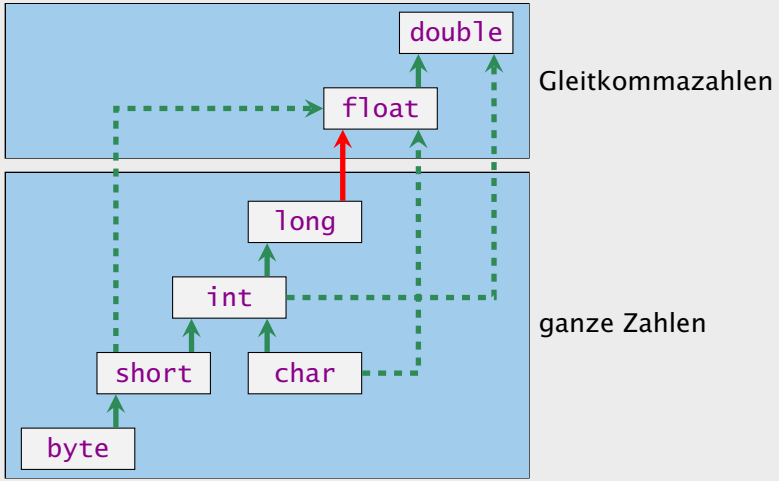
```
+ : int ⟶ int
+ : long ⟶ long
+ : float ⟶ float
+ : double ⟶ double

+ : int × int ⟶ int
+ : long × long ⟶ long
+ : float × float ⟶ float
+ : double × double ⟶ double

+ : String × String ⟶ String
```

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

Erlaubte Implizite Typecasts – Numerische Typen



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

Impliziter Typecast

Der Compiler wertet nur die Typen des Ausdrucksbaums aus.

- Für jeden inneren Knoten wählt er dann die geeignete Funktion (z.B. `+ : long` $\times$ `long` $\longrightarrow$ `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. `min(int a, int b)` und `min(long a, long b)`).
- Der Compiler nimmt die Funktion mit der speziellsten Signatur.

Welcher Typ wird benötigt?

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

```
+ : int ⟶ int
+ : long ⟶ long
+ : float ⟶ float
+ : double ⟶ double

+ : int × int ⟶ int
+ : long × long ⟶ long
+ : float × float ⟶ float
+ : double × double ⟶ double

+ : String × String ⟶ String
```

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

Speziellste Signatur

Impliziter Typecast

Der Compiler wertet nur die Typen des Ausdrucksbaums aus.

- Für jeden inneren Knoten wählt er dann die geeignete Funktion (z.B. `+ : long` $\times$ `long` $\longrightarrow$ `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. `min(int a, int b)` und `min(long a, long b)`).
- Der Compiler nimmt die Funktion mit der speziellsten Signatur.

Ordnungsrelationen

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

- **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 der Menge der k-Tupel von Typen.

Speziellste Signatur

Ordnungsrelationen

Relation $\preceq$: TypA $\preceq$ 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.

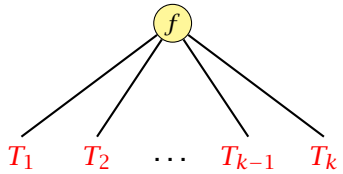
Relation $\preceq_k$: $(T_1, \ldots, T_k) \preceq_k (T_1', \ldots, T_k')$ falls $T_i \preceq T_i'$ für alle $i \in \{1, \ldots, k\}$.

- **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 der Menge der k-Tupel von Typen.

Speziellste Signatur

$R_1 \quad f(\mathcal{T}_1)$
$R_2 \quad f(\mathcal{T}_2)$
$\vdots$
$R_\ell \quad f(\mathcal{T}_\ell)$



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

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

Menge aller möglichen Funktionen/Tupel:

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

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

Ordnungsrelationen

Relation $\preceq$: TypA $\preceq$ 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:** $T_1 \preceq T_2 \wedge T_2 \preceq T_1 \Rightarrow T_1 = T_2$

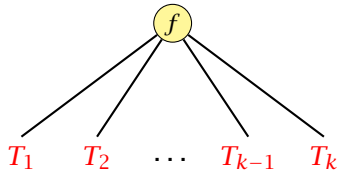
d.h., $\preceq$ definiert Halbordnung auf der Menge der Typen.

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

- **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 der Menge der k-Tupel von Typen.

$R_1 \quad f(\mathcal{T}_1)$
$R_2 \quad f(\mathcal{T}_2)$
$\vdots$
$R_\ell \quad f(\mathcal{T}_\ell)$



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

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

Menge aller möglichen Funktionen/Tupel:

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

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

Ordnungsrelationen

Relation $\preceq$: TypA $\preceq$ 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:** $T_1 \preceq T_2 \wedge T_2 \preceq T_1 \Rightarrow T_1 = T_2$

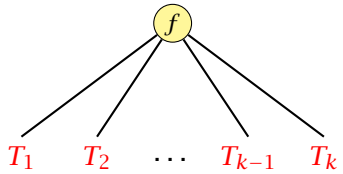
d.h., $\preceq$ definiert Halbordnung auf der Menge der Typen.

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

- **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 der Menge der k-Tupel von Typen.

$$\begin{array}{ll} R_1 & f(\mathcal{T}_1) \\ R_2 & f(\mathcal{T}_2) \\ & \vdots \\ R_\ell & f(\mathcal{T}_\ell) \end{array}$$



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

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

Menge aller möglichen Funktionen/Tupel:

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

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

Ordnungsrelationen

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

- **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 der Menge der k-Tupel von Typen.

Impliziter Typecast – Numerische Typen

Angenommen wir haben Funktionen

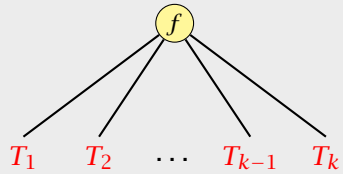
```
int min(int a, int b)
float min(float a, float b)
double min(double a, double b)
```

definiert.

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

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

$$\begin{array}{ll} R_1 & f(\mathcal{T}_1) \\ R_2 & f(\mathcal{T}_2) \\ & \vdots \\ R_\ell & f(\mathcal{T}_\ell) \end{array}$$



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

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

Menge aller möglichen Funktionen/Tupel:

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

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

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

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

Impliziter Typecast

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

Beispiel: Zuweisungen

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

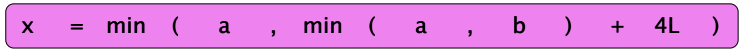
Es wird nur der Parameter konvertiert, der nicht dem Seiteneffekt unterliegt.

5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

symbol	*name*	*types*	*L/R*	*level*
`()`	Funktionsaufruf	Funktionsname, *	links	1

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



5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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

5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

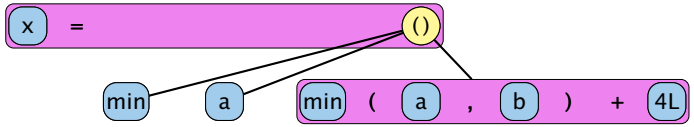
Beispiel: x = min(a, min(a,b) + 4L)

5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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

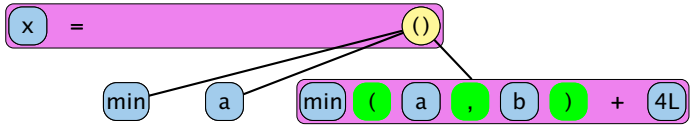


5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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

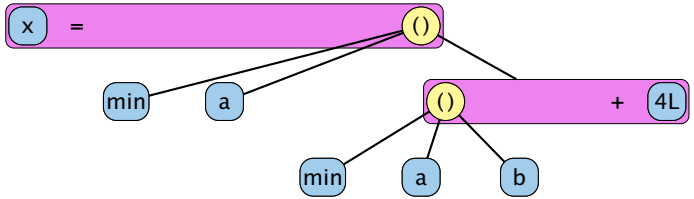


5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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

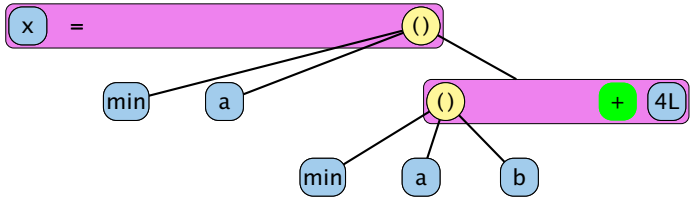


5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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

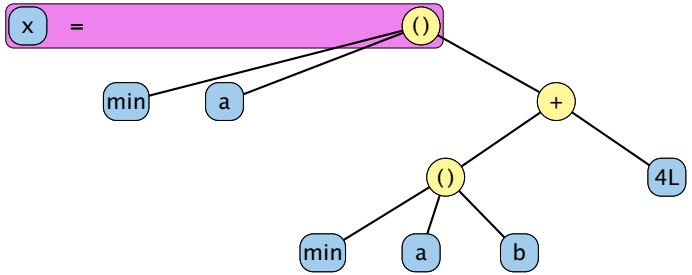


5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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

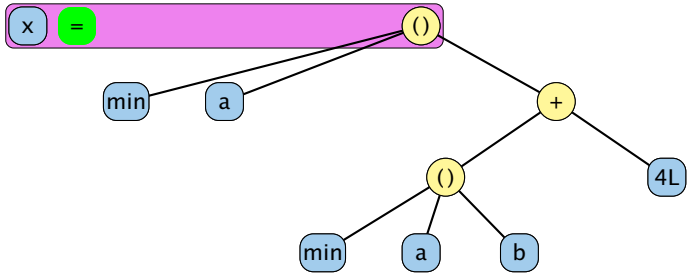


5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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

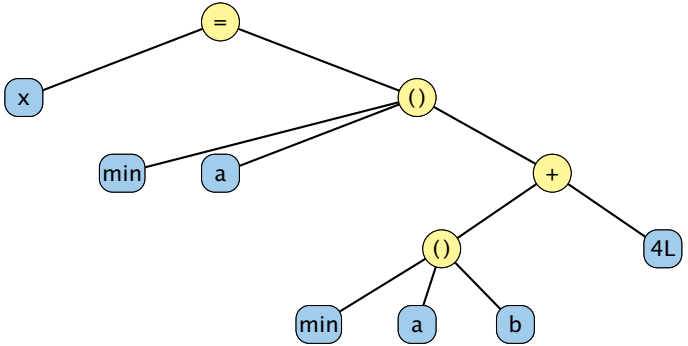


5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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




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

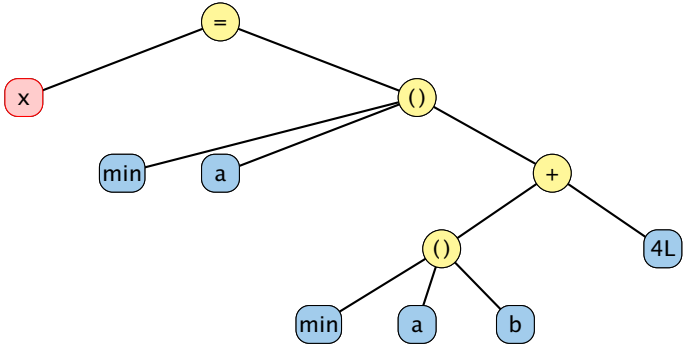
long x 7 int a 3 int b 5

5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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




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

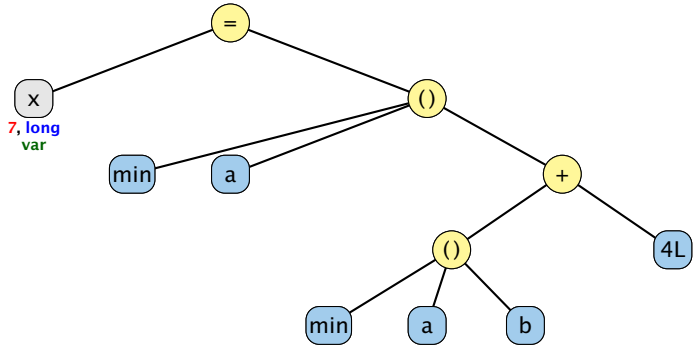
long x 7 int a 3 int b 5

5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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




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

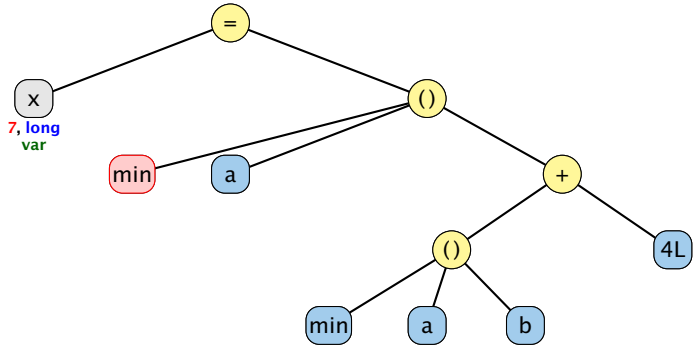
long x 7 int a 3 int b 5

5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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




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

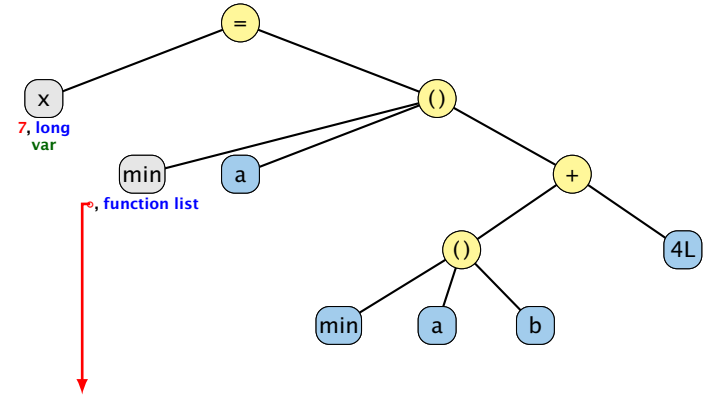
long x 7 int a 3 int b 5

5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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

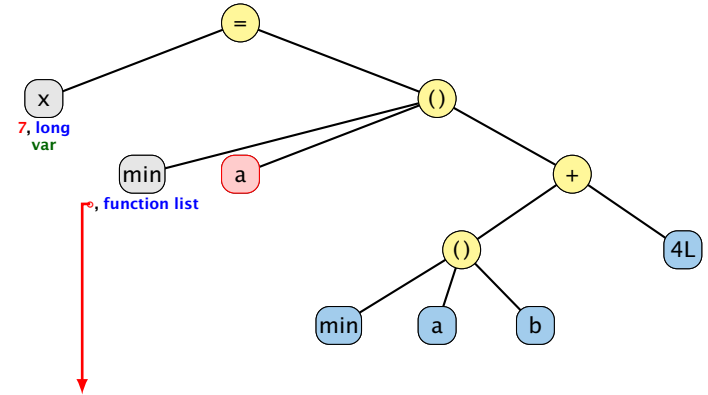


5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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

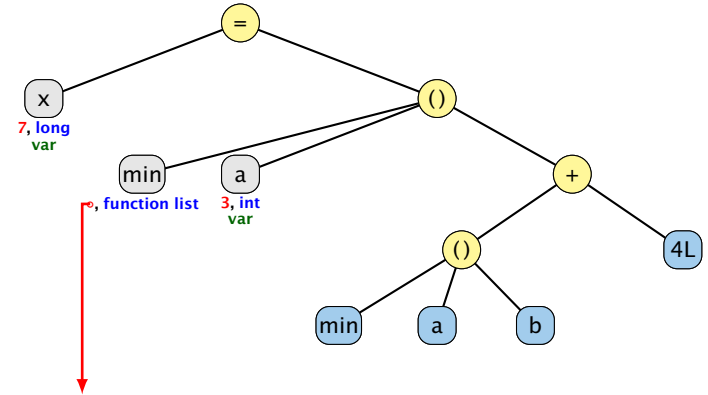


5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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

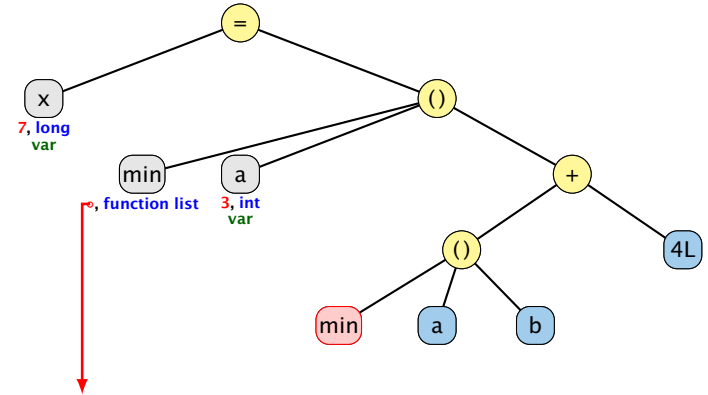


5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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

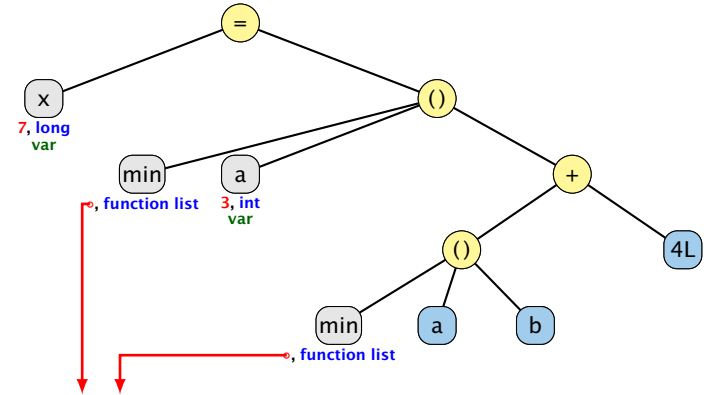


5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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

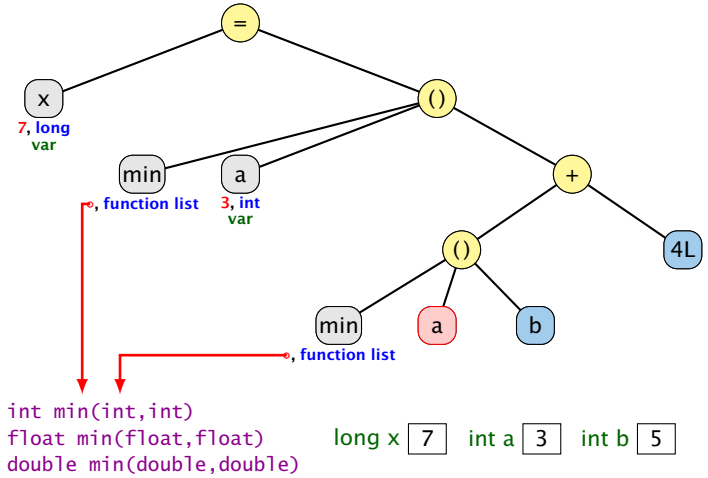


5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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

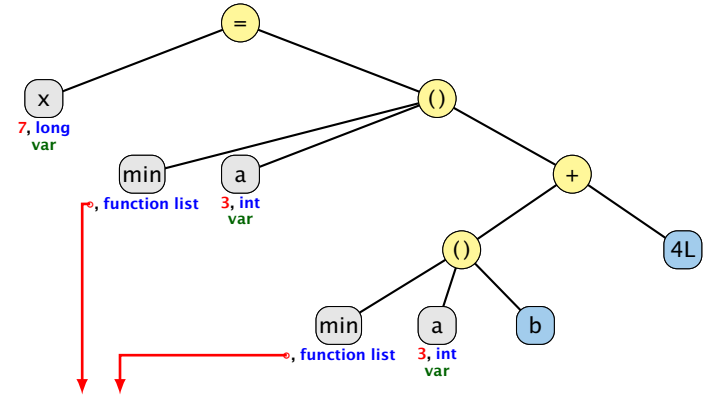


5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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

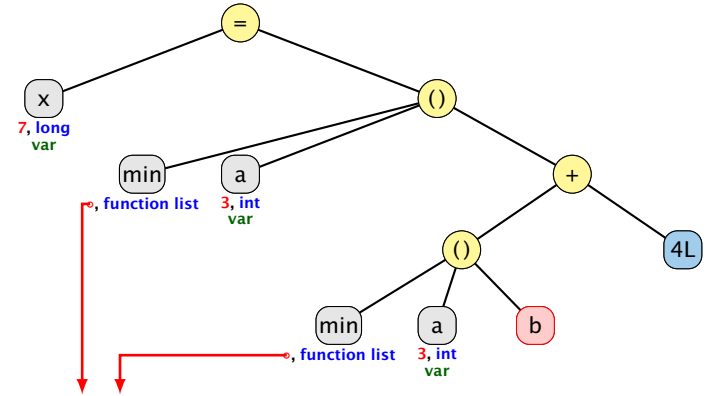


5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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

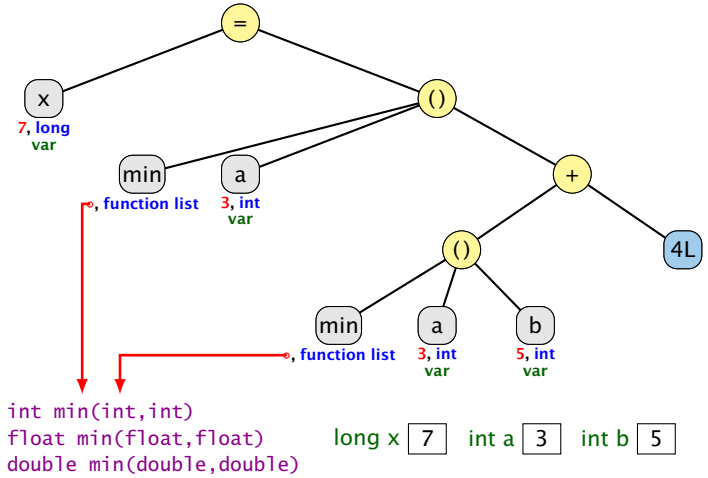


5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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

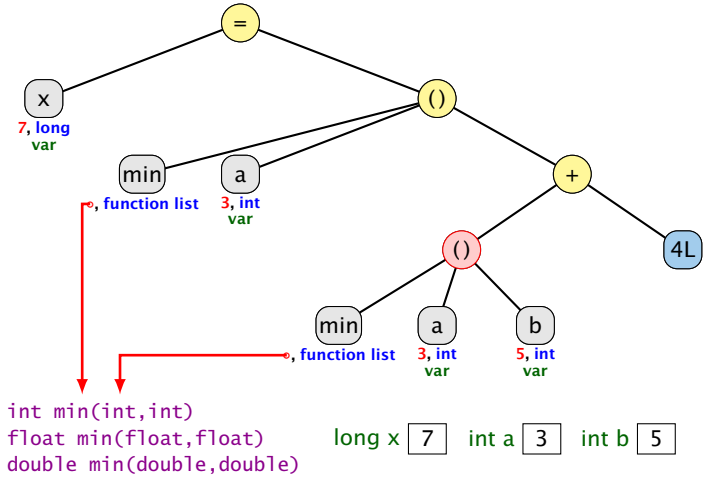


5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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

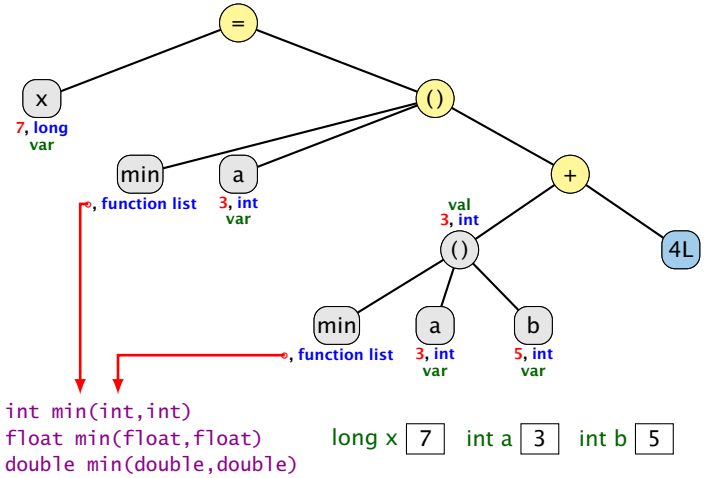


5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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

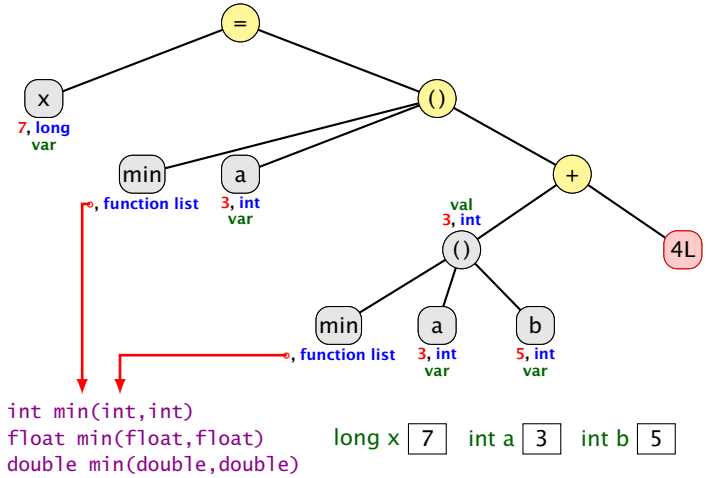


5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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

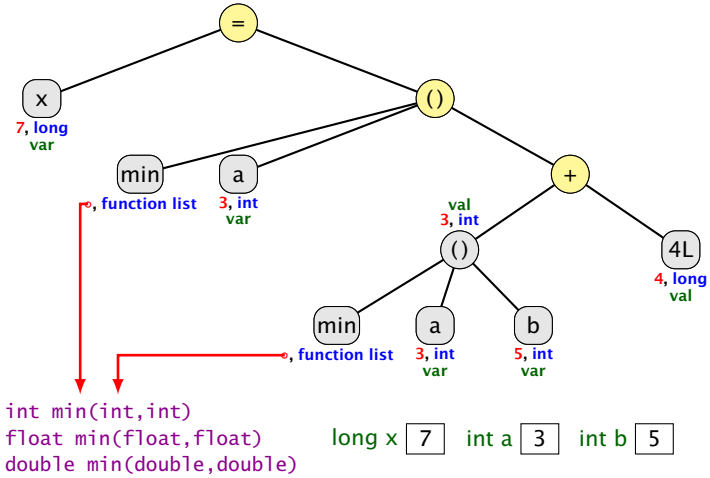


5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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

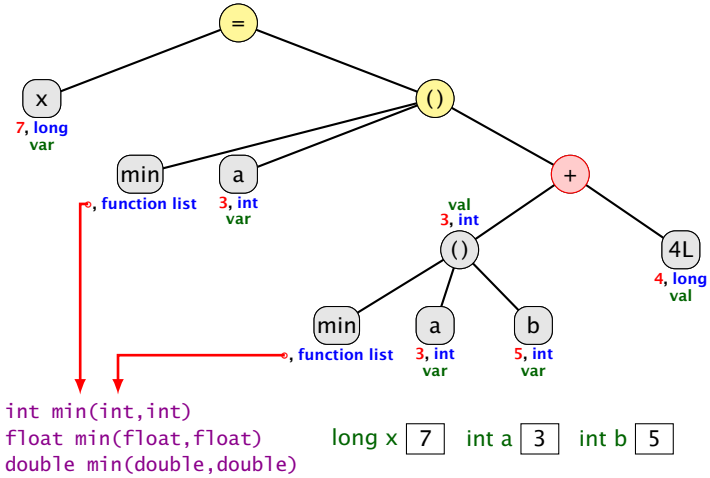


5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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

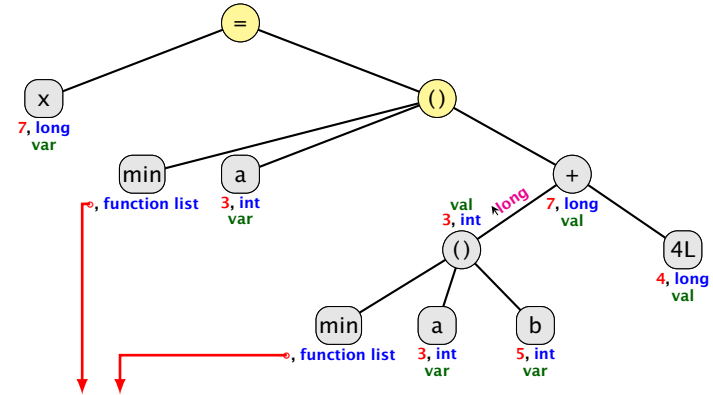


5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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

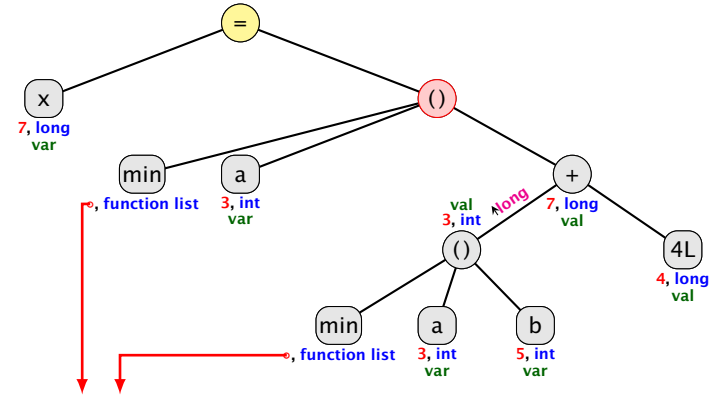


5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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

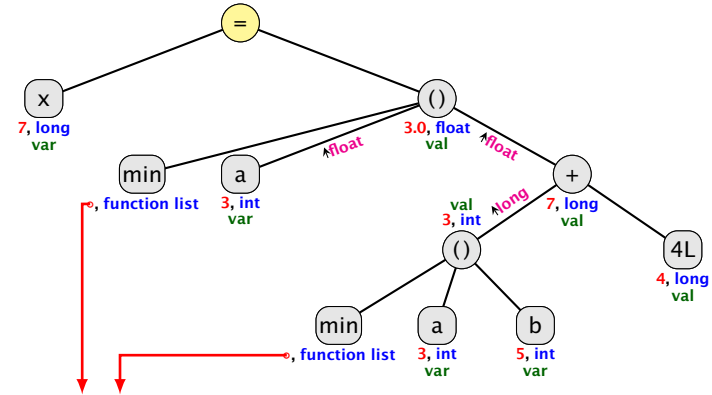


5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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

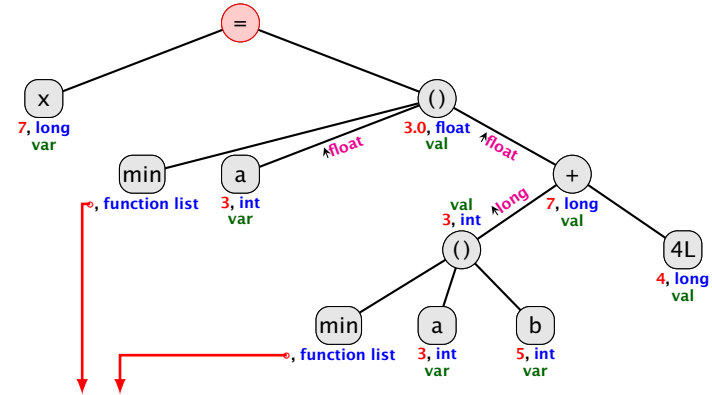


5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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

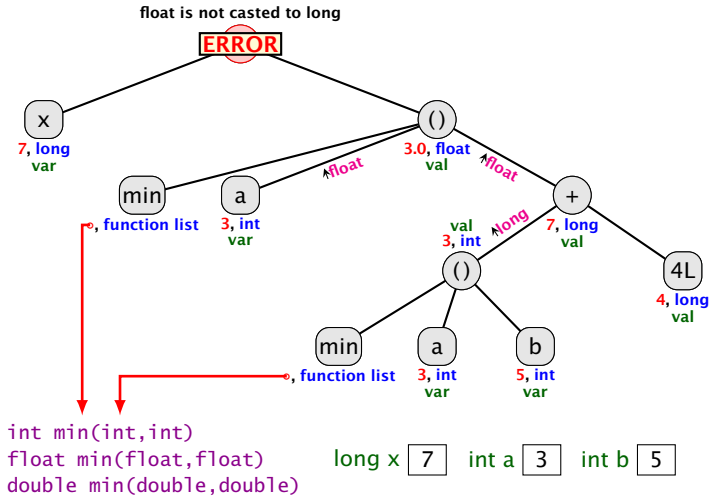


5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

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



5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

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

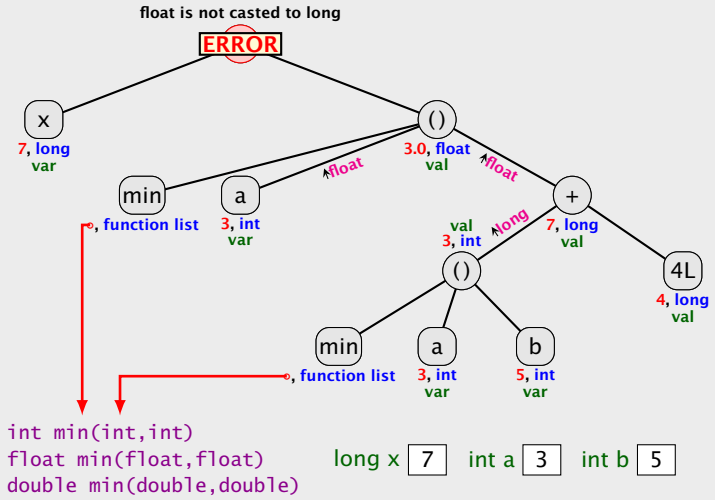
Impliziter Typecast – Strings

Spezialfall

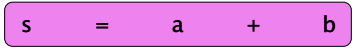
- Falls beim Operator + ein Typ vom Typ String ist, wird der andere auch in einen String umgewandelt.
  $\Longrightarrow$ Stringkonkatenation.
- Jeder Typ in Java besitzt eine Stringrepresentation.

Funktioniert nicht bei selbstgeschriebenen Funktionen.

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



Beispiel: s = a + b



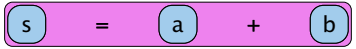
Impliziter Typecast – Strings

Spezialfall

- Falls beim Operator + ein Typ vom Typ String ist, wird der andere auch in einen String umgewandelt.
  $\Longrightarrow$ Stringkonkatenation.
- Jeder Typ in Java besitzt eine Stringrepresentation.

Funktioniert nicht bei selbstgeschriebenen Funktionen.

Beispiel: s = a + b



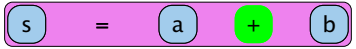
Impliziter Typecast – Strings

Spezialfall

- Falls beim Operator + ein Typ vom Typ String ist, wird der andere auch in einen String umgewandelt.
  $\Longrightarrow$ Stringkonkatenation.
- Jeder Typ in Java besitzt eine Stringrepresentation.

Funktioniert nicht bei selbstgeschriebenen Funktionen.

Beispiel: s = a + b



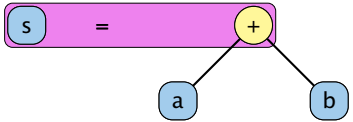
Impliziter Typecast – Strings

Spezialfall

- Falls beim Operator + ein Typ vom Typ String ist, wird der andere auch in einen String umgewandelt.
  $\Longrightarrow$ Stringkonkatenation.
- Jeder Typ in Java besitzt eine Stringrepresentation.

Funktioniert nicht bei selbstgeschriebenen Funktionen.

Beispiel: s = a + b



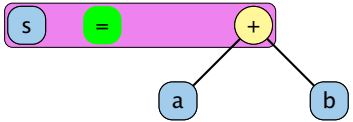
Impliziter Typecast – Strings

Spezialfall

- Falls beim Operator + ein Typ vom Typ String ist, wird der andere auch in einen String umgewandelt.
  $\Longrightarrow$ Stringkonkatenation.
- Jeder Typ in Java besitzt eine Stringrepresentation.

Funktioniert nicht bei selbstgeschriebenen Funktionen.

Beispiel: s = a + b



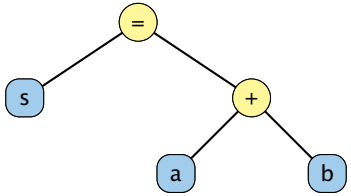
Impliziter Typecast – Strings

Spezialfall

- Falls beim Operator + ein Typ vom Typ String ist, wird der andere auch in einen String umgewandelt.
  $\Longrightarrow$ Stringkonkatenation.
- Jeder Typ in Java besitzt eine Stringrepresentation.

Funktioniert nicht bei selbstgeschriebenen Funktionen.

Beispiel: s = a + b



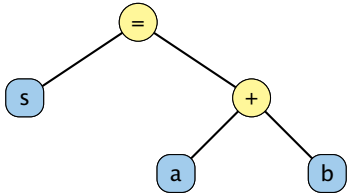
Impliziter Typecast – Strings

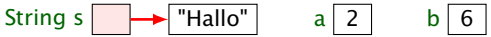
Spezialfall

- Falls beim Operator + ein Typ vom Typ String ist, wird der andere auch in einen String umgewandelt.
  $\Longrightarrow$ Stringkonkatenation.
- Jeder Typ in Java besitzt eine Stringrepresentation.

Funktioniert nicht bei selbstgeschriebenen Funktionen.

Beispiel: s = a + b





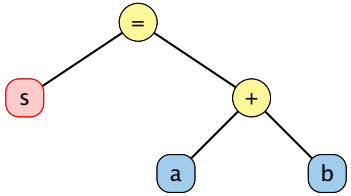
Impliziter Typecast – Strings

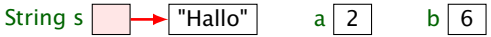
Spezialfall

- Falls beim Operator + ein Typ vom Typ String ist, wird der andere auch in einen String umgewandelt.
  $\Longrightarrow$ Stringkonkatenation.
- Jeder Typ in Java besitzt eine Stringrepresentation.

Funktioniert nicht bei selbstgeschriebenen Funktionen.

Beispiel: s = a + b





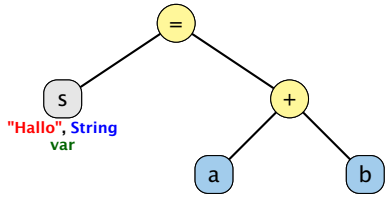
Impliziter Typecast – Strings

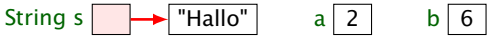
Spezialfall

- Falls beim Operator + ein Typ vom Typ String ist, wird der andere auch in einen String umgewandelt.
  $\Longrightarrow$ Stringkonkatenation.
- Jeder Typ in Java besitzt eine Stringrepresentation.

Funktioniert nicht bei selbstgeschriebenen Funktionen.

Beispiel: s = a + b





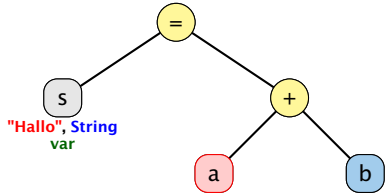
Impliziter Typecast – Strings

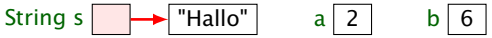
Spezialfall

- Falls beim Operator + ein Typ vom Typ String ist, wird der andere auch in einen String umgewandelt.
  $\Longrightarrow$ Stringkonkatenation.
- Jeder Typ in Java besitzt eine Stringrepresentation.

Funktioniert nicht bei selbstgeschriebenen Funktionen.

Beispiel: s = a + b





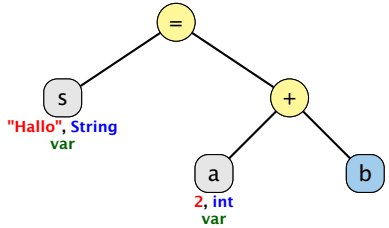
Impliziter Typecast – Strings

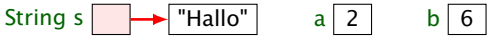
Spezialfall

- Falls beim Operator + ein Typ vom Typ String ist, wird der andere auch in einen String umgewandelt.
  $\Longrightarrow$ Stringkonkatenation.
- Jeder Typ in Java besitzt eine Stringrepresentation.

Funktioniert nicht bei selbstgeschriebenen Funktionen.

Beispiel: s = a + b





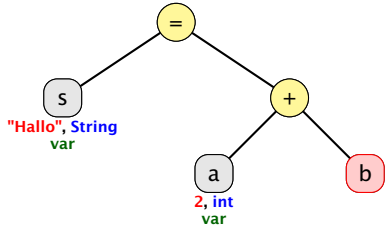
Impliziter Typecast – Strings

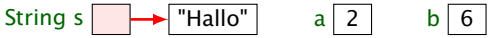
Spezialfall

- Falls beim Operator + ein Typ vom Typ String ist, wird der andere auch in einen String umgewandelt.
  $\Longrightarrow$ Stringkonkatenation.
- Jeder Typ in Java besitzt eine Stringrepresentation.

Funktioniert nicht bei selbstgeschriebenen Funktionen.

Beispiel: s = a + b





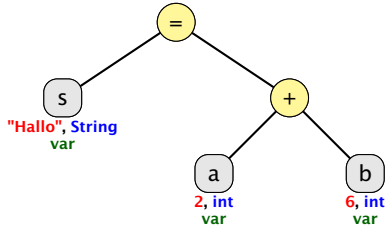
Impliziter Typecast – Strings

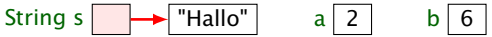
Spezialfall

- Falls beim Operator + ein Typ vom Typ String ist, wird der andere auch in einen String umgewandelt.
  $\Longrightarrow$ Stringkonkatenation.
- Jeder Typ in Java besitzt eine Stringrepresentation.

Funktioniert nicht bei selbstgeschriebenen Funktionen.

Beispiel: s = a + b





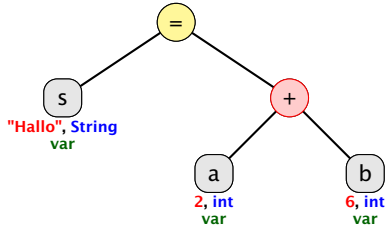
Impliziter Typecast – Strings

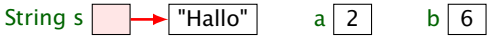
Spezialfall

- Falls beim Operator + ein Typ vom Typ String ist, wird der andere auch in einen String umgewandelt.
  $\Longrightarrow$ Stringkonkatenation.
- Jeder Typ in Java besitzt eine Stringrepresentation.

Funktioniert nicht bei selbstgeschriebenen Funktionen.

Beispiel: s = a + b





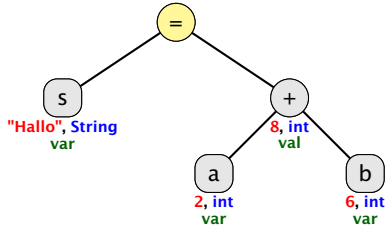
Impliziter Typecast – Strings

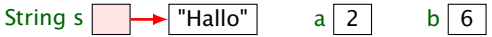
Spezialfall

- Falls beim Operator + ein Typ vom Typ String ist, wird der andere auch in einen String umgewandelt.
  $\Longrightarrow$ Stringkonkatenation.
- Jeder Typ in Java besitzt eine Stringrepresentation.

Funktioniert nicht bei selbstgeschriebenen Funktionen.

Beispiel: s = a + b





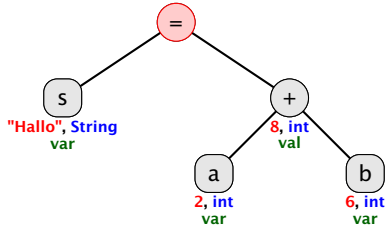
Impliziter Typecast – Strings

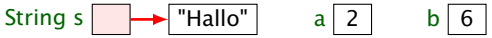
Spezialfall

- Falls beim Operator + ein Typ vom Typ String ist, wird der andere auch in einen String umgewandelt.
  $\Longrightarrow$ Stringkonkatenation.
- Jeder Typ in Java besitzt eine Stringrepresentation.

Funktioniert nicht bei selbstgeschriebenen Funktionen.

Beispiel: s = a + b





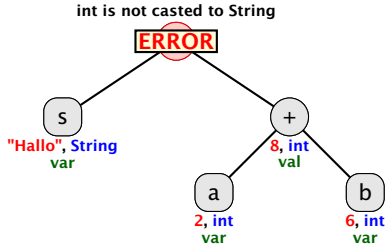
Impliziter Typecast – Strings

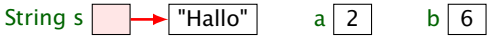
Spezialfall

- Falls beim Operator + ein Typ vom Typ String ist, wird der andere auch in einen String umgewandelt.
  $\Longrightarrow$ Stringkonkatenation.
- Jeder Typ in Java besitzt eine Stringrepresentation.

Funktioniert nicht bei selbstgeschriebenen Funktionen.

Beispiel: s = a + b





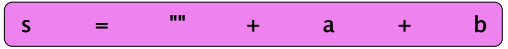
Impliziter Typecast – Strings

Spezialfall

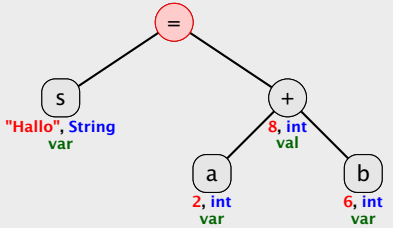
- Falls beim Operator + ein Typ vom Typ String ist, wird der andere auch in einen String umgewandelt.
  $\Longrightarrow$ Stringkonkatenation.
- Jeder Typ in Java besitzt eine Stringrepresentation.

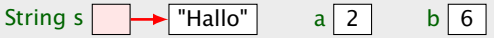
Funktioniert nicht bei selbstgeschriebenen Funktionen.

Beispiel: s = "" + a + b

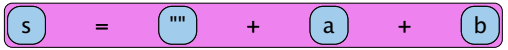


Beispiel: s = a + b

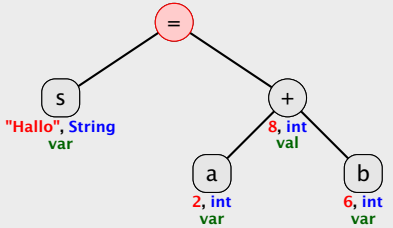


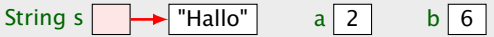


Beispiel: s = "" + a + b

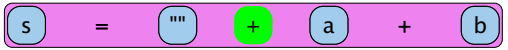


Beispiel: s = a + b

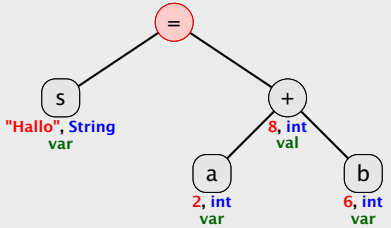




Beispiel: s = "" + a + b

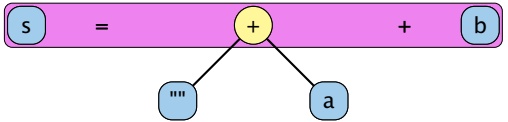


Beispiel: s = a + b

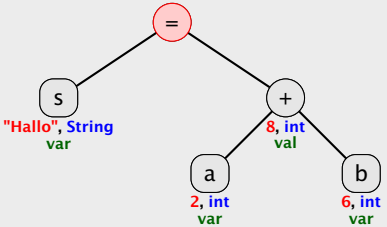


String s "Hallo" a 2 b 6

Beispiel: s = "" + a + b

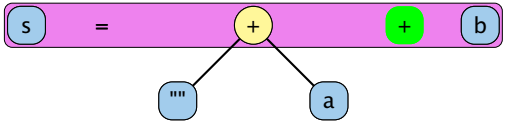


Beispiel: s = a + b

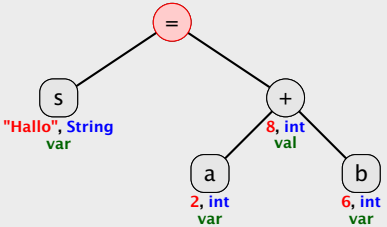


String s → "Hallo" a 2 b 6

Beispiel: s = "" + a + b

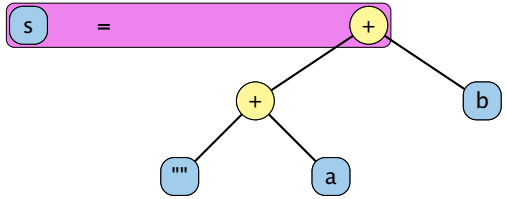


Beispiel: s = a + b

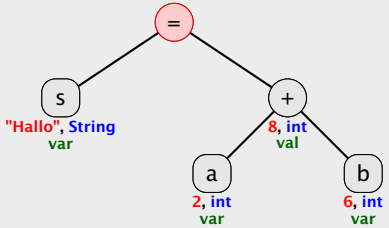


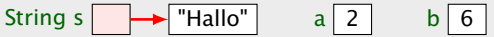
String s → "Hallo" a 2 b 6

Beispiel: s = "" + a + b

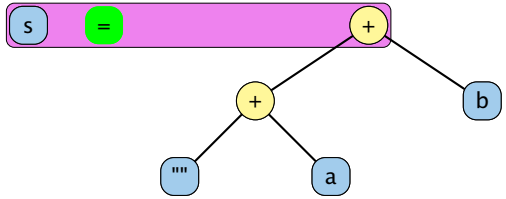


Beispiel: s = a + b

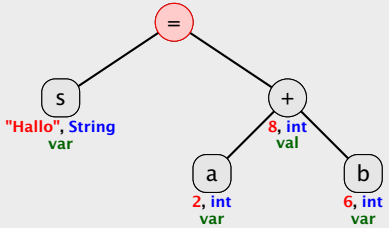




Beispiel: s = "" + a + b

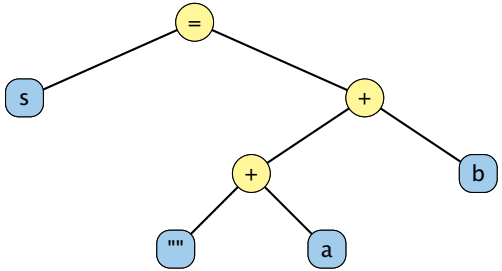


Beispiel: s = a + b



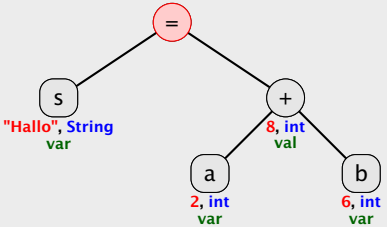
String s → "Hallo" a 2 b 6

Beispiel: s = "" + a + b



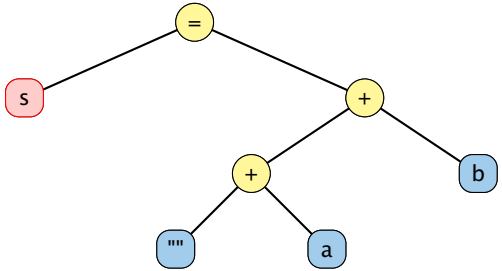
String s → "Hallo" a 2 b 6

Beispiel: s = a + b



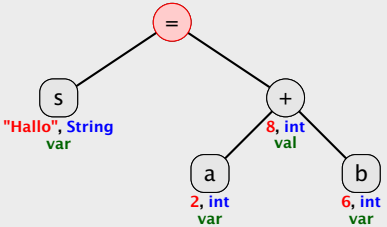
String s → "Hallo" a 2 b 6

Beispiel: s = "" + a + b



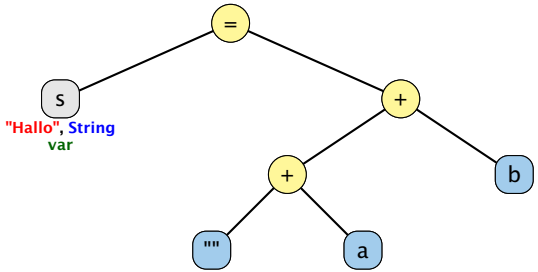
String s → "Hallo" a 2 b 6

Beispiel: s = a + b



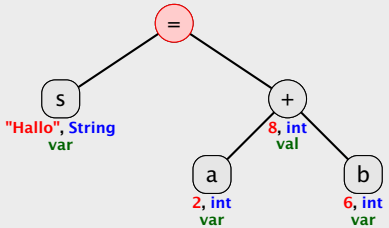
String s → "Hallo" a 2 b 6

Beispiel: s = "" + a + b



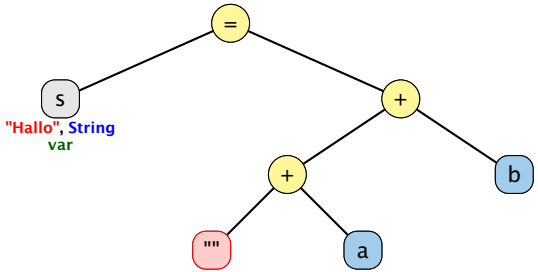
String s → "Hallo" a 2 b 6

Beispiel: s = a + b



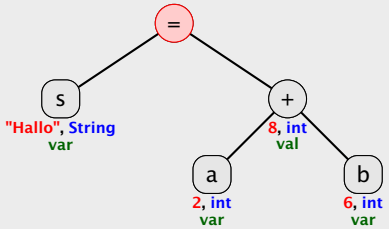
String s → "Hallo" a 2 b 6

Beispiel: s = "" + a + b



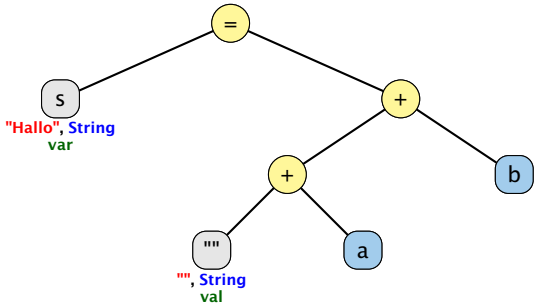
String s → "Hallo" a 2 b 6

Beispiel: s = a + b



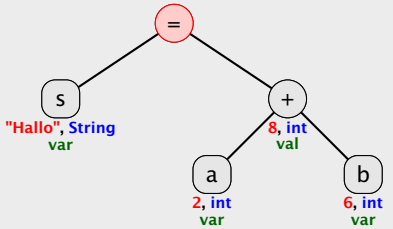
String s → "Hallo" a 2 b 6

Beispiel: s = "" + a + b



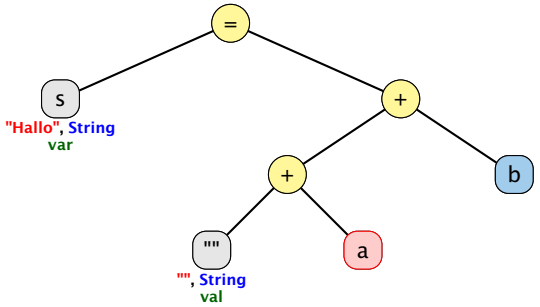
String s → "Hallo" a 2 b 6

Beispiel: s = a + b



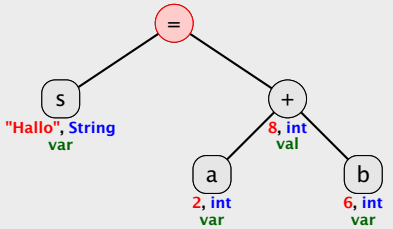
String s → "Hallo" a 2 b 6

Beispiel: s = "" + a + b



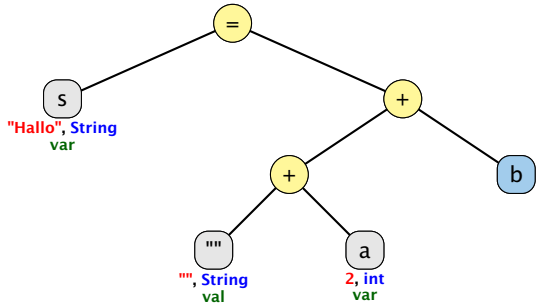
String s → "Hallo" a 2 b 6

Beispiel: s = a + b



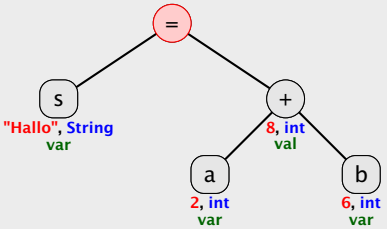
String s → "Hallo" a 2 b 6

Beispiel: s = "" + a + b



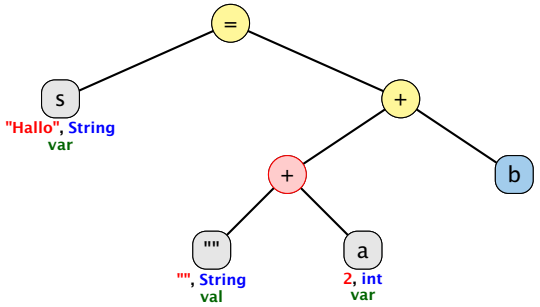
String s → "Hallo" a 2 b 6

Beispiel: s = a + b



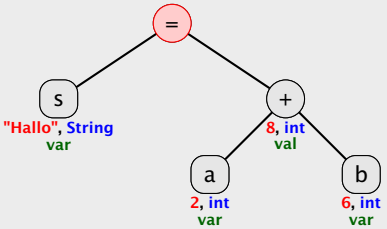
String s → "Hallo" a 2 b 6

Beispiel: s = "" + a + b



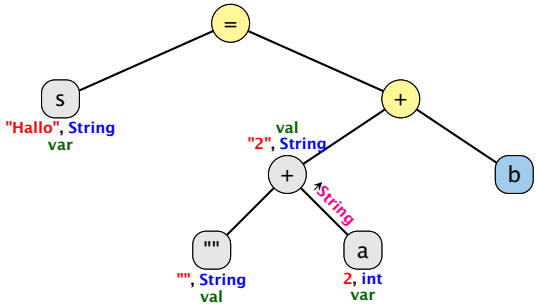
String s → "Hallo" a 2 b 6

Beispiel: s = a + b



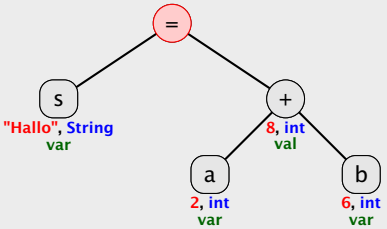
String s → "Hallo" a 2 b 6

Beispiel: s = "" + a + b



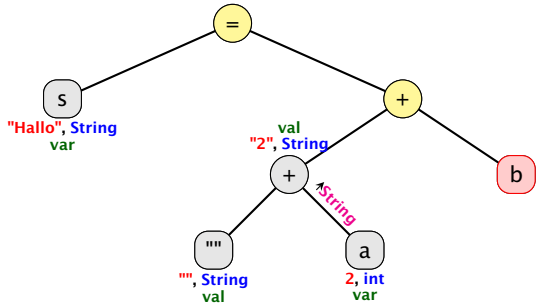
String s → "Hallo" a 2 b 6

Beispiel: s = a + b



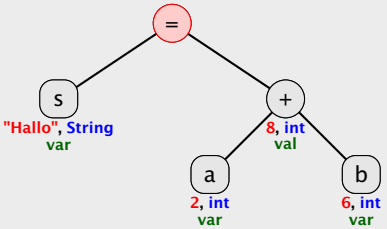
String s → "Hallo" a 2 b 6

Beispiel: s = "" + a + b



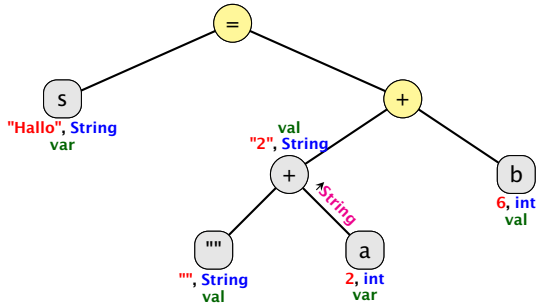
String s → "Hallo" a 2 b 6

Beispiel: s = a + b



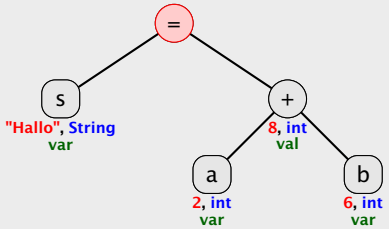
String s → "Hallo" a 2 b 6

Beispiel: s = "" + a + b



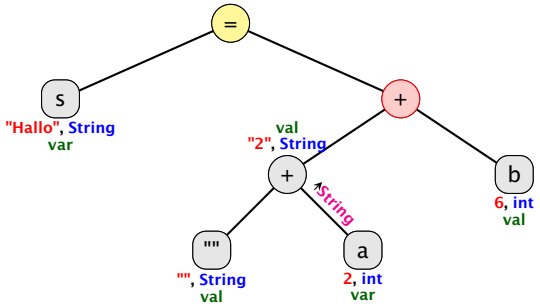
String s → "Hallo" a 2 b 6

Beispiel: s = a + b



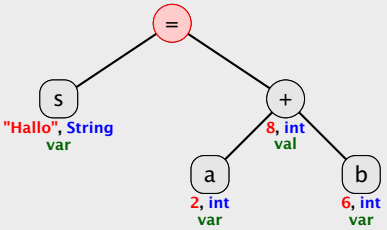
String s → "Hallo" a 2 b 6

Beispiel: s = "" + a + b



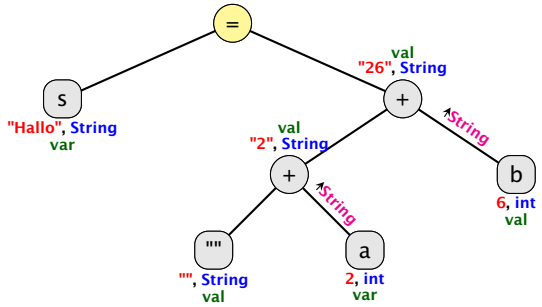
String s → "Hallo" a 2 b 6

Beispiel: s = a + b



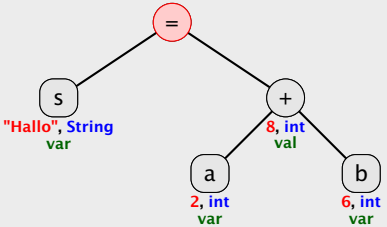
String s → "Hallo" a 2 b 6

Beispiel: s = "" + a + b



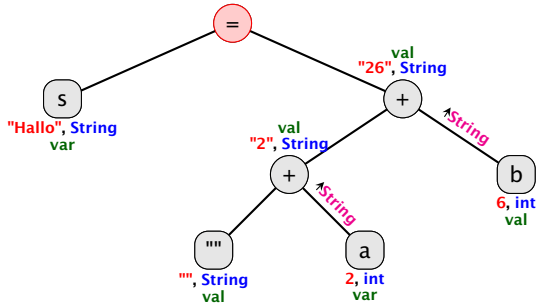
String s → "Hallo" a 2 b 6

Beispiel: s = a + b



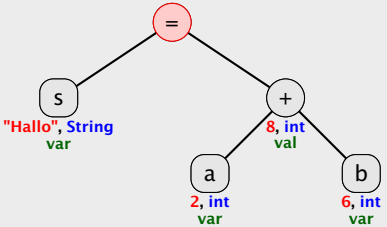
String s → "Hallo" a 2 b 6

Beispiel: s = "" + a + b



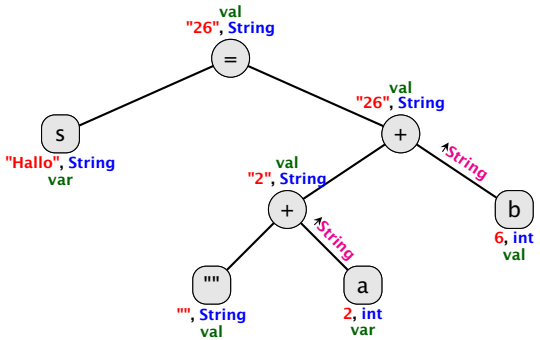
String s → "Hallo" a 2 b 6

Beispiel: s = a + b

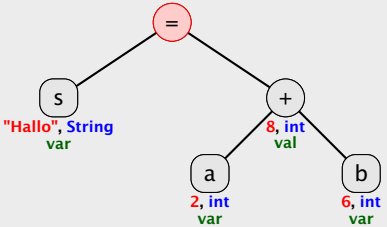


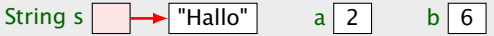
String s → "Hallo" a 2 b 6

Beispiel: s = "" + a + b

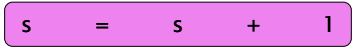


Beispiel: s = a + b

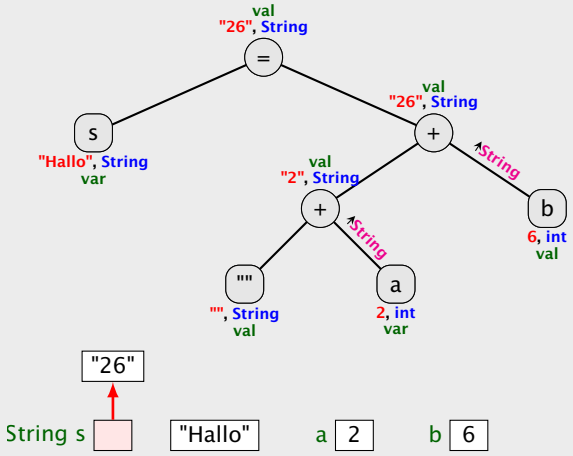




Beispiel: s = s + 1

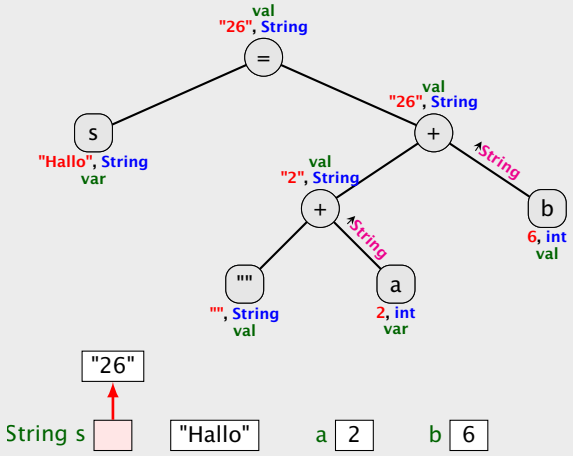


Beispiel: s = "" + a + b



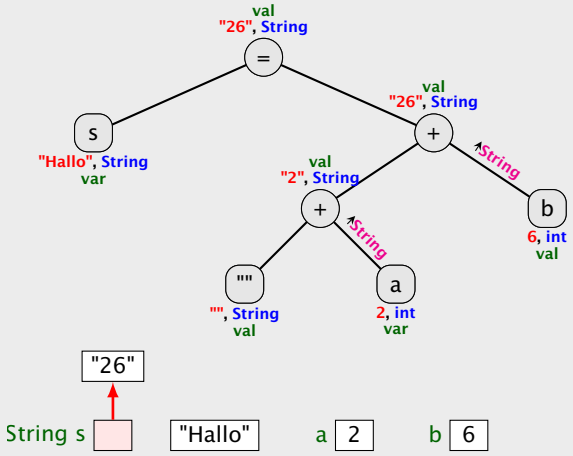
Beispiel: s = s + 1

Beispiel: s = "" + a + b

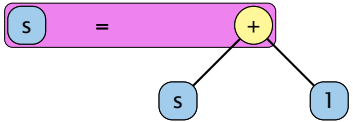


Beispiel: s = s + 1

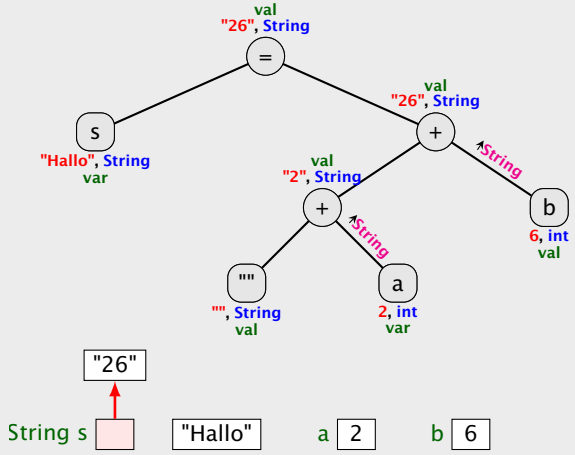
Beispiel: s = "" + a + b



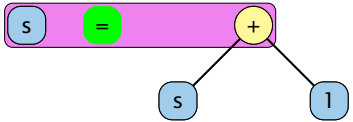
Beispiel: s = s + 1



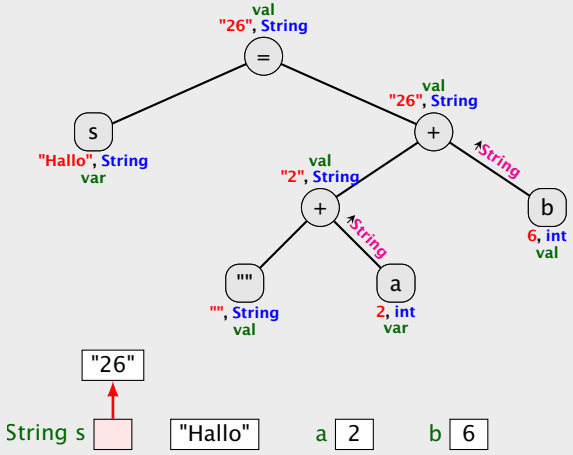
Beispiel: s = "" + a + b



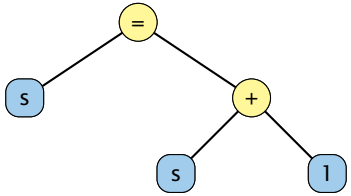
Beispiel: s = s + 1



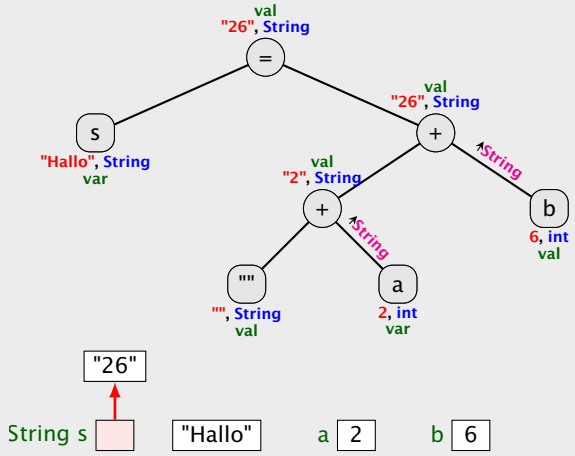
Beispiel: s = "" + a + b



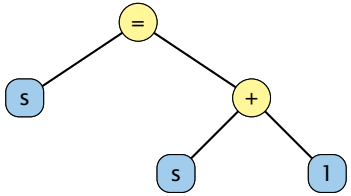
Beispiel: s = s + 1



Beispiel: s = "" + a + b

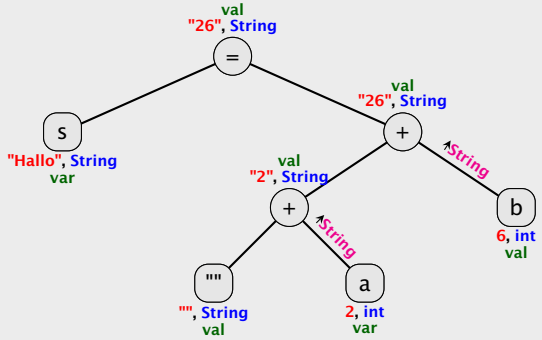


Beispiel: s = s + 1



short s 4

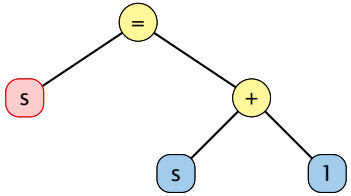
Beispiel: s = "" + a + b



"26"

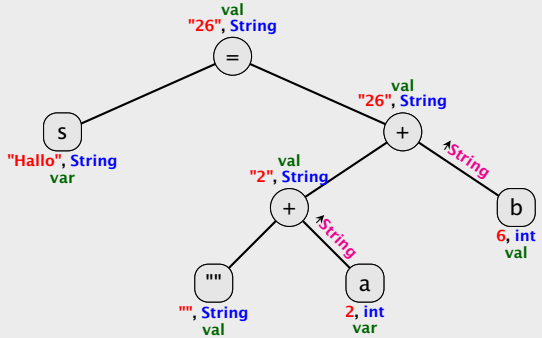
String s "Hallo" a 2 b 6

Beispiel: s = s + 1



short s 4

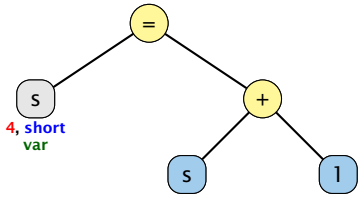
Beispiel: s = "" + a + b



"26"

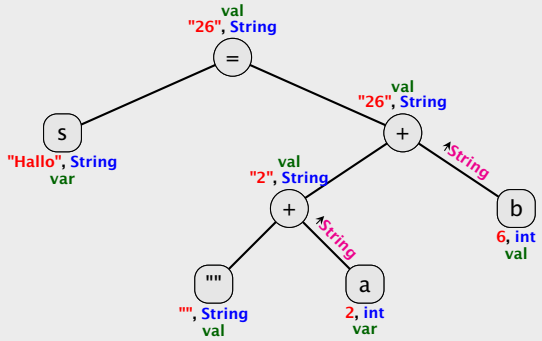
String s "Hallo" a 2 b 6

Beispiel: s = s + 1



short s 4

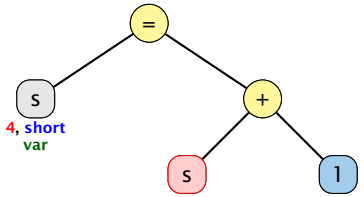
Beispiel: s = "" + a + b



"26"

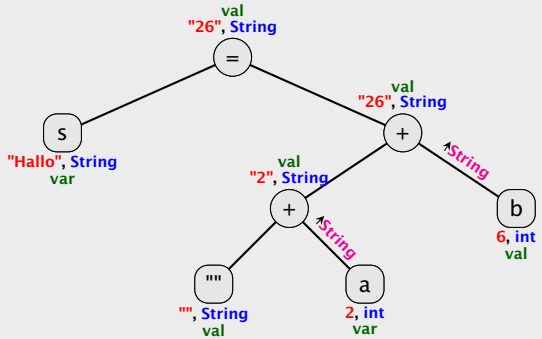
String s "Hallo" a 2 b 6

Beispiel: s = s + 1



short s 4

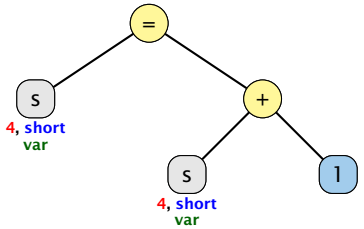
Beispiel: s = "" + a + b



"26"

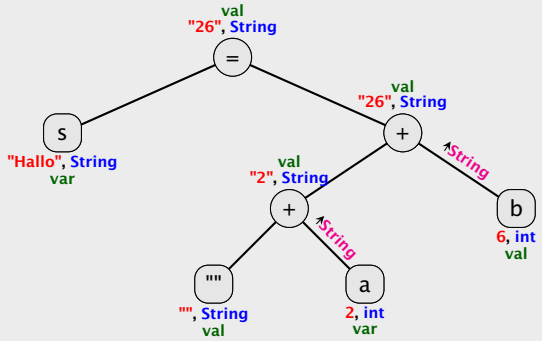
String s "Hallo" a 2 b 6

Beispiel: s = s + 1



short s | 4 |

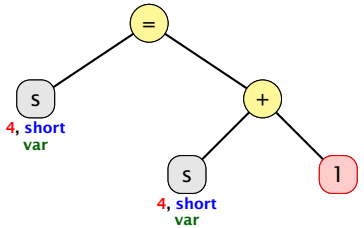
Beispiel: s = "" + a + b



"26"

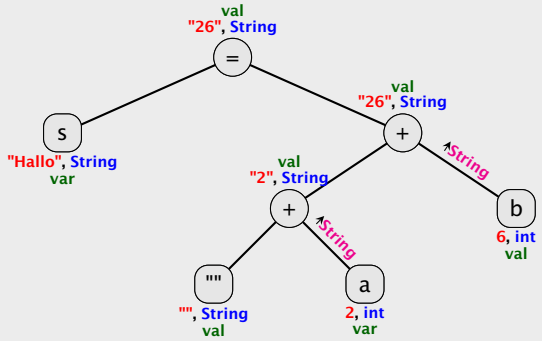
String s | "Hallo" | a | 2 | b | 6 |

Beispiel: s = s + 1



short s | 4 |

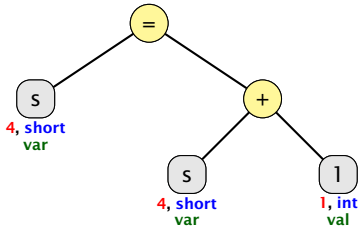
Beispiel: s = "" + a + b



"26"

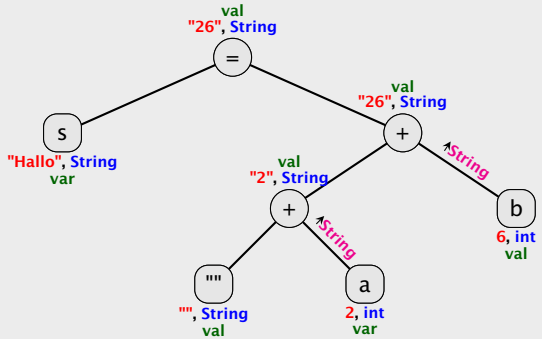
String s | "Hallo" | a | 2 | b | 6 |

Beispiel: s = s + 1



short s 4

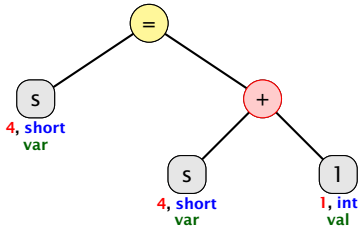
Beispiel: s = "" + a + b



"26"

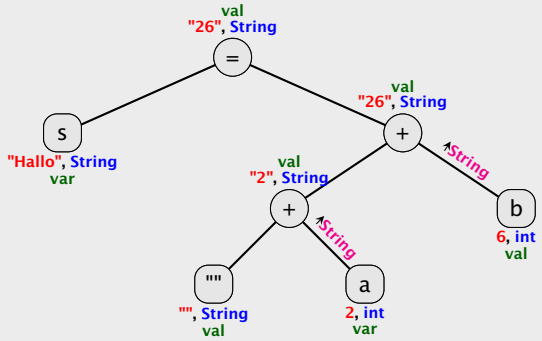
String s "Hallo" a 2 b 6

Beispiel: s = s + 1



short s | 4 |

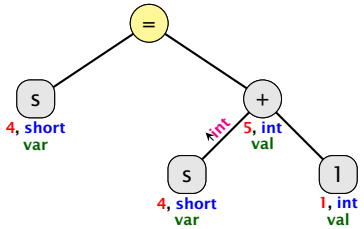
Beispiel: s = "" + a + b



"26"

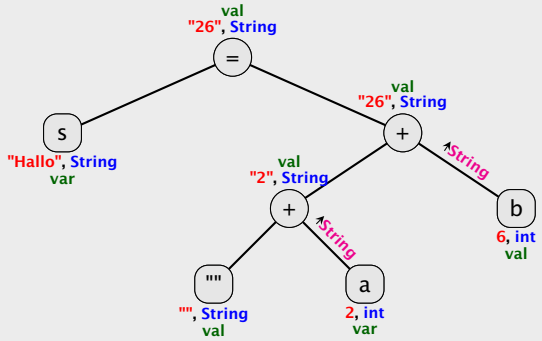
String s | "Hallo" | a | 2 | b | 6 |

Beispiel: s = s + 1



short s 4

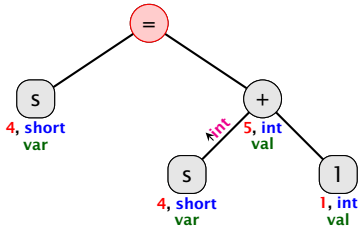
Beispiel: s = "" + a + b



"26"

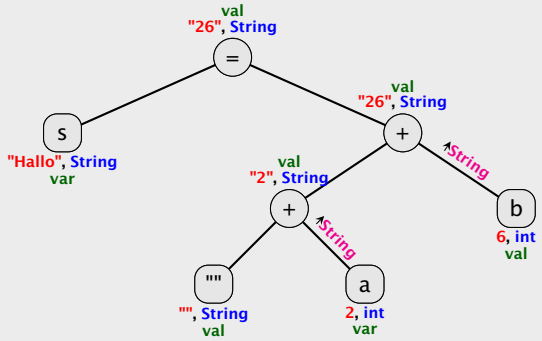
String s "Hallo" a 2 b 6

Beispiel: s = s + 1



short s | 4 |

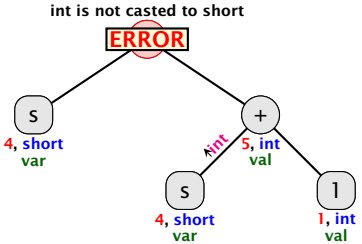
Beispiel: s = "" + a + b



"26"

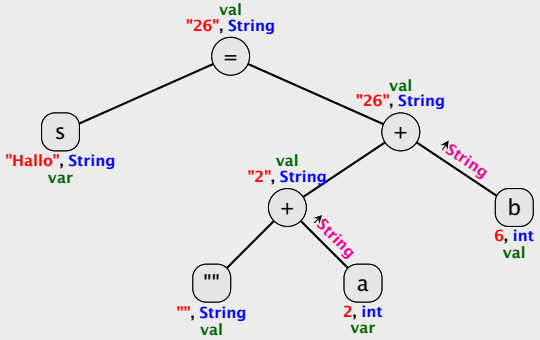
String s | "Hallo" | a | 2 | b | 6 |

Beispiel: s = s + 1



short s | 4 |

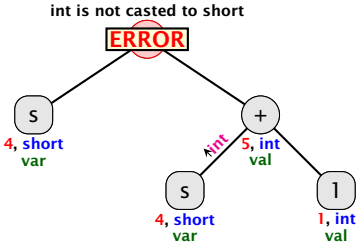
Beispiel: s = "" + a + b



"26"

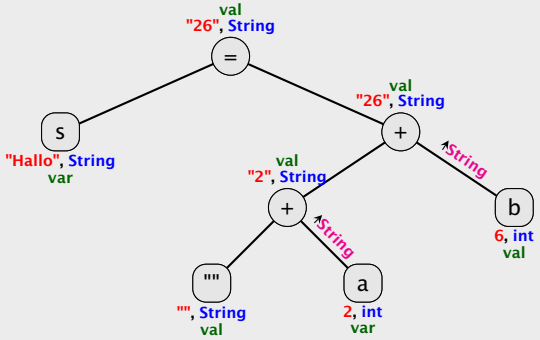
String s | "Hallo" | a | 2 | b | 6 |

Beispiel: s = s + 1



short s | 4 |

Beispiel: s = "" + a + b

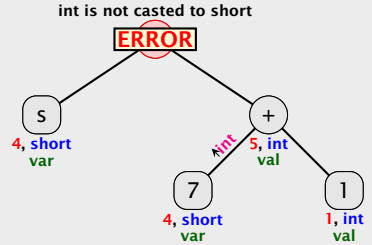


"26"

String s | "Hallo" | a | 2 | b | 6 |

Beispiel: s = 7 + 1

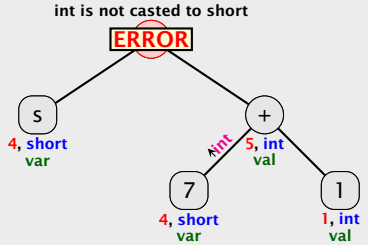
Beispiel: s = s + 1



short s 4

Beispiel: s = 7 + 1

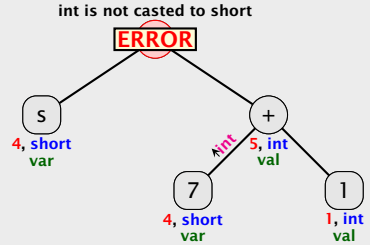
Beispiel: s = s + 1



short s 4

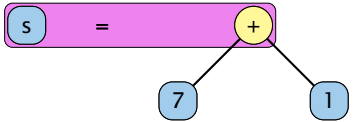
Beispiel: s = 7 + 1

Beispiel: s = s + 1

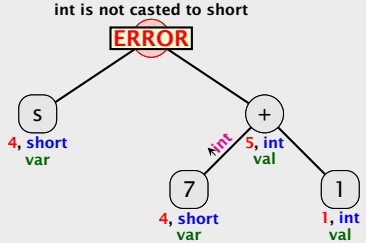


short s 4

Beispiel: s = 7 + 1

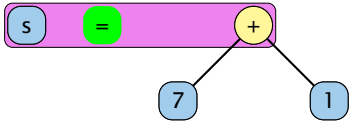


Beispiel: s = s + 1

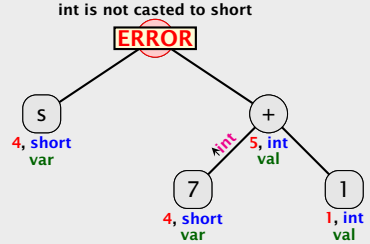


short s 4

Beispiel: s = 7 + 1

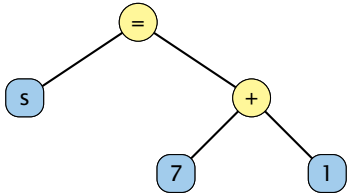


Beispiel: s = s + 1

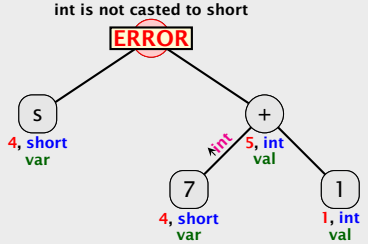


short s 4

Beispiel: s = 7 + 1

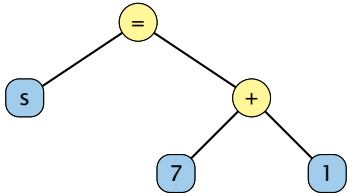


Beispiel: s = s + 1



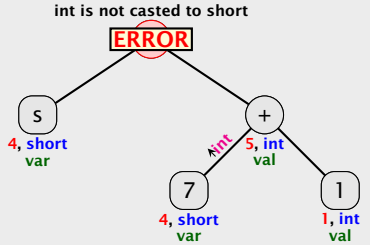
short s 4

Beispiel: s = 7 + 1



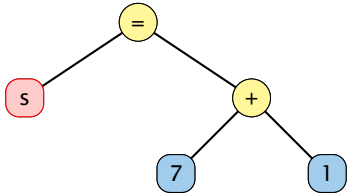
short s 4

Beispiel: s = s + 1



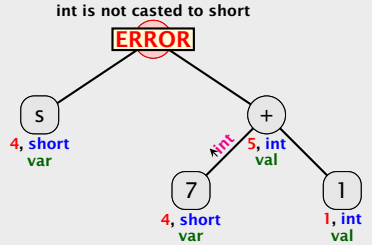
short s 4

Beispiel: s = 7 + 1



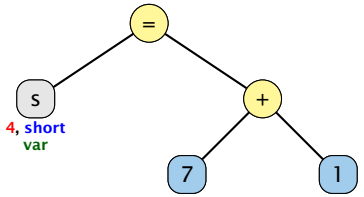
short s 4

Beispiel: s = s + 1



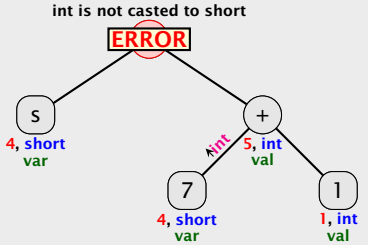
short s 4

Beispiel: s = 7 + 1



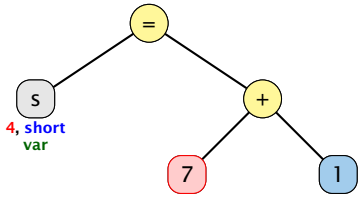
short s 4

Beispiel: s = s + 1



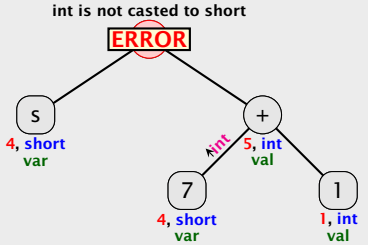
short s 4

Beispiel: s = 7 + 1



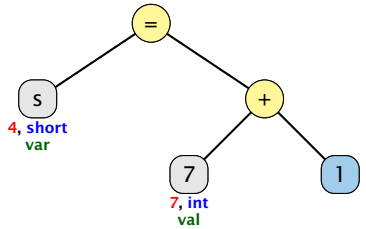
short s 4

Beispiel: s = s + 1



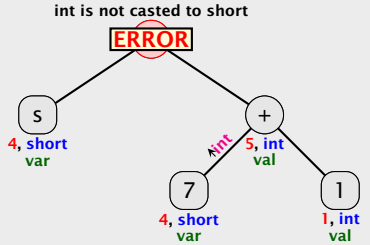
short s 4

Beispiel: s = 7 + 1



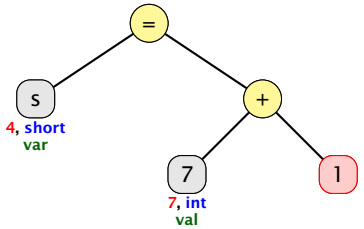
short s 4

Beispiel: s = s + 1



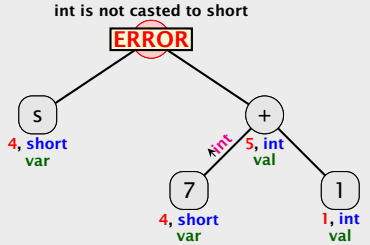
short s 4

Beispiel: s = 7 + 1



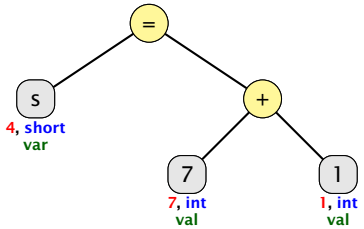
short s 4

Beispiel: s = s + 1



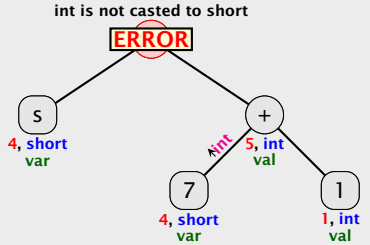
short s 4

Beispiel: s = 7 + 1



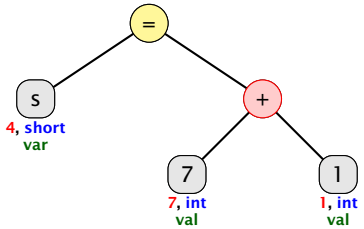
short s 4

Beispiel: s = s + 1



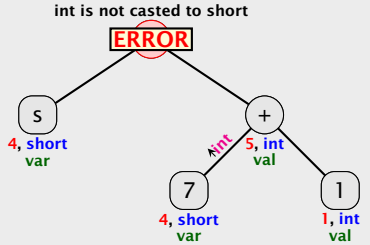
short s 4

Beispiel: s = 7 + 1



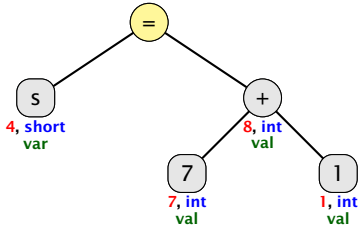
short s 4

Beispiel: s = s + 1



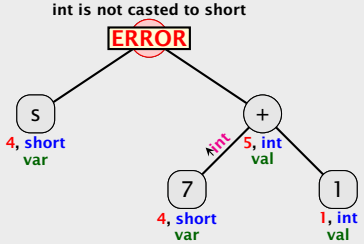
short s 4

Beispiel: s = 7 + 1



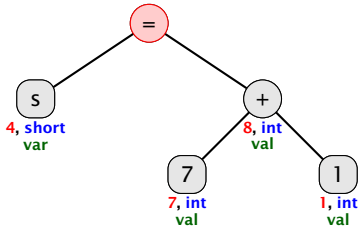
short s 4

Beispiel: s = s + 1



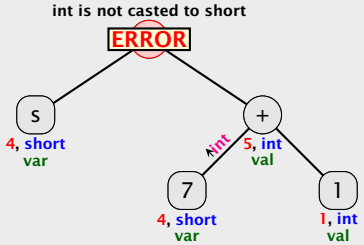
short s 4

Beispiel: s = 7 + 1



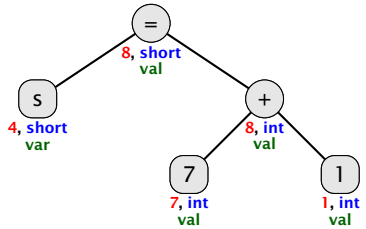
short s 4

Beispiel: s = s + 1



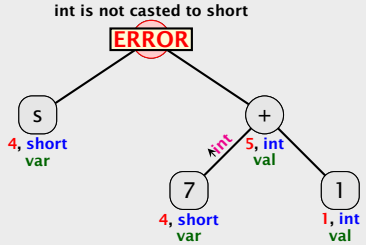
short s 4

Beispiel: s = 7 + 1



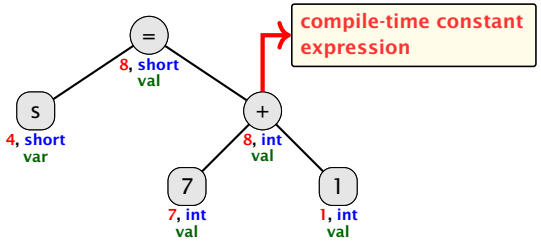
short s 8

Beispiel: s = s + 1



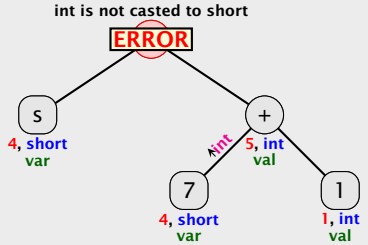
short s 4

Beispiel: s = 7 + 1



short s 8

Beispiel: s = s + 1



short s 4

Expliziter Typecast

symbol	*name*	*type*	*L/R*	*level*
(type)	typecast	zahl, char	rechts	3

Beispiele mit Datenverlust

- `short s = (short) 23343445;`

  Die obersten bits werden einfach weggeworfen...

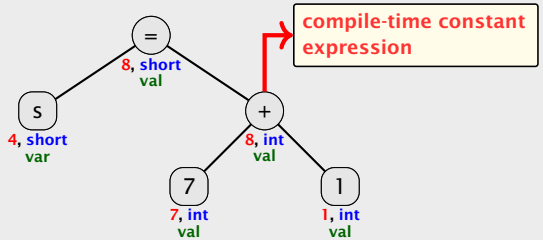
- ```
  double d = 1.5;
  short  s = (short) d;
  ```

  d hat danach den Wert 1.

...ohne Datenverlust:

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

Beispiel: s = 7 + 1



short s 8