Fachrichtung 6.2 — Informatik
Universität des Saarlandes
Tutorenteam der Vorlesung Programmierung 1

Programmierung 1 (Wintersemester 2012/13)

Erklärung 2

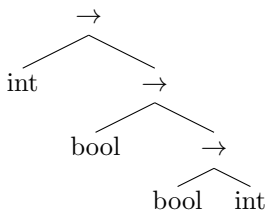
(Baumdarstellung von Phrasen)

Hinweis: Dieses Blatt enthält eine zusätzliche Erklärung erstellt von den Tutoren. Für die Richtigkeit besteht daher keine Gewähr.

Die Erklärung sowie ihr Thema sind weder für die Klausur relevant noch irrelevant.
Die ursprüngliche Erklärung stammt aus dem Wintersemester 11/12

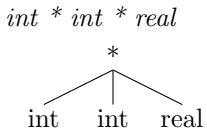
1 Baumdarstellung von Typen

a) Der Pfeil klammert rechts, deshalb wird der Typ $int \to bool \to bool \to int$ wie folgt dargestellt:

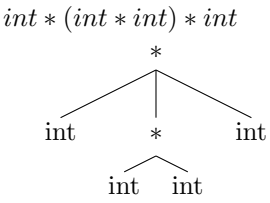


Der Baum entspricht auch der Darstellung des Typs *int -¿ (bool -¿ (bool -¿ int)))*. Man kann zu einer Baumdarstellung also immer mehrere Zeichendarstellungen finden, zu einer Zeichendarstellung gibt es allerdings immer nur genau eine Baumdarstellung.

b) Der Stern klammert bei Typen stärker als der Pfeil. Außerdem klammert Stern der bei Typen weder links noch rechts (im Gegensatz zu Mal). Ansonsten könnte man keine Tupel beliebiger Länge darstellen. Setzen wir in Tupeltypen klammern, so ändern sich die Typen. Ein Beispiel: Für den Typen $int * int * int$ ist z.B. $(2, 3, 4)$ ein möglicher Wert, wohingegen für $int * (int * int)$ der Tupel nicht in Frage kommt, sondern nur Tupel wie z.B. $(2, (3, 4))$ Während der erste Tupel ein Tripel ist, ist der zweite ein Paar, mit einem Paar in zweiter Komponente.



c) In der Baumdarstellung eines Typs können niemals Klammern vorkommen. Die Klammern werden durch den Baum dargestellt. Hat man z.B. ein Tripel, der ein Paar enthält, so sieht der Baum wie folgt aus:



2 Baumdarstellung zu Ausdrcken und Deklarationen

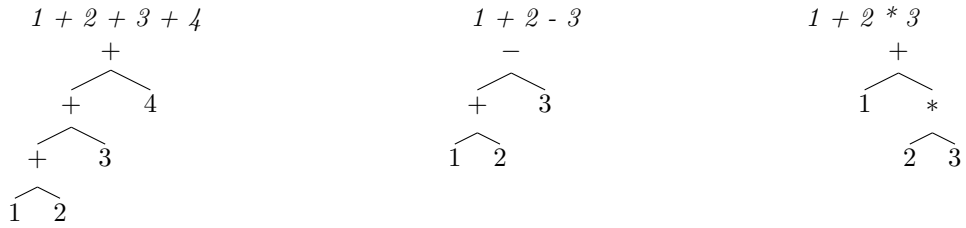
2.1 Operatoren

a) Zur Wiederholung: Einteilung der Operatoren nach der Stärke der Klammerung, von schwach(oben) nach stark(unten):

- $=, <>, <, <=, >, >=$
- $+, -$
- $*, /, div, mod$

b) Wenn zwei Operatoren der selben Stärke vorkommen, entscheidet die "Richtung" der Klammerung über die Struktur des Baumes.

- +, -, *, /, div und mod klammern links (*Zur Erinnerung: links-klammernd heißt, dass man die Klammern links weglassen darf ohne die Baumstruktur zu verändern*)
- Später lernen wir noch zwei Operatoren kennen, die rechts klammern.



2.2 Projektion

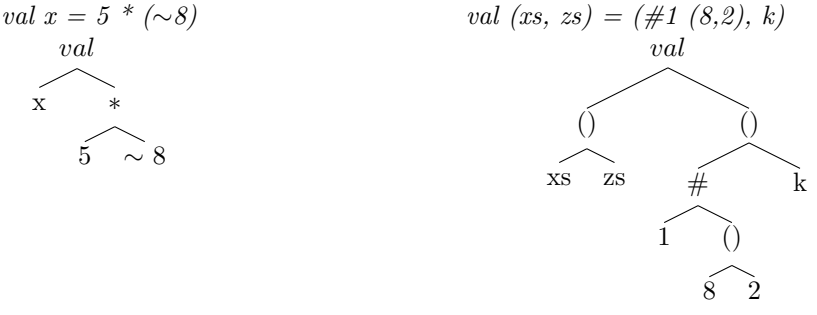
Hat man eine Projektion im Ausdruck, so hat der zugehörige Knoten im Baum immer genau zwei Zweige nach unten. Links steht die Position auf die projiziert werden soll, rechts das Tupel oder der Bezeichner auf den die Projektion angewendet wird. Um anzudeuten, dass es sich um einen Tupel handelt, stehen im Baum die runden Klammern und darunter alle Komponenten!

2.3 Prozeduranwendung

Die Prozeduranwendung wird im Baum mithilfe eines Punktes verdeutlicht. Links unter dem Punkt steht der Name der Prozedur, rechts davon worauf diese angewendet wird. Wichtig ist: Die Prozeduranwendung klammert links!

2.4 Val-Deklaration

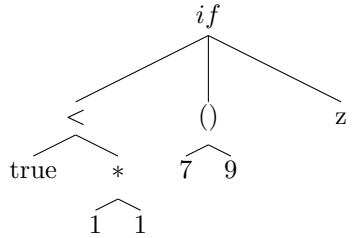
Haben wir eine Deklaration mit *val*, so gehen vom entsprechenden Knoten im Baum immer genau zwei Zweige ab: links steht der Bezeichner, der gebunden wird, und rechts steht der Ausdruck, an den der Bezeichner gebunden werden soll. Das Schlüsselwort = kommt dabei in dem Baum nicht vor. Steht ein = im Baum, so handelt es sich um den Vergleichsoperator.



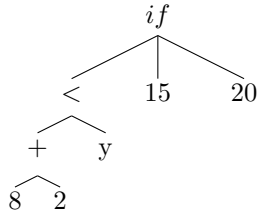
2.5 Konditional

Bei einem Konditional hat man als Wurzel das *if*, von dem genau drei Kanten abgehen. Zuerst kommt die Bedingung, dann die Konsequenz und danach die Alternative.

if true $< 1 * 1$ *then* $(7, 9)$ *else z*



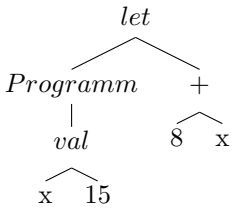
if $8 + 2 < y$ *then* 15 *else* 20



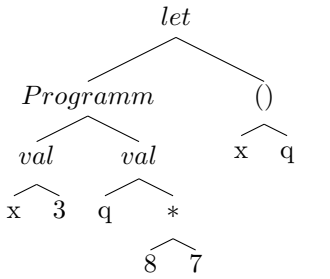
2.6 Let-Ausdruck

Bei einem Let-Ausdruck ist die Wurzel das *let*, links folgt immer *Programm* und rechts der Ausdruck, der normalerweise zwischen *in* und *end* steht. Das Programm ist hierbei ein Unterknoten, der selbst wieder beliebig viele Unterknoten haben kann, je nach dem wie viele Deklarationen zwischen *let* und *in* stehen.

let val $x = 15$ *in* $8 + x$ *end*



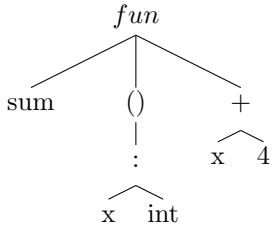
let val $x = 3$ *val* $q = 8 * 7$ *in (x,q) end*



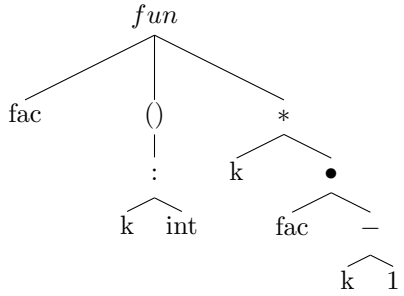
2.7 Prozedurdeklaration

Bei einer Prozedur hat man drei bis vier Zweige unterhalb des Knotens. Die Wurzel ist immer das Schlüsselwort *fun*. Der Zweig ganz links beinhaltet immer den Namen der Prozedur, der zweite Zweig das Argumentmuster (*Achtung:* Hier stehen *immer* Klammern, auch wenn es nur ein Argument gibt!). Der Dritte ist optional, dort kann der Ergebnistyp stehen, falls dieser angeben ist. Am letzten Zweig befindet sich dann der Rumpf der Prozedur.

fun sum (x:int) = $x + 4$



*fun fac (k:int) = k * fac(k-1)*



2.8 Abstraktionen

Wenn wir uns Abstraktionen als Prozeduren ohne Namen darstellen, so ist die Baumdarstellung direkt aus der für Prozeduren ableitbar. Man muss lediglich den Namen weglassen, den gibt es bei einer Abstraktion schlielich nicht. Außerdem ist der oberste Knoten nicht mehr *fun*, sondern entsprechend *fn*

fn $x \Rightarrow x * x$