Zwischendarstellung des *TassKaf*-Frontends

Marc Langenbach

12. April 2001

Inhaltsverzeichnis

1 Klassen **2**
- 1.1 Klasse `Node` 2
- 1.2 Klasse `Types` 3
- 1.3 Klasse `Util` 3
- 1.4 Klasse `GDLVisualizer` 4

2 Listen **4**

3 Typen **4**

4 Deklarationen **4**
- 4.1 Klassendeklarationen 4
- 4.2 Variablendeklarationen 5
- 4.3 Methodendeklarationen 6

5 Ausdrücke **7**
- 5.1 Konstanten 7
- 5.2 Variablen 7
- 5.3 Unäre Operatoren 8
- 5.4 Binäre Operatoren 8
- 5.5 New-Anweisung 8
- 5.6 Methodenaufrufe 9

6 Anweisungen **10**
- 6.1 Zuweisungen 10
- 6.2 Schleifen 10
- 6.3 Return-Anweisungen 11
- 6.4 If-Anweisungen 11
- 6.5 Switch-Anweisungen 11

1 Klassen

Bei der vom Frontend erzeugten Zwischendarstellung handelt es sich um einen dekorierten abstrakten Syntaxbaum. Die Knoten dieses Baums sind Objekte der Klasse `Node`. Die verschiedenen Knotentypen und Zugriffsmodi sind durch Konstanten in der Klasse `Types` definiert. The Klasse `Util` stellt einige Methoden zur Namenskonvertierung und weitere Hilfsmethoden zur Verfügung. Um GDL-Dateien zu erzeugen, kann die Klasse `GDLVisualizer` verwendet werden. Ein vollständige Erklärung dieser Klassen ist unter

`http://www.cs.uni-sb.de/~mlangen/TassKaf/packages.html`

zu finden.

1.1 Klasse **Node**

Die Klasse `Node` besteht aus einer Liste von Kindern, einer Referenz zum Elternknoten (falls dieser existiert) und zusätzlichen Feldern, die für die Codeerzeugung benötigt werden.

Relevante Felder sind:

access beschreibt die Art der Deklaration von Variablen oder Methoden. Falls es sich um einen `T_METHCALL`-Knoten handelt, bestimmt `access`, wie die Methode aufzurufen ist. Mögliche Werte sind `T_LOCAL`, `T_STATIC` und `T_DEFAULT`, wobei lediglich die beiden ersten von Bedeutung sind. Ist ein Methodenaufruf als `T_LOCAL` deklariert, wird eine Methode der eigenen Klasse aufgerufen und nicht einer Basisklasse. `T_STATIC` wird verwendet, um Klassenfelder und Klassenmethoden zu kennzeichnen. Ist die Zugriffsart einer Variable `T_LOCAL`, handelt es sich um eine lokale Variable, also eine, die in der Methode deklariert wurde.

data ist eine Referenz zu zusätzlichen Informationen. Es kann sich hierbei um ein `String`- oder `Integer`-Object handeln oder `null` sein.

decl ist eine Referenz zu der Deklaration einer Variable oder Methode. Im Falle einer Variable verweist `decl` auf einen `T_VARSPEC`-Knoten, handelt es sich um einen Methodenaufruf, wird die korrespondierende Methodendeklaration, also ein Knoten des Typs `T_METHDEC`, referenziert.

etype Falls der Knoten Teil eines Ausdrucks ist, ist der Typ des Ausdrucks als JVM-Typstring hierin enthalten.

offset ist der Offset einer lokalen Variable.

parent ist eine Referenz zum Elternknoten oder `null`.

type enthält den Typ des Knotens. Mögliche Werte sind in der Klasse `Types` definiert.

Methoden der Klasse `Node`, die während der Codeerzeugung benötigt werden, sind:

int getChildrenCount() liefert die Anzahl der Kinder, die der Knoten hat.

Node getChild(int idx) gibt das *idx*te Kind eines Knotens zurück. Zulässige Werte liegen zwischen 0 und `getChildrenCount()-1` (inklusive).

1.2 Klasse `Types`

Die Klasse `Types` stellt verschiedene Konstanten zur Verfügung, die verwendet werden, um den Typ eines Knotens festzulegen. Für eine vollständige Liste sei hier auf die Dokumentation der Klasse verwiesen. Zusätzlich enthält die Klasse eine Klassenmethode `String toString(int type)`, die eine Typkonstante in deren Stringrepräsentation konvertiert.

1.3 Klasse `Util`

Diese Klasse besteht aus verschiedenen statischen Methoden, die Java-Typen, wie sie in der Zwischendarstellung repräsentiert werden, in Zeichenketten konvertieren, die den JVM/Jasmin Namenskonventionen entsprechen. Zusätzlich wird eine Methode zum Auffinden einer Klasse oder eines Klassennamens bereitgestellt.

Methoden, die diese Klasse enthält, sind:

String toJVMName(String name) ersetzt lediglich alle Vorkommen von '.' durch '/'.

String toJVMTypeName(Node n, Node dim) erzeugt einen Typstring für den durch `n` und `dim` spezifizierten Typ, wobei es sich bei `dim` um `null` oder einen `T_DUMMY`-Knoten handeln kann, um skalare Typen darzustellen. Ist `dim` ein `T_LIST`- oder `T_DIMENSION`-Knoten, wird ein Typstring für ein Feld erzeugt.

Node getClass(Node n) liefert den `T_CLASS`-Knoten der Klasse, in der `n` spezifiziert wurde.

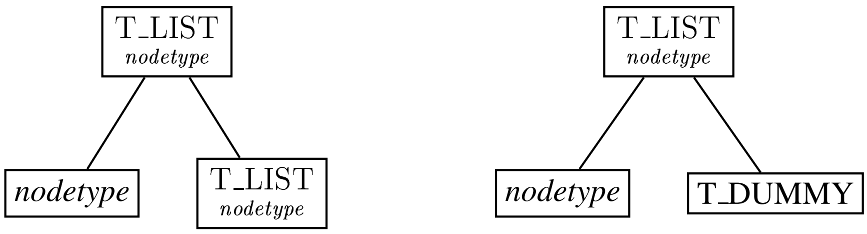
String getClassName(Node n) ähnlich wie `getClass()`, liefert aber den Namen der Klasse, in der `n` spezifiziert wurde. Dieser entspricht *nicht* der JVM-Namenskonvention.

1.4 Klasse `GDLVisualizer`

Die Klasse `GDLVisualizer` kann verwendet werden, um GDL-Dateien des Syntaxbaums zu erzeugen. Dazu ist die statische Methode `createFile()` aufzurufen, die den Dateinamen als String und die Wurzel des Baumes als Parameter übergeben bekommt. Die erzeugten Dateien können mittels `aisee` oder `xvcg` visualisiert werden.

2 Listen

Listen werden verwendet, um Sequenzen von Knoten gleichen Typs zu repräsentieren. Eine Liste mit Knoten vom Typ *nodetype* sieht wie folgt aus:



Die Listen sind rechtsrekursiv, d. h. das erste Element ist der am weitesten rechtsstehende Knoten. Mit anderen Worten, das erste Kind eines Knotens von Typ `T_LIST` ist ein Knoten vom Typ *nodetype*, das zweite Kind eine weitere Liste oder ein `T_DUMMY`-Knoten, um die Rekursion zu stoppen.

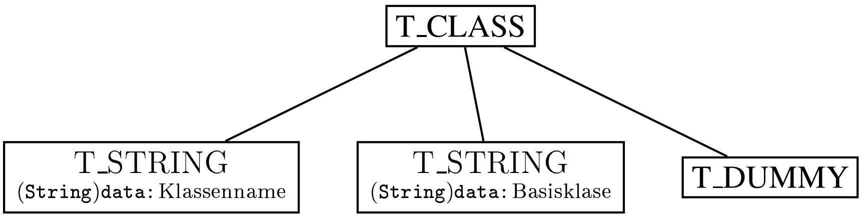
3 Typen

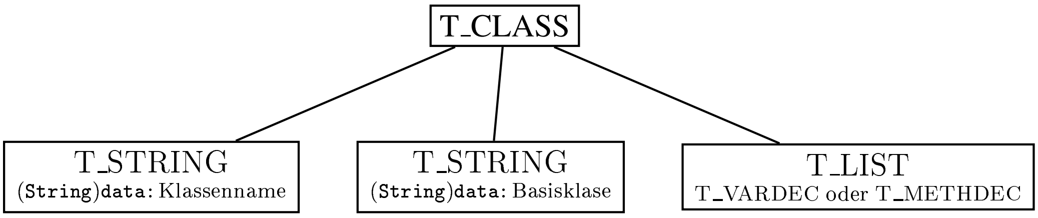
Es existieren drei verschiedene Typen von Knoten, um die Typen der Klasse *Tass-Kaf* zu repräsentieren – nicht zu verwechseln mit den Konstanten der Klasse `Types`. Diese sind:

4 Deklarationen

4.1 Klassendeklarationen

Eine Klassendeklarationen hat eines der folgenden Formate:

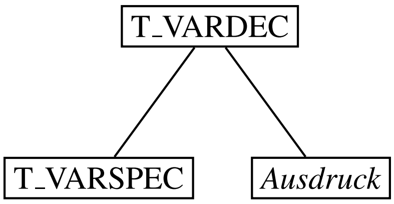




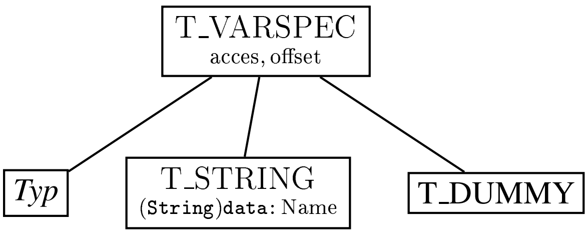
Das erste Kind enthält den Namen der Klasse, das zweite den Namen der Basisklasse. Ist der Typ des dritten Kindknotens nicht `T_DUMMY`, dann handelt es sich um eine Liste, die Variablen und/oder Methodendeklarationen enhält.

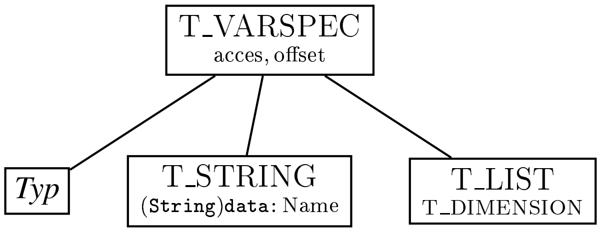
4.2 Variablendeklarationen

Eine Variablendeklaration besteht aus einem `T_VARSPEC`-Knoten und einem Ausdruck, der zur Initialisierung der Variable verwendet wird



Ein Variablenspezifikation besteht aus Typ, Name und einer Dimensionsliste, die leer sein kann:

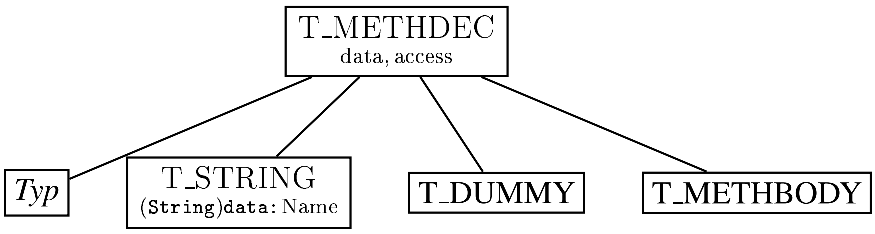


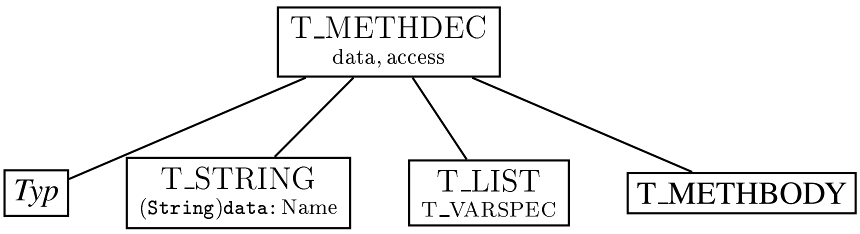


Das erste Kind repräsentiert den Typ der Variable, deren Name im zweiten Kind im `data`-Feld zu finden ist. Das dritte Kind – sofern es sich nicht um einen `T_DUMMY`-Knoten handelt – gibt Informationen über die Dimensionen der Variable: die Anzahl der `T_DIMENSION`-Knoten entspricht der Anzahl der Dimensionen der Variable. Das `access`-Feld gibt an, ob es sich um eine lokale Variable, eine Klassenvariable oder ein Objektfeld handelt.

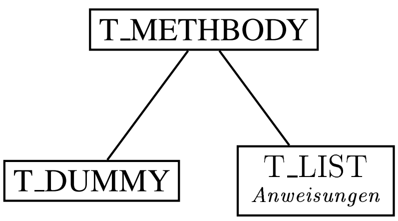
4.3 Methodendeklarationen

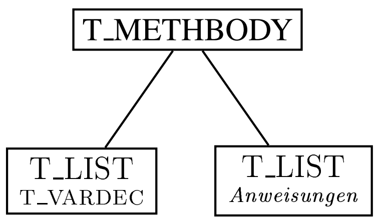
Methodendeklarationen gibt es in zwei Ausprägungen:





Erstes und zweites Kind beschreiben Typ und Name der Methode, deren Rumpf im vierten Kind zu finden ist. Erwartet die Methode formale Parameter, verweist das dritte Kind auf eine Liste aus `T_VARSPEC`-Knoten. Ist das `access`-Feld auf `T_STATIC` gesetzt, wurde die Methode als statisch deklariert. Das `data`-Feld referenziert ein `Integer`-Objekt, das die Anzahl der lokalen Variablen enhält.



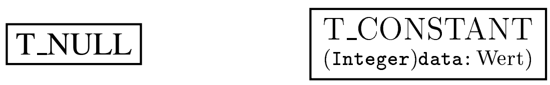


Das erste Kind eines Methodenrumpfes ist eine (möglicherweise leere) Liste von Variablendeklarationen. Das zweite Kind enthält eine Liste von Anweisungen.

5 Ausdrücke

5.1 Konstanten

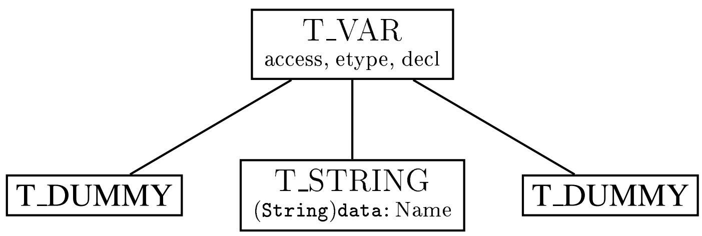
Es gibt lediglich zwei Arten von Konstanten, nämlich `null` und ganze Zahlen:

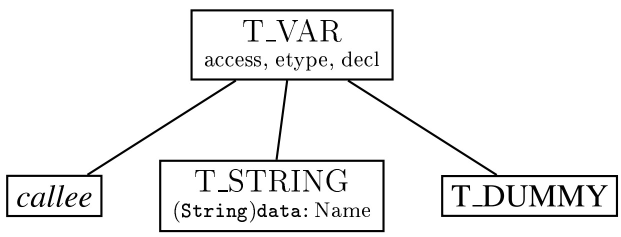


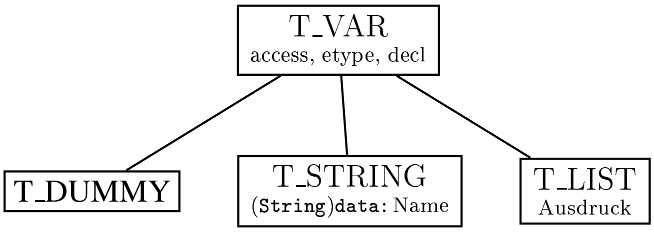
5.2 Variablen

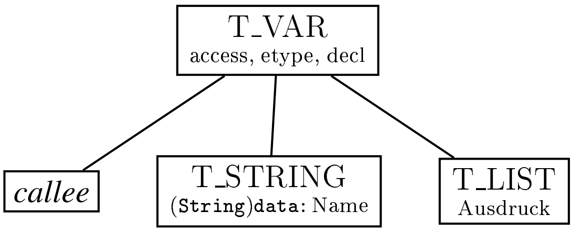
Ein `T_VAR`-Knoten verfügt über drei Kinder: einem *callee*, einem Namen und einer Ausdrucksliste. *callee* und Ausdrucksliste können `T_DUMMY`-Knoten sein.

Beim *callee* kann es sich um eine Zeichenkette (für Klassennamen), einen `T_VAR`-Knoten (für Objekte) oder um einem `T_THIS`-Knoten (zur Referenzierung des aktuellen Objekts) handeln.



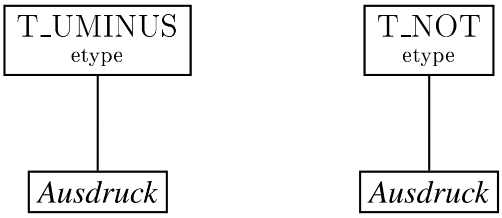






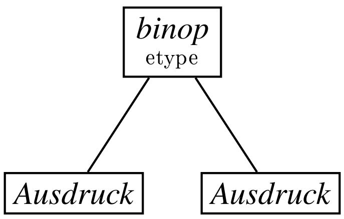
5.3 Unäre Operatoren

Unäre Operatoren besitzen lediglich ein Kind, nämlich einen Ausdruck. Das `etype`-Feld ist auf `'I'` gesetzt für `T_UMINUS`-Knoten, `'Z'` für `T_NOT`-Knoten:



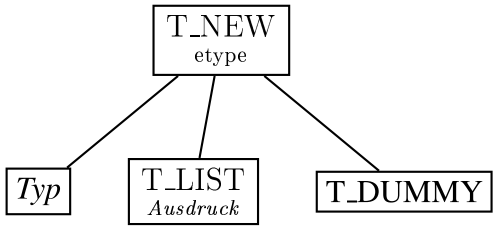
5.4 Binäre Operatoren

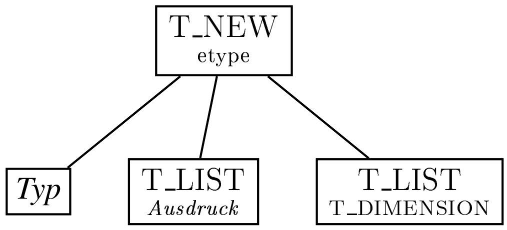
Binäre Operatoren kombinieren zwei Ausdrücke und haben somit zwei Kinder. Das `etype`-Feld enthält `'Z'` für `T_AND`- und `T_OR`-Knoten, sonst wird der Typ durch die beiden Ausdrücke und den Operator bestimmt.



5.5 New-Anweisung

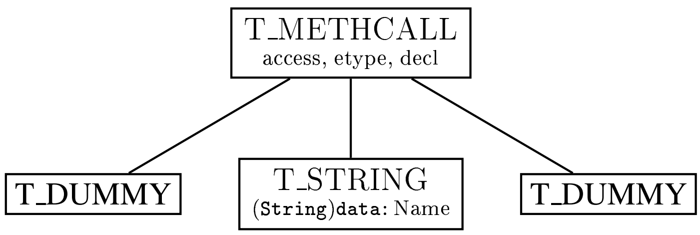
Ein `T_NEW`-Knoten besitzt drei Kinder: einen Typ, eine Ausdrucksliste und eine Dimensionsliste. Das zweite und dritte Kind können `T_DUMMY`-Knoten sein. Falls das zweite Kind ein `T_DUMMY`-Knoten ist, gilt dies auch für das dritte.

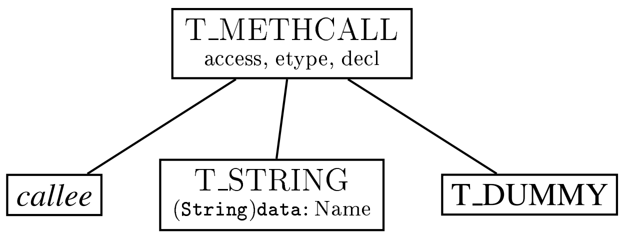


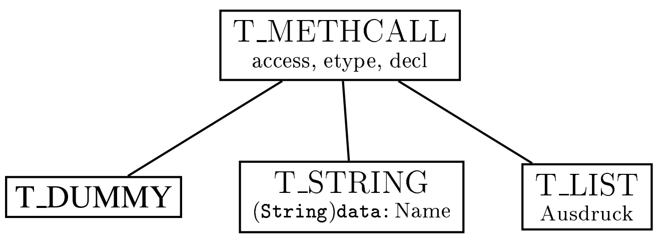


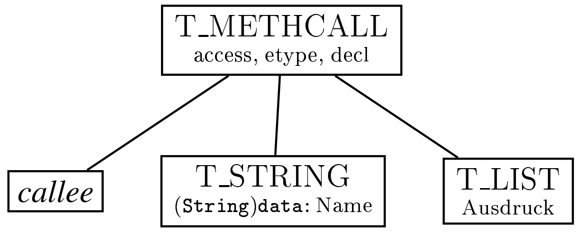
5.6 Methodenaufrufe

Knoten vom Typ `T_METHCALL` sind ähnlich aufgebaut wie `T_VAR`-Knoten: ein *callee*, ein Methodenname und eine Ausdrucksliste, die die aktuellen Parameter enthält, sind dessen Kinder. *callee* und Ausdrucksliste können `T_DUMMY`-Knoten sein.





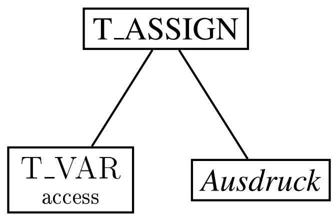




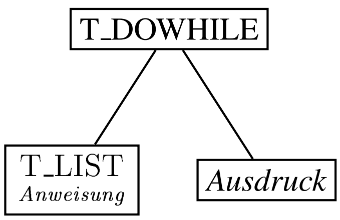
6 Anweisungen

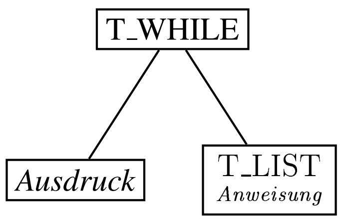
Die folgenden Knotentypen werden aus den bereits eingeführten Knotentypen zusammengesetzt. Die Semantik der neuen Knoten ist offensichtlich.

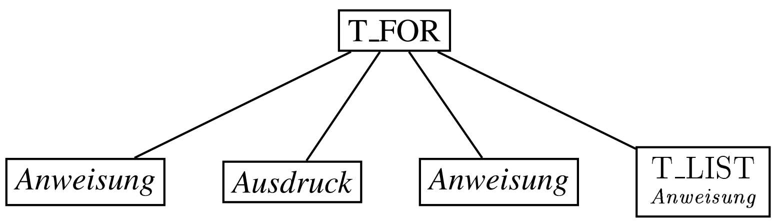
6.1 Zuweisungen



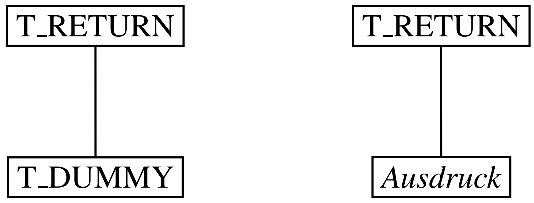
6.2 Schleifen





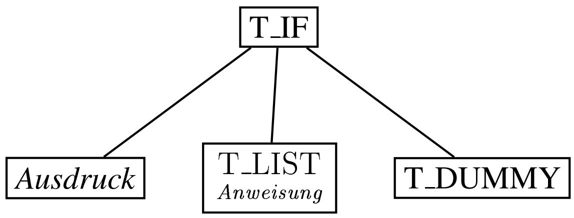


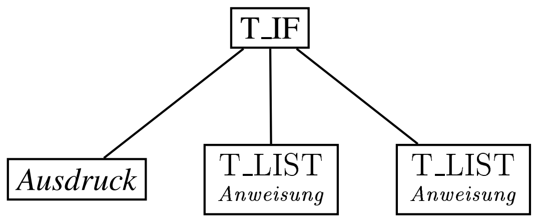
6.3 Return-Anweisungen



6.4 If-Anweisungen

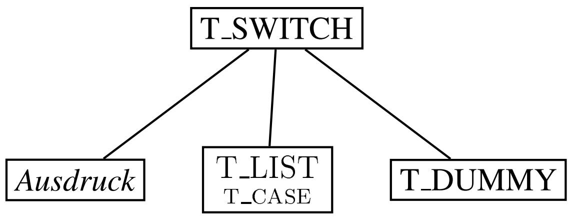
Das dritte Kind eines `T_IF`-Knotens repräsentiert den *else*-Teil einer *if*-Anweisung.

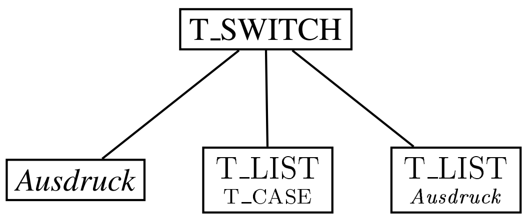


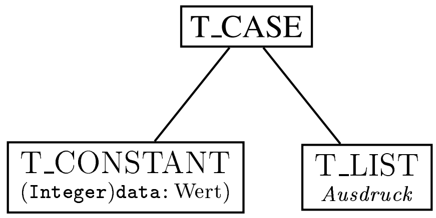


6.5 Switch-Anweisungen

Das dritte Kind eines `T_SWITCH`-Knotens repräsentiert das *default*-Label.






T_CASE
T_CONSTANT
(Integer)data: Wert)
T_LIST
Ausdruck