Max Wisniewski , Alexander Steen

Tutor: Ansgar Schneider

Aufgabe 1

Ändern Sie die Syntax von *WHILE*, indem Sie *INTEGER* und *REAL* Zahlen unterscheiden.

Lösung:
Wir müssen zunächst die Menge *ZAHL* in die Mengen *INTEGER* und *REAL* aufteilen. Haben wir dies erledigt, muss man noch alle Operationen so aufteilen, dass man nicht einfach REAL und INTEGER Zahlen mischen kann. Will man dies, geben wir noch eine explizite konversion von INTEGER nach REAL dazu.

Syntax
Elementare Einheiten:

INTEGER	::=	0 | 1 | -1 | 2 | -2 | ...
ZIFFERNFOLGE]	::=	0ZIFFERNFOLGE | ... | 9ZIFFERNFOLGE | ε
REAL	::=	INTEGER.ZIFFERNFOLGE
Z	::=	INTEGER | REAL

An den restlichen Definitionen muss nichts geändert werden, damit wir syntaktisch REAL und INTEGER unterscheiden können. Nun ist allerdings auch syntaktisch korrekt, die beiden Zahlenarten miteinander zu verrechnen. Dies könnte man zwar auch syntaktisch verbieten, in dem man z.B. die Terme T und die Operationen BOP und OP in zwei verschiedene Ausprägungen (je eine für REAL und INTEGER) unterteilt. Dann müsste man aber noch künstlichen Einschränkungen an die Variablenbezeichner anlegen, sodass hier ebenfalls unterschieden werden kann. Da dies allerdings nicht wirklich sinnvoll erscheint, müsste man gemischte Terme auf semantischer Ebene überprüfen und dort entscheiden, welches Verhalten gewünscht ist.

Aufgabe 2

Definieren Sie eine konkrete Syntax, die eindeutig ist.

Lösung:

Die elementaren Einheiten können diesmal unverändert bleiben.

Induktive Einheiten:

T	::=	Z | I | (T_1 OP T_2) | *read*
B	::=	W | *not* B | T_1 BOP T_2 | *read*
C	::=	*skip* | I := T | (C_1; C_2) | if B then C_1 else C_2 fi | while B do C od | *output* T | *output* B
P	::=	C

Wir haben nun jede Produktion, die aus mehr als einem Nicht-Terminal Symbol besteht, entweder mit Klammern oder mit einem Abschlussschlüsselwort versehen. So werden

arithmetische Operationen nun auf jedenfall durch eine Klammer abgeschlossen. Dies ist etwas übertrieben, da wir auf semantischer Ebene Bindungsstärken vereinbaren können, aber damit die Grammatik syntaktisch eindeutig ist, muss alles geklammert werden. Das *if* wird nach dem else durch ein Abschließendes *fi* ersetzt. Bei *while* wird das *do* durch ein *od* geschlossen. Da man den Gültigkeitsbereich nun explizit angeben muss, wird die konkrete Syntax eindeutig.
Die boolschen Operationen müssen nicht abgeschlossen werden, da wir in unserer Syntax boolsche Werte nicht weiter kombinieren können. So schließt eine boolsche Operation den Ausdruck an sich ab. Die Komposition sieht im ersten Moment komisch aus, wenn wir diese erzeugen, da wir nun viele unnütze Klammern haben, aber nur auf diese Weise können wir verhindern, dass $c_1; c_2; c_3$ auf 2 Arten interpretiert werden kann.

Aufgabe 3

Formulieren Sie informell eine Präzisierung der angegebenen WHILE-Semantik, die die genannte Fehlerquellen:

1. Bereichsüberschreitungen
2. Division durch Null
3. Berechnung von *read* bei leerer Eingabedatei
4. Typkonflikte

behandelt.

Lösung:
Beschreibung:

Anmerkung: Manche meiner Ideen haben eine Syntaxänderung zur folge. Keine Ahnung was wir hier machen sollen, da keine Fehlerbehandlung möglich ist, wäre wohl Termination das einzig mögliche.

Bereichsüberlauf: Wir könnten an dieser Stelle in *errno* z.B. 1 schreiben. Dies wird für alle Anweisungen, die nach dieser ausgeführt werden bedeuten, dass wir einen Overflow hatten. Im Falle von Integer, werden wir bei einem Überlauf bei MinValue starten und von dort aus den Rest addieren, bei einem Unterlauf, werden wir von MaxValue den Rest abziehen. Bei REAL Zahlen werden wir bei einem Überlauf INFTY und bei einem Unterlauf -INFTY ausgeben.

Division by zero: Wir werden hier in *errno* die Zahl *0* schreiben. Diese kann als Identifier benutzt werden um zu testen, ob durch 0 geteilt wurde. Als Wert werden wir die REAL Zahl NaN zurückgeben.

***read* bei leerer Datei:** Wir vergeben den Fehlercode *2* und schreiben bei Typ T eine 0 und bei B ein false. An dieser Stelle ist das erste mal wirklich notwendig auf den *errno* zurückzugreifen um den Fehler konkret zu verarbeiten.

Typkonflikte: Wir geben wie beim leeren lesen vor und schreiben diesmal *3* in *errno*. Konflikte können nur bei T und B auftreten, daher können wir so verfahren. Bei C in der Zuweisung kann es keine Probleme geben, da es keine statisch getypten Variablen gibt und es so zu keinem Konflikt kommen kann.