# Switch-Case Anweisung über int32 für IML

CPIB, HS2016 – Gruppe 10: Remo Rossi und Roman Meier

# Einleitung

## Switch-Case mit Beispiel

IML verfügt über keine Switch-Case Anweisung. Wenn auch diese über eine verschachtelte IF-Anweisung umsetzbar wäre, ist eine Switch-Case oft übersichtlicher und einfacher verständlich. Um die Funktionsweise und den Einsatz zu verdeutlichen: ein Besipiel.

program SwitchCase(in n:int32, out f:int32)

global

fun switchIntFun(n:int32) returns var res:int32

do

res init := 0;

switch n

case 1 : res := n + n;

res := n - n

case 2 : res := n \* n

case 3 : res := n \* n + n

casedefault : res := n

endswitch

endfun

do

f init := switchInt(n)

endprogram

Die Switch-Anweisung überprüft die Variable n und ruft dann den dazugehörigen Case, falls dieser existiert, oder den Default-Case auf. Danach springt sie zu «endswitch».

Da IML momentan zwei Typen (int und bool) unterstützt, lag es auf der Hand die Switch-Case Anweisung für diese beiden Typen zu implementieren. Während der Implementierung ist uns aufgefallen, dass die implementierung für bool sehr viele Fragen aufwirft und die Grammatik unnötig komplizierter macht. So haben wir uns entschlossen diese nicht zu entwerfen, zumal sie einfach mit der IF-Anweisung umsetzbar ist. Bei der Gestaltung der Syntax haben wir mit der Java Syntax der Switch Anweisung begonnen und Elemente, die wir für unnötig empfunden haben entfernt. So ist etwa die break-Anweisung entfallen. (Weshalb kein bool? -> andenken oke, implementation muss nicht sein)

## Lexikalische Erweiterung

Lexikalisch wird die Syntax um folgende Token erweitert:

|  |  |
| --- | --- |
| Pattern | Token |
| switch | SWITCHINT32 |
| endswitch | ENDSWITCH |
| case | CASE |
| casedefault | CASEDEFAULT |

## Grammatikalische Erweiterung

Man sieht, es muss mindestens ein Case geben und auf jeden Fall den default Case. Die restliche Grammatik wurde 1:1 von IML übernommen. Diese Erweiterung wurde mit FixFoxi überprüft und als LL(1) Grammatik eingestuft.

Cmd ::= SKIP

| expr BECOMES expr

| IF expr THEN cpsCmd ELSE cpsCmd ENDIF

| WHILE expr DO cpsCmd ENDWHILE

| CALL IDENT exprList [globInits]

| DEBUGIN expr

| DEBUGOUT expr

| SWITCH expr

CASE LITERAL COLON cpsCmd

{CASE LITERAL COLON cpsCmd}

CASEDEFAULT COLON cpsCmd

ENDSWITCH

Warum DefaultCase? Literal nach case, damit man einen typfehler zur compilationszeit erkennen kann?

## Typ- und Kontexteinschränkungen

1. Die Switch Anweisung kann nur auf int32 angewendet warden. Der Typchecker stellt dies sicher.
2. Der Typ der «expr» muss mit den Typen der Casebezeichner übereinstimmen. Der Typchecker überprüft dies.
3. Was mit mehrfach gleicher Case?

## Implementations Details

Implementationstechnisch halten wir uns möglichst genau an die Vorlage der Slides. Programmiert wird in Java. Um die Implementation in IML möglichst einfach zu halten, versuchen wir uns möglichst gut an die bisherige Syntax zu halten. Unsere Implementierung des Scanners lässt keine Kommentare oder Hochkommata zu. Was ist mit Tabs und Spaces?

## Mögliche Erweiterungen

Oder Verknüpfung der cases.

Gleicher weg von cases.