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

Programmierung 1 (Wintersemester 2012/13)

Erklärung 1

(Ausführung von Programmen)

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.

In diesem Dokumenten soll die interne Arbeitsweise des Interpreters noch einmal betrachtet werden. Dabei werden die Regeln, die in Kapitel 2 eingeführt werden nocheinmal näher betrachtet. Im Anschluss an diese Erläuterung sollten Sie sich Aufgabe 3.18 erneut anschauen, um Ihr Wissen an einem komplizierteren Beispiel zu erproben.

Wir wollen folgendes Beispielprogramm dabei ausgewerten:

```
val a = 7
fun f x = a + x
val a = a + 2
val x = f 5
```

Wie bei jedem SML-Programm handelt es sich hierbei um eine Folge von Deklarationen, die eine nach der anderen (vom obersten zum untersten) bearbeitet werden. Die Ausführung beginnt dabei in der leeren Umgebung [ ], d.h. wenn der Interpreter die val-Deklaration

```
val a = 7
```

betrachtet, ist noch kein anderer Bezeichner gebunden.
Obwohl die Umgebung noch leer ist, wertet der Interpreter zuerst den Ausdruck auf der rechten Seite aus und nutzt dafr die bisher vorhandene (leere) Umgebung. Die Auswertung von 7 liefert natrlich den Wert 7. Anschlieend wird der Bezeichner, hier a, an den berechneten Wert gebunden. Diese Bindung wird der Umgebung hinzugefgt. Die neue Umgebung nach Ausführung der ersten Deklaration lautet also $[a := 7]$. Die Auswertung wird dann mit der zweiten Deklaration

```
fun f x = a + x
```

fortgesetzt. Diese beginnt mit `fun`, also handelt es sich um eine Prozedurdeklaration. Somit wird der Bezeichner f an die Tripeldarstellung einer Prozedur gebunden. Zur Analyse des Ausdrucks auf der rechten Seite wird wieder die aktuelle Umgebung herangezogen, es erfolgt allerdings *keine* Auswertung dieses Ausdrucks. Lediglich die Stufen der statischen Analyse werden durchlaufen, das heit, unter anderem wird der Rumpf der `fun`-Deklaration auf syntaktische und Typ-Korrektheit überprft. Wir können den Prozedurrumpf nicht auswerten, da der Wert des Argumentes nicht bekannt ist. Wie wir später noch sehen werden, wird der Rumpf einer Prozedur erst dann ausgewertet, wenn die Prozedur angwendet wird, denn erst dann, beim Prozeduraufruf, wird die Argumentvariable an einen Wert gebunden. Der Bezeichner a, der im Rumpf von f vorkommt, ist zu diesem Zeitpunkt jedoch bereits gebunden. Auch wenn der Bezeichner a, wie in unserem Beispiel, danach erneut in einer Deklaration gebunden wird, ist der Wert, der innerhalb des Rumpfs von f an a gebunden wird, der zum Zeitpunkt der Auswertung der Prozedur (nicht ihrer Anwendung!) gültige. Aus diesem Grund enthält die Tripeldarstellung der Prozedur auch die aktuell gültige Umgebung, jedoch reduziert auf die Bezeichner, die auch im Prozedurrumpf vorkommen. Damit sieht die neue Umgebung wie folgt aus: $[a := 7, f := (\mathit{fun}\ f\ x = a + x, \mathit{int} \to \mathit{int}, [a := 7])]$

Die dritte Deklaration

```
val a = a + 2
```

ist wieder eine `val`-Deklaration, also wird zuerst der Ausdruck auf der rechten Seite $(a + 7)$ in der oben angegebenen Umgebung ausgewertet. Das ergibt offensichtlich $7 + 2 = 9$, da a noch an 7 gebunden ist. Der Bezeichner a wird nun 9 gebunden. Man möchte sagen, die alte Bindung von a wird überschrieben. In der Tat ist es jedoch semantisch korrekter, wenn wir uns vorstellen, dass die alte Bindung nur nicht mehr sichtbar ist, da wir zweimal den selben Bezeichner a verwendet haben. Man kann Bezeichnerbindungen aber überschreiben, so wie Variablen in imperativen Programmiersprachen. Wir erhalten nun die Umgebung $[f := (\mathit{fun}\ f\ x = a + x, \mathit{int} \to \mathit{int}, [a := 7]), a := 9]$.
Man beachte, dass in der Tripeldarstellung der Prozedur, die an f gebunden ist, der Bezeichner a weiterhin an den alten Wert gebunden ist. Dieser wird also nicht geändert!
Die letzte `val`-Deklaration bindet die Variable x an die Auswertung des Ausdrucks $f\ 5$. Dabei handelt es sich diesmal um eine Prozeduranwendung. Nun muss die Anwendung ausgeführt werden, wofür der Interpreter sich eine neue Umgebung öffnet. Diese wird nur vorübergehend erstellt und ist vollständig getrennt von der durch das Programm berechneten Umgebung. Das bedeutet genauer, wenn wir im folgenden neue Bindungen in diese temporäre Umgebung einfügen (z.B. in einem let-Ausdruck), werden diese nicht in Umgebung eingefügt, die wir bisher betrachtet haben. Die temporäre Umgebung besteht aus der Umgebung der Tripeldarstellung der Prozedur zusammen mit der Bindung der Prozedur fr den rekursiven Selbstaufruf sowie der Bindung der Argumente.$([a := 7] + [f := (\mathit{fun}\ f\ x = a + x, \mathit{int} \to \mathit{int}, [a := 7])]) + [x := 5]$ $= [a := 7, f := (\mathit{fun}\ f\ x = a + x, \mathit{int} \to \mathit{int}, [a := 7]), x := 5]$
In dieser Umgebung wird nun der Rumpf der Prozedur, $a + x$ ausgewertet. Sowohl a als auch x sind hier definiert, das Ergebnis lautet dann 12 und die Prozeduranwendung ist abgeschlossen. Man beachte nochmals, dass a hier an 7 gebunden ist, da dies die gültige Bindung zu dem Zeitpunkt war, als f deklariert wurde.
Unsere temporär erstellte Umgebung wird nun verworfen. Der Ausdruck $f\ 5$ hat also den Wert 12, und daran wird x gebunden. Das Programm berechnet also insgesamt folgende Umgebung:

$[f := (\text{fun f x} = a + x, \mathit{int} \to \mathit{int}, [a := 7]), a := 9, x := 12]$