

12. Ausnahmen (exceptions)

Dynamische Fehler werden durch Auslösen von Ausnahmen gemeldet. (statische Fehler durch das Typsystem).

z.B 5 div 0 statisch korrekt, hat Typ int
 dynamisch wird die Ausnahme Div ausgelöst

12.1 Eingebaute Ausnahmen

Div	Division durch 0
Overflow	Überschreiten des Zahlenbereichs
Chr	Anwendung der Funktion chr auf eine ganze Zahl, die nicht ASCII-Code eines Zeichens ist
Empty	Anwendung hd, tl auf nil

Match:

Bei Definition von Funktionen über Fälle und mit nicht ausschöpfenden Falllisten. Interpreter warnt: „non exhaustive match“.

Bei Funktionsanwendung auf nicht abgedeckten Fall: Ausnahme Match

Bind:

val pat = exp

pat passt nicht auf exp, dann Ausnahme Bind

Beide Ausnahmen können nur auftreten wenn die Warnung „non exhaustive match“ gekommen ist.

Notation: 5 div 0 \Downarrow raise Div

Einschub:

Konstruktoren – Destruktoren

konstruieren zerlegen Werte

- Konstruktoren in Ausdrücken: konstruieren Werte
- Konstruktoren in Mustern: zerlegen Werte

12.2 Benutzerdefinierte Ausnahmen

Der Programmierer kann

- Ausnahmen deklarieren
- festlegen, wann sie ausgelöst werden
- sie behandeln

Beispiel: Fakultätsfunktion mit Ausnahme für negative Argumente `efact`

Typproblematik: Im Fall LESS gibt `efact` keinen Wert vom Typ `int` zurück

Widerspruch `efact: int → int ?`

Im Fall LESS liefert `efact` keinen Wert sondern hat einen Effekt, nämlich das Auslösen der Ausnahme `Factorial`.

Nachteil von `efact`: prüft bei jedem Aufruf auf negative Argumente

12.3 Behandeln von Ausnahmen

Bisher: Auslösen von Ausnahmen führt zu Programmabbruch

Jetzt: Abfangen und Behandeln von Ausnahmen und Fortfahren mit Programmausführung, eventuell auf andere Art

Ausnahmebehandler (exception handler)

- fängt Ausnahme ab
- assoziiert mit Ausdruck; zuständig für Ausnahmen, die bei der Auswertung des Ausdrucks ausgelöst werden

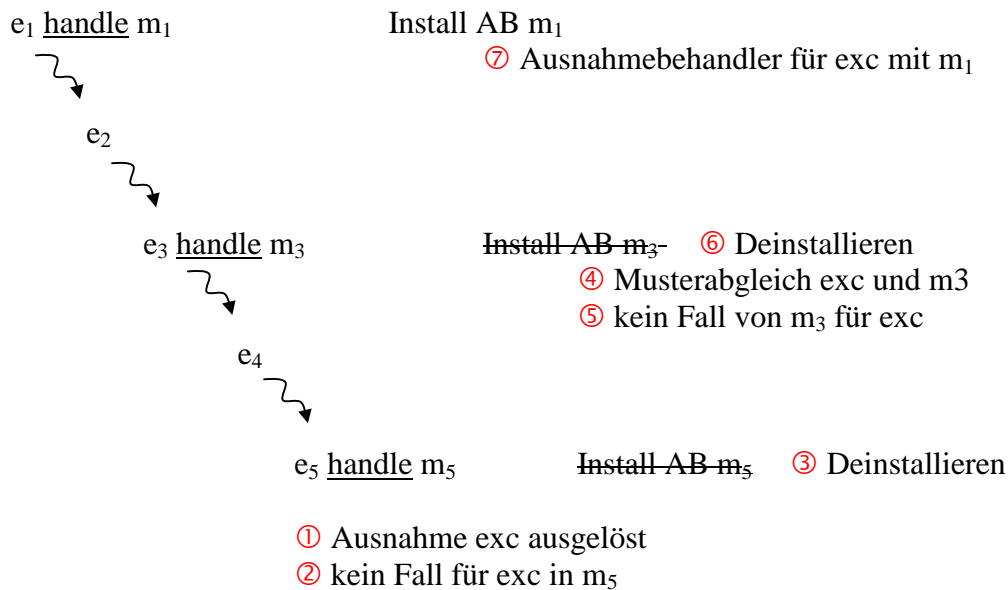
`e handle m` `e` Ausdruck
 `m` Fallliste,

definiert alle Ausnahmen für deren Behandlung bei der Auswertung von `e` dieser Ausnahmebehandler zuständig ist

Semantik von $e' \equiv e \text{ handle } m$

1. `e` wird ausgewertet
Tritt keine Ausnahme auf und ist Wert `v`, so ist der Wert von `e'` `v`.
2. Tritt eine Ausnahme `exc` auf, dann wird ein Musterabgleich zwischen `exc` und `m` gemacht.
3. Trifft ein Fall $p \Rightarrow e''$ aus `m` auf `exc`, so wird `e''` (in der durch Musterabgleich entstandenen Wertumgebung) ausgewertet. Der Wert ist der Wert von `e'`.
4. Trifft keiner der Fälle von `m` auf `exc`, so wird erneut die Ausnahme `exc` ausgelöst und macht mit einem „dynamischen Vorgänger“ weiter. Der dynamische Vorgänger ist ein Ausnahmebehandler, der mit einem Ausdruck `e'''` assoziiert ist, dessen Auswertung die Auswertung `e` erforderte. Genauer gesagt der „jüngste dynamische Vorgänger“ wird versucht.

$e \rightsquigarrow e'$: Auswertung von `e` führt zur Auswertung von `e'`



Auswertung von e_i handle m_i

1. Auswertung von e_i mit aktueller AB m_i
 $e_i \Downarrow v_i$ Deinstallation von m_i
 vorherige AB wird aktiv
2. Ausnahme exc ausgelöst: behandelt mit AB m_i
3. Fall: $p \Rightarrow e''$ trifft exc
 AB m_i wird deinstalliert
 vorheriger AB wird aktueller AB
 e'' wird mit diesem AB ausgewertet
4. kein Fall von m_i trifft exc: AB m_i deinstalliert
 vorheriger AB wird aktueller AB
 exc wird an ihn übergeben

Nichnullstellige Ausnahmen

Ausnahmen sind ähnlich zu Wertkonstruktoren
 Beim Auslösen kann man ein Argument mitgeben

Beispiel: fun find in Abschnitt 10.7

exception Undecl Var of string

fun find (s,nil) = raise Undecl Var s

| find (s, (s',n) :: rest) =

if s = s' then n else find (s,rest)

Bemerkungen zu Ausnahmen in SML

1. Erzwingt Betrachtung der Ausnahmefälle
2. Trennt „normalen“ Code von Code für Ausnahmen
3. Erlauben das Programmieren mit „Zurücksetzen“ (back tracking): Gängige Technik für das Durchmustern von Suchräumen