3.3.3 Pattern-Matching

ML stellt mit dem Pattern-Matching ein leistungsfähiges und übersichtliches Konzept zur Parameterübergabe zur Verfügung, mit dem man auf eine explizite Anwendung von Selektoren auf Datenstrukturen und die damit verbundene explizite Fallunterscheidung durch geschachtelte bedingte Ausdrücke in Abhängigkeit vom Aufbau der Datenstruktur weitgehend verzichten kann. Stattdessen gibt man Muster für den aktuellen Parameter an und legt für jedes Muster fest, welchen Wert eine Funktion besitzen soll, wenn der Parameter in das Muster paßt. Die Zuordnung der Bezeichner und Strukturen des aktuellen Parameters an das Muster des formalen Parameters (*Matching*) wird von ML übernommen. Implementiert wird das Pattern-Matching durch einen effizienten Algorithmus zur Ermittlung der Isomorphie von geordneten, markierten Bäumen.

Beispiele:

1) Wir definieren die Funktion

   and: bool×bool→bool.

   Ohne Nutzung des Pattern-Matching definiert man wie üblich

```
fun and(x,y) = if x then
                   if y then true else false
               else false.
```

   Mit Pattern-Matching schreiben wir stattdessen übersichtlicher

```
fun  and(true,true) = true |
     and(_,_) = false.
```

   Hier haben wir zwei Muster für den aktuellen Parameter vorgegeben, das Muster

   (true,true)

   und das Muster

   (_,_).

   Paßt der aktuelle Parameter (a,b) in das erste Muster, also a=b=true, so ist true das Ergebnis der Funktion and. Jede andere Belegung des aktuellen Parameters paßt in das Muster (_,_) und führt zum Funktionswert false. Hierbei steht das Zeichen _ (sog. *wildcard*) für ein nicht spezifiziertes Objekt eines Musters. Es kann mit jedem Objekt gematcht werden.

2) Wir kodieren eine 2×2-Matrix zeilenweise durch ein Paar von Paaren. Eine Funktion, die eine Matrix daraufhin überprüft, ob es sich um die Einheitsmatrix handelt, lautet:

```
fun  einheit ((1,0),(0,1)) = true |
     einheit (_,_) = false.
```

   Um zu überprüfen, ob die Elemente der Hauptdiagonalen übereinstimmen, verwendet man:

```
fun  eqdiag ((x,_),(_,y) = (x=y).
```

   Abb. 1 veranschaulicht die Abläufe beim Aufruf von eqdiag((7,2),(3,7)).

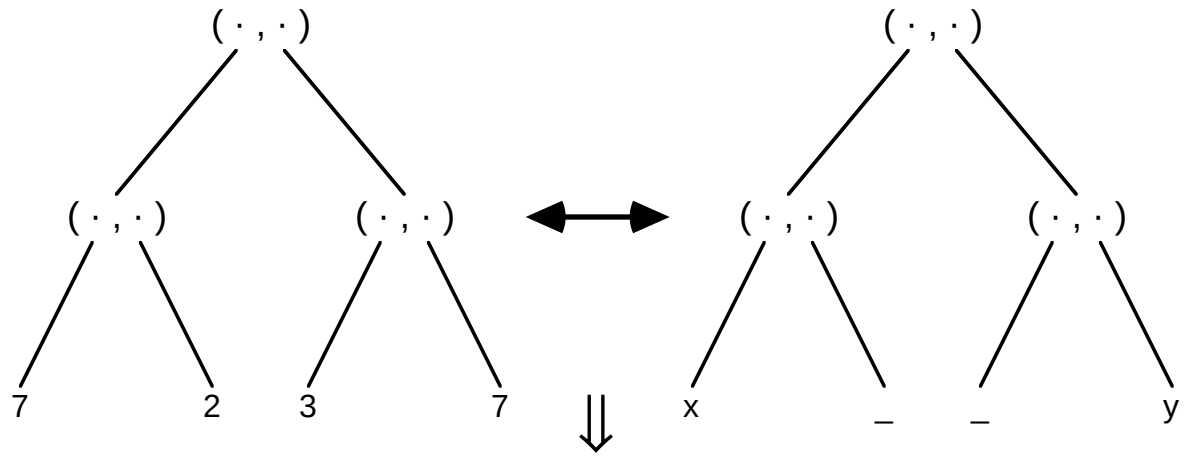

Abb. 1: Pattern-Matching

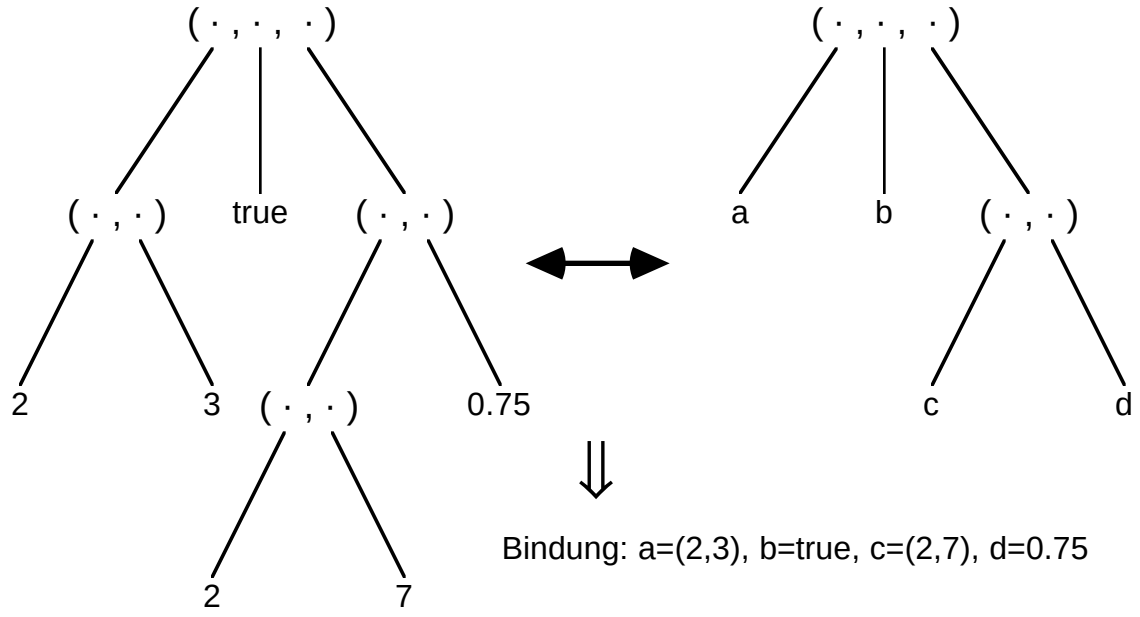

Abb. 2: Pattern-Matching

3) Pattern-Matching funktioniert auch in Verbindung mit Wertdeklarationen (Abb. 2):

val (a,b,(c,d))=((4-2,3),true,((2,7),3.0/4.0)).

Definition A:

Ein **Pattern** in ML ist ein Ausdruck, der nur aus Bezeichnern für Variablen, Konstruktoren und dem Zeichen _ (wildcard) besteht. Dabei müssen alle Variablenbezeichner paarweise verschieden sein.

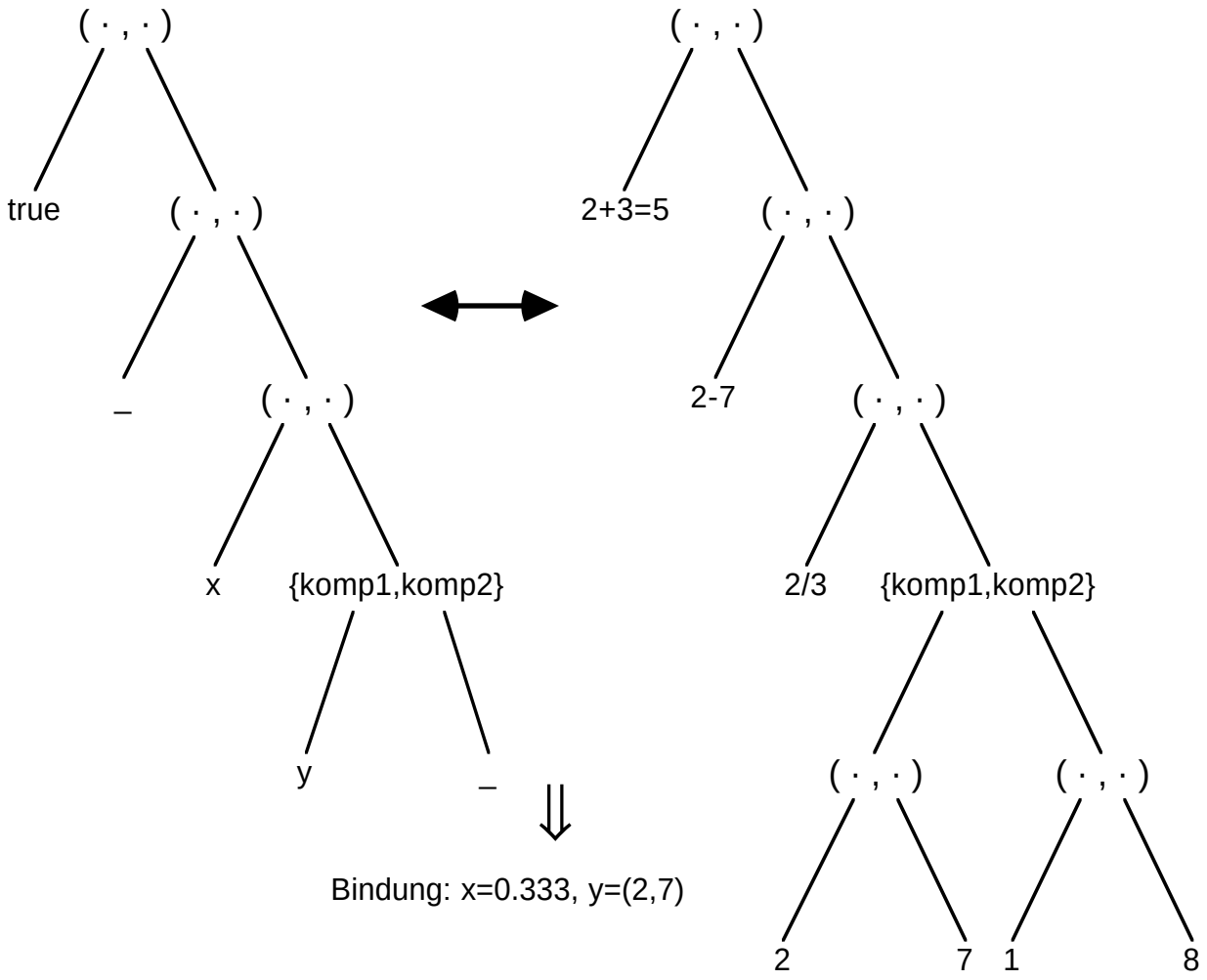

Abb. 3: Pattern-Matching

Es ist zu beachten, daß zu den Konstruktoren auch die Konstanten der Grundtypen int, real, bool, string gehören (es sind 0-stellige Konstruktoren) sowie die Standardkonstruktoren

(·,·,...,·) (Tupel)
{·,·,...,·} (Records)
[·,·,...,·] (Listen, s. 3.3.6).

Wie werden nun die Objekte eines Musters einem konkreten Objekt zugeordnet (gematcht)? Hierfür gelten folgende Regeln:

1. *Variablen:* Variablen können mit jedem Objekt, sei es strukturiert oder elementar, gematcht werden.
   Beispiel: fun id x =x.
   Hier ist x ein Muster, das mit jedem aktuellen Parameter gematcht werden kann, z.B.
   id(2,3) Bindung: x=(2,3)
   id({zaehler=2,nenner=3},(2,true)) Bindung: x=({zaehler=2,nenner=3},(2,true)).
2. *Wildcards:* Sie können wie Variablen mit jedem Objekt gematcht werden. Im Unterschied zu Variablen wird jedoch keine Bindung durchgeführt.

Beispiel: fun projx(x,_)=x;
projx(2,(("a","b"),true)) Bindung: x=2.

3. *Konstruktoren:* Ein Konstruktor innerhalb eines Pattern kann nur mit identischen Objekten gematcht werden. Auch hier wird im Erfolgsfall keine Bindung durchgeführt.
   Beispiel: fun test(true,(_,(x,{komp1=y,komp2=_}))) = ...
   Aufruf: test(2+3=5,(2-7,(2.0/3.0,{komp1=(2,7),komp2=(1,8)}))).
   Bindung: Abb. 3.

Zur übersichtlichen Steuerung unterschiedlicher Aktionen in Abhängigkeit von dem Muster, das einem Argument zugrundeliegt, bietet ML die Möglichkeit, mehrere Musteralternativen bei Funktionsdefinitionen anzugeben. Das System versucht dann die Alternativen der Reihe nach gegen das Argument zu matchen, bis eine Variante Erfolg hat. Die einzelnen Alternativen trennt man durch einen senkrechten Strich.

Beispiele:

1)
```
fun  not true = false |
     not false = true.
```

2) Beim Kinderspiel Papier, Schere, Stein gilt folgender Gewinnplan:

gewinnt gegen	Papier	Schere	Stein
Papier	X	-	+
Schere	+	X	-
Stein	-	+	X

In ML:
```
datatype Spiel = Papier | Schere | Stein;
fun  gewinnt (Papier,Stein) = true |
     gewinnt (Schere,Papier) = true |
     gewinnt (Stein,Schere) = true |
     gewinnt (_,_) = false.
```

Eine besondere Unterstützung bei der Vermeidung von Fehlern bietet ML dem Benutzer, der die Möglichkeit des Pattern-Matching intensiv nutzt: Das ML-System erkennt einerseits, wenn die für eine Funktion angegebenen Musteralternativen nicht alle möglichen Fälle abdecken. Andererseits meldet es auch redundante Alternativen, also Muster, die bereits durch die vorhergehenden vollständig erfaßt werden. Auf diese Weise werden bereits frühzeitig typische Programmierfehler erkannt.