Informatik I

4. Zusammengesetzte und gemischte Daten

Jan-Georg Smaus

Albert-Ludwigs-Universität Freiburg

28. Oktober 2010

Informatik I

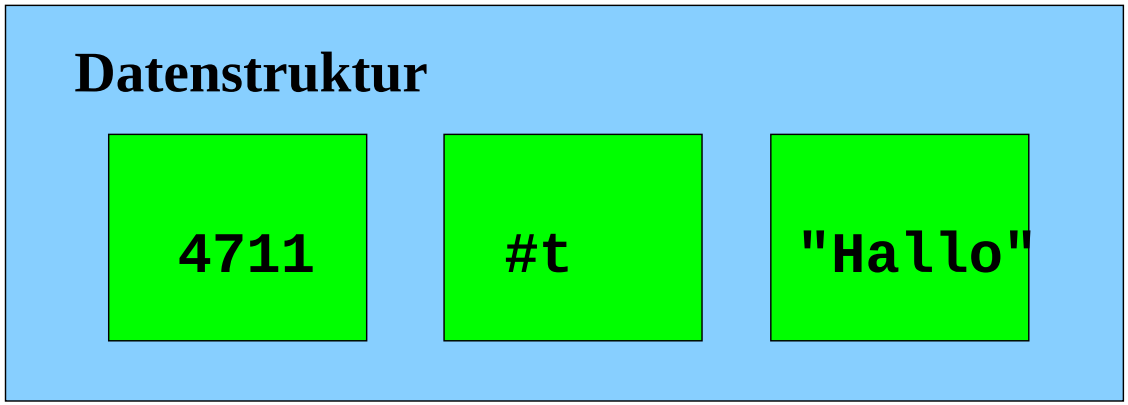
28. Oktober 2010 — 4. Zusammengesetzte und gemischte Daten

4.1 Zusammengesetzte Daten

4.2 Gemischte Daten

4.3 Zusammenfassung

Zusammengesetzte Daten



- Ein Wert einer zusammengesetzten Sorte besteht aus mehreren Werten, die zu unterschiedlichen Sorten gehören können.
- Record, Produkt
- Beispiele
  - Schokokeks
  - Punkt im kartesischen Koordinatensystem
  - Viergängemenü

Gemischte Daten

- Eine gemischte Sorte ist die Vereinigung aus mehreren unterschiedlichen Sorten. Ein Wert einer gemischten Sorte ist demnach ein Wert einer der Teilsorten.
- Variante, Summe
- Beispiele
  - Keks = entweder Schokokeks oder Marmeladenkeks
  - Punkt = entweder kartesischer Punkt oder Punkt im Polarkoordinatensystem
  - Essen = entweder Frühstück oder Mittagessen oder Abendessen
- Kennzeichen einer gemischten Sorte:
  Gemeinsame Operationen auf allen Alternativen

4.1 Zusammengesetzte Daten

- Eingebaute Kekse
- Selbst gebaute kartesische Koordinaten
- Allgemeine Form
- Rechnen mit Records

Der Schokokeks

Wir wollen Schokokekse modellieren. Ein Schokokeks sei definiert durch seinen Schokoladenanteil und seinen Keksanteil. Z.B.

Komponente	Wert
Schoko:	12g
Keks:	14g

Schokokekse in Scheme

Wir brauchen eine Sorte `chocolate-cookie`. Diese ist in der DrRacket-Sprachebene `Die Macht der Abstraktion - Anfänger` schon vordefiniert.
Jetzt stellen Sie sich vielleicht die Frage:

- Wie sieht ein Schokokeks in Scheme aus?
- Oder: Wie schreibe ich einen Schokokeks in Scheme hin?

Das ist gar nicht so einfach! Rekapitulieren wir: wie sahen denn unsere bisherigen Daten aus? Es waren Literale

```
42  -17  2/3  3.14 "Banane" "gaseous" "Grossbrief"
```

die so aussahen und die man auch so hinschrieb.
Bei zusammengesetzten Daten müssen wir uns von der Vorstellung des „Hinschreibens“ oder „Aussehens“ ein wenig verabschieden . . .

Konstruktion eines Schokokekses

Ein Schokokeks wird nicht einfach „hingeschrieben“, er wird konstruiert, und zwar aus dem Schokoladenanteil und dem Keksanteil. Die Konstruktion hat die Signatur

```
; Schokokeks konstruieren
(: make-chocolate-cookie
                (real real -> chocolate-cookie))
```

`make-chocolate-cookie` ist der Konstruktor, der auf zwei Argumente angewendet wird. Das Ergebnis ist ein Wert:

```
   (make-chocolate-cookie 12 14)          (1)
=> #<record:chocolate-cookie 12 14>       (2)
```

Wir schreiben also (1) hin, aber der Keks sieht nicht so aus! Vielmehr löst das Hinschreiben eine „echte Berechnung“ aus. Eher noch sieht der Keks aus wie in (2), aber das ist recht willkürlich.

Schokokekse an Variable binden

Ein Schokokeks kann an eine Variable gebunden werden:

```
   (define doppelkeks (make-chocolate-cookie 12 28))
   doppelkeks
=> #<record:chocolate-cookie 12 28>
```

Schokokekse zerlegen

Quelle User:Eldred http://upload.wikimedia.org/wikipedia/commons/thumb/a/a6/Doppelkeks.jpg/800px-Doppelkeks.jpg

Das Gegenstück zum Konstruktor sind die Selektoren. Mit ihnen kann man auf die Komponenten zugreifen.

```
; Selektor: Schoko-Anteil ermitteln
(: chocolate-cookie-chocolate
       (chocolate-cookie -> real))
```

Beispiel:

```
   (chocolate-cookie-chocolate doppelkeks)
=> 12
```

Schokokekse zerlegen II

Quelle User:Eldred http://upload.wikimedia.org/wikipedia/commons/thumb/a/a6/Doppelkeks.jpg/800px-Doppelkeks.jpg

Das Gegenstück zum Konstruktor sind die Selektoren. Mit ihnen kann man auf die Komponenten zugreifen.

```
; Selektor: Keks-Anteil ermitteln
(: chocolate-cookie-cookie
       (chocolate-cookie -> real))
```

Beispiel:

```
   (chocolate-cookie-cookie doppelkeks)
=> 28
```

Gleichungen für Konstruktoren und Selektoren

Das Zerlegen eines gerade konstruierten Schokokekses liefert wieder genau die ursprünglichen Komponenten. Es gelten die folgenden definierenden Gleichungen für Selektoren:

```
(chocolate-cookie-chocolate
                (make-chocolate-cookie x y)) = x
(chocolate-cookie-cookie
                (make-chocolate-cookie x y)) = y
```

Sortenprädikat für Schokokekse

Wir kennen schon Konstruktoren und Selektoren. Jetzt fehlt noch das so genannte Sortenprädikat:

```
   (: chocolate-cookie? (%value -> boolean)))
   (chocolate-cookie? doppelkeks)
=> #t
   (chocolate-cookie? 4711)
=> #f
   (chocolate-cookie? "chocolate-cookie")
=> #f
```

Gewicht eines Schokokekses

Das Gewicht eines Schokokekses ist die Summe der Gewichte der Komponenten. Mit bisherigen Konstruktionsanleitungen:

```
; Gewicht eines Schokokekses bestimmen
(: chocolate-cookie-weight (chocolate-cookie -> real))
(define chocolate-cookie-weight
  (lambda (c)
    ...))
```

Da c von der Sorte `chocolate-cookie` ist, müssen im Rumpf die Selektoren verwendet werden.

```
(define chocolate-cookie-weight
  (lambda (c)
    (+ (chocolate-cookie-chocolate c)
       (chocolate-cookie-cookie c))))
```

Wir werden das später noch als Konstruktionsanleitung präzisieren.

Kekse zusammengefasst: Records

- Zusammengesetzte Daten heißen in Scheme Records, d.h., ein Record ist eine bestimmte Art von Sorte.
- Ein Record muss einen Konstruktor, 0 oder mehr Selektoren und ein Sortenprädikat haben.
- Ein Element eines Records, z.B. einen Keks, kann man nicht einfach „hinschreiben“ und „anschauen“; stattdessen muss man es konstruieren und daraus selektieren.
- Die Recordsorte `chocolate-cookie` ist in DrRacket vordefiniert, aber natürlich kann man Records auch selbst definieren ...

Definition von Records

Beispiel: kartesische Koordinaten

Ein Punkt im kartesischen Koordinatensystem in der Ebene besteht aus einer X- und einer Y-Koordinate.
Was muss definiert werden?

- Name der Sorte: `cartesian`
- Name des Konstruktors: `make-cartesian`
- Name des Sortenprädikats (Sortentests): `cartesian?`
- Die Namen der Selektoren: `cartesian-x`, `cartesian-y`

Die Record-Definition in Scheme lautet:

```
(define-record-procedures cartesian
  make-cartesian cartesian?
  (cartesian-x cartesian-y))
```

Welche Sorte müssen die Koordinaten haben?

Verwendung der kartesischen Koordinaten

Konstruktor

```
; kartesischen Punkt konstruieren
(: make-cartesian (real real -> cartesian))
```

Es empfiehlt sich unbedingt, diese Signatur in das Programm zu schreiben, denn nur dadurch ist festgelegt, dass die Koordinaten von der Sorte `real` sind. U.U. kann DrRacket dann Signaturverletzungen erkennen.

Beispiele

```
   (make-cartesian 0 0)
=> #<record:cartesian 0 0>
   (make-cartesian 100 50)
=> #<record:cartesian 100 50>
```

Sortenprädikat

```
; Test auf Vorliegen eines kartesischen Punktes
(: cartesian? (%value -> boolean))
```

Die Sorte eines Sortenprädikats ist zwar immer `%... -> boolean`, aber es empfiehlt sich trotzdem, um der Klarheit willen die Signatur hinzuschreiben.

Beispiele

```
   (cartesian? #t)
=> #f
   (cartesian? (make-cartesian 17 4))
=> #t
   (cartesian? (make-chocolate-cookie 22 22))
=> #f
```

Ca. ab November 2010 wird es statt `%...` die Sorte `any` geben.

Selektoren

```
(: cartesian-x (cartesian -> real))
(: cartesian-y (cartesian -> real))
```

Die Sorten der Selektoren ergeben sich automatisch aus der Sorte des Konstruktors, aber es empfiehlt sich trotzdem, um der Klarheit willen Signaturen hinzuschreiben.

Beispiele

```
   (define mypoint (make-cartesian 1.0 2.5))
   (cartesian-x mypoint)
=> 1.0
   (cartesian-y mypoint)
=> 2.5
```

Definierende Gleichungen

(cartesian-x (make-cartesian x y)) $= x$
(cartesian-y (make-cartesian x y)) $= y$

Diese Gleichungen gelten automatisch aufgrund des Konstruktes `define-record-procedures`.

Beispiele

```
   (cartesian-x (make-cartesian 17 4))
=> 17
   (cartesian-y (make-cartesian 17 4))
=> 4
   (cartesian-x (make-cartesian -1 6/7))
=> -1
   (cartesian-y (make-cartesian -1 6/7))
=> 0.857142
```

Allgemeine Form von `define-record-procedures`

```
(define-record-procedures t
  c p
  (s1 ... sn))
```

definiert eine zusammengesetzte Sorte (Record) mit

- t ist der Name der definierten Sorte
- c ist der Name des Konstruktors
- p ist der Name des Sortenprädikats
- s_i sind die Namen der Selektoren

Für die Namen des Konstruktors, des Sortenprädikat und der Selektoren gelten folgende Konventionen: `make-`t, t`?` und t`-...`.
Die Klammern um $s_1 \ldots s_n$ sind auch in den Fällen $n = 0$ und $n = 1$ nötig!

Informelle Datendefinitionen . . .

. . . sollen vor der formellen Definition erstellt werden!

Schokokekse

```
; Ein Schokokeks ist ein Wert
;  (make-chocolate-cookie x y)
; wobei x und y Zahlen sind, die den Schoko- bzw.
; den Keks-Anteil des Schokokekses darstellen.
```

Kartesische Koordinaten

```
; Ein Punkt im kartesischen Koordinatensystem
; in der Ebene ist ein Wert
;  (make-cartesian x y)
; wobei x und y Zahlen sind, die die X- bzw. die
; Y-Koordinate darstellen.
```

Konstruktionsanleitung 3 (Zusammengesetzte Daten)

Wenn bei der Datenanalyse zusammengesetzte Daten vorkommen, so muss zunächst ermittelt werden, zu welchen Sorten die Komponentendaten gehören.
Dann wird eine informelle Datendefinition erstellt:

```
; Ein t ist ein Wert
;  (c f1 ... fn)
; wobei ...
```

Dabei ist t der Name der zu definierenden Sorte, c der Name des Konstruktors und die f_i die Namen der Komponenten. Die anschließende Beschreibung muss für jede Komponente die Sorte sowie eine kurze Erläuterung der Komponente enthalten.

Konstruktionsanleitung 3 (Zusammengesetzte Daten) II

Daraus ergibt sich die Record-Definition

```
(define-record-procedures t
  c p
  (s1 ... sn))
```

in der noch Namen für die Selektoren s_i gewählt werden müssen. Für Namen des Konstruktors und des Sortenprädikat gelten folgende Konventionen:

```
(define-record-procedures t
  make-t t?
  (s1 ... sn))
```

Records konsumieren

Abstand vom Ursprung

```
; Abstand vom Ursprung bestimmen
(: distance-to-origin (cartesian -> real))
```

Gerüst dazu

```
(define distance-to-origin
  (lambda (xy)
    ...))
```

Konsumieren eines Records bedeutet, ihn in seine Komponenten zu zerlegen. Also müssen im Rumpf die Selektoren vorkommen ⇒ Schablone für den Rumpf:

```
(define distance-to-origin
  (lambda (xy)
    ...(cartesian-x xy) ... (cartesian-y xy) ...))
```

Ausfüllen der Ellipse

- Bekannt: Abstand eines Punktes vom Ursprung

$$d = \sqrt{x^2 + y^2}$$

- Bekannt: Prozedur zum Quadrieren

```
; Eine Zahl quadrieren
(: square (number -> number))
(define square
  (lambda (x)
    (* x x)))
```

- Vordefiniert: Signatur für die Quadratwurzel

```
; Quadratwurzel ziehen
; (: sqrt (number -> number))
```

Vollständige Prozedur

```
; Abstand vom Ursprung bestimmen
(: distance-to-origin (cartesian -> real))
(define distance-to-origin
  (lambda (xy)
    (sqrt (+ (square (cartesian-x xy))
             (square (cartesian-y xy))))))
```

- aus der Konstruktionsanleitung
- aus der Schablone
- aus Formel unter Verwendung bereits vorhandener Prozeduren

Konstruktionsanleitung 4 (Records als Argumente)

Sei x ein Prozedurargument von der Recordsorte t.

- Ermittle die Komponenten der Recordsorte t, von denen das Ergebnis abhängt.
- Im Rumpf der Prozedur muss für jede dieser Komponenten der Ausdruck $(s\ x)$ auftreten, wobei s der entsprechende Selektor von t ist.
- Vervollständige den Rumpf durch Konstruktion eines Ausdrucks, in dem diese Selektorausdrücke vorkommen.

Records produzieren

Beispiel: Verschieben eines Punktes

```
; Punkt verschieben
(: cartesian-move (cartesian real real -> cartesian))
```

Gerüst dazu

```
(define cartesian-move
  (lambda (c dx dy)
    ...))
```

Konstruktionsanleitung „Records als Argumente“ verwenden

```
(define cartesian-move
  (lambda (c dx dy)
    ...(cartesian-x c) ... (cartesian-y c) ...))
```

Punkt verschieben

```
(define cartesian-move
  (lambda (c dx dy)
    (make-cartesian (+ (cartesian-x c) dx)
                    (+ (cartesian-y c) dy))))
```

- Bestimme den Werte beider Komponenten
- Verwende den Konstruktor

Konstruktionsanleitung 5 (Records als Ausgabe)

Falls eine Prozedur als Ergebnis einen neuen Wert einer zusammengesetzten Sorte liefert, so muss in ihrem Rumpf der Konstruktor der zugehörigen Recordsorte auftreten.

4.2 Gemischte Daten

Creme-Marmelade-Kekse


```
; Ein Creme-Marmelade-Keks ist ein Wert
;  (make-cream-jelly-cookie x y z)
; wobei x, y und z Zahlen sind, die den Creme-,
; Marmeladen- bzw. Keks-Anteil darstellen.
```

Creme-Marmelade-Kekse als Record

Nach der Konstruktionsanleitung für zusammengesetzte Daten ergibt sich sofort die Recorddefinition:

```
(define-record-procedures cream-jelly-cookie
  make-cream-jelly-cookie cream-jelly-cookie?
  (cream-jelly-cookie-cream
   cream-jelly-cookie-jelly
   cream-jelly-cookie-cookie))
```

Gewicht eines Creme-Marmelade-Kekses

```
; Gewicht eines Creme-Marmelade-Kekses ermitteln
(: cream-jelly-cookie-weight
                    (cream-jelly-cookie -> real))
(define cream-jelly-cookie-weight
  (lambda (cjc)
    (+ (cream-jelly-cookie-cream cjc)
       (cream-jelly-cookie-jelly cjc)
       (cream-jelly-cookie-cookie cjc))))
```

Das geht ganz analog zu Schokokeksen, aber hier gibt es drei Bestandteile!

Gemischte Sorte `keks`

Ein Keks ist entweder ein Schokokeks oder ein Creme-Marmelade-Keks.
Signatur:

```
; Ein Keks ist eine der folgenden Alternativen
; - ein Schokokeks
; - ein Creme-Marmelade-Keks
; Name: cookie
(define cookie
  (signature
    (mixed chocolate-cookie cream-jelly-cookie)))
```

Dadurch wird eine neue Sorte `cookie` definiert. Ein Ausdruck hat die Sorte `cookie`, wenn er die Sorte `chocolate-cookie` oder die Sorte `cream-jelly-cookie` hat.

Prozeduren mit gemischter Sorte als Argument

Beispiel: Gewicht eines Kekses

```
; Gewicht eines Kekses ermitteln
(: cookie-weight (cookie -> real))
(define cookie-weight
  (lambda (c)
    ...))
```

Daten einer gemischten Sorte fallen „natürlich“ in Kategorien, die den Fällen in der Definition der Sorte entsprechen.
Déjà vu? Dies ähnelt sehr dem Fall einer Aufzählungssorte!
Daher verwenden wir Konstruktionsanleitung 2 in einer Spezialisierung, die fast identisch mit der Anleitung für eine Aufzählungssorte als Eingabe (Beispiel Briefporto) ist.
Wir bekommen eine Verzweigung. Jede Alternative in der Definition der gemischten Sorte liefert einen alternativen Fall im Rumpf von `cookie-weight`.

Erinnerung: Konstruktionsanleitung 2 für Aufzählungssorte als Eingabesorte

Wenn sich die Fallunterscheidung direkt aus der Sorte einer Eingabe a ergibt, da diese aus n Ausdrücken $e_1, \ldots, e_n$ besteht, kann man einen Konstruktionsschritt überspringen und gleich folgende Schablone verwenden:

```
(cond
 ((e1?) ...)
 ...
 ((en?) ...))
```

wobei jedes e_i? ein Ausdruck ist, der a auf Gleichheit mit e_i testet.

Konstruktionsanleitung 2 für gemischte Sorte als Eingabesorte

Nunmehr heißt es:
Wenn sich die Fallunterscheidung direkt aus der Sorte einer Eingabe a ergibt, da diese eine gemischte Sorte aus n Sorten ist, kann man einen Konstruktionsschritt überspringen und gleich folgende Schablone verwenden:

```
(cond ((p_1 a) ...)
      ...
      ((p_n a) ...))
```

wobei jedes p_i das Sortenprädikat für die $i$te Sorte der gemischten Sorte ist (und somit a darauf testet, ob es in der $i$ten Sorte ist).

Gemischte Sorte ⇒ Fallunterscheidung

```
; Gewicht eines Kekses ermitteln
(: cookie-weight (cookie -> real))
(define cookie-weight
  (lambda (c)
    (cond
     ((chocolate-cookie? c) ...)
     ((cream-jelly-cookie? c) ...))))
```

- In der Ellipse des ersten Falls ist klar, dass `c` ein `chocolate-cookie` ist.
- In der Ellipse des zweiten Falls ist klar, dass `c` ein `cream-jelly-cookie` ist.
- Also können dort die Prozeduren für die entsprechenden Sorten aufgerufen werden.

Gewicht eines Kekses: Lösung

```
; Gewicht eines Kekses ermitteln
(: cookie-weight (cookie -> real))
(define cookie-weight
  (lambda (c)
    (cond
     ((chocolate-cookie? c)
      (chocolate-cookie-weight c))
     ((cream-jelly-cookie? c)
      (cream-jelly-cookie-weight c)))))
```

Konstruktionsanleitung 6 (gemischte Daten)

Wenn bei der Datenanalyse gemischte Daten auftauchen:

- erstelle informelle Datendefinition der Form

```
; Ein x der Sorte s ist eine der folgenden
; Alternativen
; - ein x_1
;   ...
; - ein x_n
```

  Die x_i benennen die unterschiedlichen Sorten, die ein x sein kann.

- Definiere Signatur von s:

```
(define s
  (signature
    (mixed x_1 ... x_n)))
```

Für eine Prozedur, die gemischte Daten als Ein- oder Ausgabe hat, kann man dann Konstruktionsanleitung 2 verwenden.

Nochmals: Konstruktionsanleitung 2 für gemischte Daten als Eingabesorte

Wenn sich die Fallunterscheidung direkt aus der Sorte einer Eingabe a ergibt, da diese eine gemischte Sorte aus n Sorten ist, muss man folgende Schablone verwenden:

```
(cond ((p_1 a) ...)
      ...
      ((p_n a) ...))
```

wobei jedes p_i das Sortenprädikat für die $i$te Sorte der gemischten Sorte ist (und somit a darauf testet, ob es in der $i$ten Sorte ist).
Für die rechte Seite des $i$ten Zweigs muss man dann gemäß einer geeigneten Konstruktionsanleitung für Sorte x_i (für a) weitermachen.

Gemischte Sorten vs. Aufzählungssorten

Wir haben schon bemerkt: Aufzählungssorten und gemischte Daten ähneln sich sehr. Was ist überhaupt der Unterschied?

- Eine Aufzählungssorte besteht aus endlich vielen Literalen, die man zum Zweck der Definition aufzählt:

```
(define person
  (signature (one-of "Bob" "Philippa")))
```

- Eine gemischte Sorte ist die Vereinigung von (endlich vielen) Sorten, von denen jede evtl. auch unendlich viele Ausdrücke enthalten könnte. Z.B. gibt es unendlich viele Schokokekse und unendlich viele Creme-Marmelade-Kekse, und jeder davon ist ein Keks.
- Genau genommen sind Aufzählungssorten ein Spezialfall von gemischten Daten.

Erinnerung: Konstruktionsanleitung 2 für Aufzählungssorte als Ausgabesorte

Wenn sich die Fallunterscheidung direkt aus der Ausgabesorte ergibt, da diese aus n Ausdrücken $e_1, \ldots, e_n$ besteht, kann man einen Konstruktionsschritt überspringen und gleich folgende Schablone verwenden:

```
(cond
 (... e1)
 ...
 (... en))
```

Konstruktionsanleitung 2 für gemischte Sorte als Ausgabesorte

Nunmehr heißt es:

Wenn sich die Fallunterscheidung direkt aus der Ausgabesorte ergibt, da diese eine gemischte Sorte aus n Sorten ist, kann man einen Konstruktionsschritt überspringen und gleich folgende Schablone verwenden:

```
(cond
 (...   (c_1 ...))
 ...
 (...   (c_n ...)))
```

wobei jedes c_i der Konstruktor für die $i$te Sorte der gemischten Sorte ist.

Bemerkung: Schon bei Aufzählungssorten sagten wir, dass man eine Verzweigung eher von der Eingabe leiten lassen sollte. Das gilt für

Gemischte Daten als Ausgabesorte

Beispiel: Lieblingskeks

- Bobs Lieblingskeks ist ein Schokokeks
- Philippas Lieblingskeks ist ein Creme-Marmelade-Keks

```
; konstruiere Bobs oder Philippas Lieblingskeks
(: favorite-cookie (person -> cookie))
(check-expect (favorite-cookie "Bob")
              (make-chocolate-cookie 10 10))
(check-expect (favorite-cookie "Philippa")
              (make-cream-jelly-cookie 50 50 10))
(define favorite-cookie
  (lambda (person)
    ...))
```

Lieblingskekse

```
; konstruiere Bobs oder Philippas Lieblingskeks
(: favorite-cookie person -> cookie)
(check-expect (favorite-cookie "Bob")
              (make-chocolate-cookie 10 10))
(check-expect (favorite-cookie "Philippa")
              (make-cream-jelly-cookie 50 50 10))
(define favorite-cookie
  (lambda (person)
    (cond
     ((string=? person "Bob")
      (make-chocolate-cookie 10 10)
     ((string=? person "Philippa")
      (make-cream-jelly-cookie 50 50 10))))))
```

Welche Konstruktionsanleitung?

- Die Prozedur für Lieblingskekse hat die Signatur

  `(: favorite-cookie person -> cookie)`

  wobei `person` eine Aufzählungssorte und `cookie` eine gemischte Sorte ist. Man kann entweder Konstruktionsanleitung 2 aus dem vorherigen Kapitel (Aufzählungssorte als Eingabesorte) oder aber die eben angegebene Konstruktionsanleitung 2 für gemischte Daten als Ausgabesorte verwenden. Das Ergebnis ist das selbe.
- Wenn wir noch eine weitere Person `"Carla"` einführen, deren Lieblingskeks ein Creme-Marmelade-Keks mit 30g Creme, 30g Marmelade und 20g Keks ist, dann kann man immer noch beide Konstruktionsanleitungen anwenden, aber die entstehenden Programme sehen etwas anders aus (Übung!)

4.3 Zusammenfassung

Warum Produkt und Summe?

- Das Kreuzprodukt zweier Mengen ist ein Standardkonzept in der Mathematik. Beispiel: $K = \{\mathrm{B}, \mathrm{D}, \mathrm{K}, \mathrm{A}, 10, 9, 8, 7\}$, $F = \{\clubsuit, \spadesuit, \heartsuit, \diamondsuit\}$; dann ist $K \times F = \{(\mathrm{B}, \clubsuit), (\mathrm{B}, \spadesuit), \ldots, (7, \heartsuit), (7, \diamondsuit)\}$. Es gilt: $\#(K \times F) = \#K \cdot \#F$. Ein Record ist die Realisierung dieses Konzepts in einer Programmiersprache.
- Die Vereinigung zweier Mengen ist ebenfalls ein Standardkonzept in der Mathematik. Beispiel: $S = \{\clubsuit, \spadesuit\}$ und $R = \{\heartsuit, \diamondsuit\}$; dann ist $S \cup R = \{\clubsuit, \spadesuit, \heartsuit, \diamondsuit\}$. Sofern $S \cap R = \emptyset$, gilt: $\#(S \cup R) = \#S + \#R$. Eine gemischte Sorte ist die Realisierung dieses Konzepts in einer Programmiersprache und wird eben manchmal auch Summe genannt.

Zusammengesetzte und gemischte Daten

- Zusammengesetzte Daten (Records)
  - definieren,
  - konsumieren und
  - konstruieren
- Gemischte Daten
  - definieren,
  - konsumieren und
  - konstruieren