Tupel

Wechseln zu:Navigation, Suche

Dieser Artikel erfüllt die GlossarWiki-Qualitätsanforderungen **nur teilweise**:

Korrektheit: 3 (zu größeren Teilen überprüft)	**Umfang**: 4 (unwichtige Fakten fehlen)	**Quellenangaben**: 5 (vollständig vorhanden)	**Quellenarten**: 5 (ausgezeichnet)	**Konformität**: 5 (ausgezeichnet)

Das Tupel ist eines der drei wichtigesten Containerarten der Mathematik und Informatik:

Containerart	Ordnug der Elemente	Duplikate erlaubt
Menge	ungeordnet	nein
Multimenge	ungeordnet	ja
Tupel	i. Allg. geordnet	ja

Es gibt verschiedene Arten von *Tupeln* und verschiedene Repräsentationsformen. Entsprechend gibt es auch diverse Definitionen und Namen: *Liste*, *Familie*, *Folge*, *Komplex*, *Array*, *Hasharray*, *tuple*, *record*, *sequence*, *indexed family*, *property list*, *assoziation list*, *row*, *array*, *map*, *hash map* etc.

Ein Tupel ist ein Container (z. B. eine Menge, eine Klasse oder auch eine Kette von geordneten Paaren), der eine beliegibe Anzahl von Schlüssel/Wert-Paaren enthält. Dabei darf kein Schlüssel doppelt vorkommen. Die Klasse aller Schlüssel heißt Indexbereich.

Die wichtigsten Tupelarten sind:

Name	Definition	Beispiel	Indexbereich
Liste	Kette von Paaren	[math](2, (3, (5, (7, \emptyset))))[/math] Abkürzung; [math](2, 3, 5, 7)[/math]	[math]\{1, 2, 3, 4\}[/math] oder [math]\{0, 1, 2, 3\}[/math]
Familie	Menge von Paaren	[math]\{(a, 2), (b, 3), (c, 5), (d, 7)\}[/math]	[math]\{a, b, c, d\}[/math]
Assoziationsliste	Liste/Tupel von Paaren	[math]((a, 2), (b, 3), (c, 5), (d, 7))[/math]	[math]\{a, b, c, d\}[/math] *und* [math]\{1, 2, 3, 4\}[/math]

Die Listendefinition hat den Nachteil, dass der Indexbereich stets endlich ist. Üblicherweise handelt es sich dabei um ein Intervall von natürlichen Zahlen: [math][0, n-1][/math] oder [math][1, n][/math]. Der Indexbereich einer Familie kann dagegen beliebig viele Elemente enthalten. Es kann sich dabei auch um |[Komprehension|exklusive Container]] (echte Klassen, Unmengen) handeln, die so viele Elemente enthalten, dass deren Mächtigkeit gar nicht definiert ist.

Die Familendefinition hat dagegen den Nachteil, dass die Schlüssel/Wert-Paare Elemente eines Containers (z. B. einer Klasse) sind. Ein Container kann aber nur Individuen (einschließlich Individuen-Container wie Mengen) als Elemente enthalten, aber keine exklusiven Container. Das heißt, in einer Theorie, in der es exklusive Container gibt (wie {{zB]] in einer Klassentheorie), gibt es Objekte, die nicht in Familien gespeichert werden können.

Auf der anderen Seite bestehen Listen i. Allg. nur aus Paaren. Und Paare können so rafiniert definiert werden, dass nicht nur Individuen, sondern auch exklusive Container darin enthalten sein können.[1][2] Das heißt, Listen können ebenfalls exklusive Container enthalten.

Name	Vorteil	Nachteil

Liste Assotiationsliste	**Alle** Individuen und Container als Tupel-Elemente möglich	Individuenbereich endlich
Familie	Individuenbereich beliebig	Exklusiver Container **nicht** als Tupel-Elemente möglich

Aus Sicht des (praktischen nicht des theoretischen) Informatikers ist diese Tabelle weniger interessant, da er sowieso nur mit endlichen Mengen arbeitet: Der Speicher ist im Gegensatz zum Speicher der Turingmaschine begrenzt. Für ihn ist vielmehr interessant, dass im endlichen Fall jede Tupelart durch jede andere ersetzt werden kann, da die verschiedenen Darstellungen (mehr oder minder gut) bijektiv aufeinander abgebildet werden können. Er ist wichtig, die Komplexität der CRUD-Operationen für die einzelnen Darstellungen zu kennen, um für den jeweiligen Anwendungsfall die beste Darstellungsart wählen zu können.

Inhaltsverzeichnis

1 Anschauliche Definition (Kowarschick)

Ein **Tupel** ordnet jedem Element einer Klasse (oder Menge) von **Schlüsseln** oder **Attributnamen** jeweilse einen **Wert** zu. Ein Tupel ist also eine Klasse von unterschiedlich benannten *Attributen*, d. h. eine Klasse von **Schlüssel/Wert-Paaren**, wobei jeder Schlüssel nur einmal vorkommen darf. Unterschiedlichen Schlüsseln kann jedoch durchaus derselbe Wert zugeordnet werden. Das heißt, ein Wert kann durchaus mehrfach vorkommen. Wert-Duplikate sind also erlaubt.

Die Klasse aller Schlüssel heißt **Indexbereich** oder auch – sofern es sich um eine Menge handelt – **Indexmenge**. Wenn auf dem Indexbereich eine totaale Ordnung definiert ist, sind die Schlüssel/Wert-Paare des Tupels ebenfalls (bezüglich dieser Ordnungsrelation) geordnet. Unabhängig davon, ob eine Ordnung existiert oder nicht, sind die Elemente auf jeden Fall bezüglich des Indexbereichs indexiert.

Begriff	Alternativnamen	englische Bezeichnungen
Tupel	Liste, Familie, Folge, Komplex Array, Assoziationsliste, Hasharray etc.	tuple, record, sequence, indexed family, property list, assoziation list, row, array, map, hash map
Indexbereich	Definitionsbereich, Indexmenge	domain, key set
Wertebereich	Wertemenge	value set
Attribut	Schlüssel/Wert-Paar	attribute, property, key/value pair
Attributname	Schlüssel, Index	key, index, attribute/property name, attribute/property key
Attributwert	Wert, Glied	value, attribute/property value

Attribute können auch mehr als einen Attributnamen haben. Diese werden dann mit Schlüssel-Schlüssel-Wert-Tripeln etc. dargestellt.

1.1 Indexbereich und Indexmenge

Die Klasse aller Schlüssel eines Tupels wird **Indexbereich** des Tupels genannt. Wenn es sich beim Indexbereich um eine Menge handelt, kann man auch **Indexmenge** sagen.

1.2 Tupellänge

Die **Länge** eines Tupel ist gleich der Mächtigkeit des zugehörigen Indexbereichs.

Ein Tupel der Länge [math]l[/math] wird auch [math]l[/math]-**Tupel** genannt.

1.3 Gleichheit zweier Tupel

Zwei Tupel sind genau dann gleich, wenn die zugehörigen Indexbereiche gleich sind und wenn die Werte jeweils gleich benannter Elemente ebenfalls gleich sind.

2 Formale Definition (Kowarschick)

Siehe Tupel: Formale Definition.

3 Anmerkungen

Tupel sind im Prinzip nichts anderes als Funktionen, deren Definitionsbereich *Indexbereich* genannt wird. Eine Funktion ordnet jedem Element des Definitionsbereichs einen Wert zu, entsprechend ordnet ein Tupel jedem Schlüssel, also jedem Element des Indexbereichs einen Wert zu.

Tupel können auch als *geordnete Multimengen*, d.h. als *Listen* aufgefasst werden, sofern für den Indexbereich eine Ordnung definiert ist:

Werte können mehrfach vorkommen (im Gegensatz zu normalen Mengen, aber in Einklang mit Multimengen).
Die Werte sind (gemäß der auf den Schlüsseln definierten Ordnung) angeordnet (im Gegensatz zu Mengen und Multimengen).

Üblicherweise spielt in der Informatik die Ordnung der Tupelelemente nur dann eine Rolle , wenn als Indexbereich eine Menge von aufeinanderfolgenden *natürlichen Zahlen*verwendet wird.

4 Beispiele

Tupelart	**Tupelschema**	**Beispiel**
Liste/Positionstupel		
Attributnotation	`{1: String, 2: {'f','m','x'}}`	`{1: 'Anton', 2: 'm'}`
Listennotation	`(String, {'f','m','x'})`	`('Anton', 'm')`
math. Notation		[math](\text{Anton}, \text{m})[/math]
Familie/Attributtupel		
Attributnotation	`{name: String, sex: {'f','m','x'}}`	`{name: 'Anton', sex: 'm'}`
math. Notation		[math]\{(name,\text{Anton}), (sex,\text{m})\}[/math]
Assoziationsliste/ Positionsattributtupel		
Attributnotation	`{1/name: String, 2/sex: {'f','m','x'} }`	`{1/name: 'Anton', 2/sex: 'm' }`
Listennotation	`(name: String, sex: {'f','m','x'})`	`(name: 'Anton', sex:'m')`
math. Notation		[math]((name,\text{Anton}), (sex,\text{m}))[/math]

Positionsattributtupel sind Tupel, deren Attribute zwei unterschiedliche Attributnamen haben: Eine Zeichenkette und einen Positionsbezeichner. Dies ist die ideale Darstellung für Tupel, die durch Tabellenzeilen repräsentiert werden:

name	**sex**	**spouse**
`'Anton'`	`'m'`	`'Berta'`
`'Berta'`	`'f'`	`'Anton'`
`'Cäsar'`	`'m'`	`null`

Jedes Attribut dieser drei Tupel hat sowohl einen Namen, der über der jeweiligen Spalte notiert wird, sowie eine Position, die sich durch die Spaltenposition ergibt.

In den folgenden Beispielen werden Attribute teils mittels Attributnotation `a:v`, `i/a:v` und teils mittels Listennotation [math](a,v)[/math], [math](i,a,v)[/math] dargestellt.

4.1 Attributnotation

Im Fall von endlichen Indexbereichen kann ein Tupel einfach durch die explizite Angabe von Schlüssel/Wert-Paaren erfolgen.

```
t1 = {name: 'Anton', geburtsjahr: 1961, ehefrau: 'Berta'}
t2 = {ehefrau: 'Berta', geburtsjahr: 1961, name: 'Anton'}
t3 = {name: 'Anton', geburtsjahr: 1961, hochschule: 'HSA'}
t4 = {name: 'Anton', geburtsjahr: 1962, hochschule: 'HSA'}
t5 = {name: 'Anton', geburtsjahr: 1961, ehefrau: 'Berta', hochschule:
'HSA'}
```

Nur Tupel `t1` und `t2` sind gleich, alle anderen Tupel unterscheiden sich. Entweder unterscheiden sich die Indexbereiche (`t1` bis `t4` haben die Länge 3, Tupel `t5` hat dagegen die Länge 4) oder es stimmen nicht alle gleich benannten Elemente überein (alle übrigen Tupelpaare).

Folgendes ist kein Tupel (und damit auch keine Funktion), da zwei Elemente gleich benannt sind:

```
{name: 'Anton', name: 'Cäsar', hochschule: 'HSA'}
```

4.1.1 Attributnotation in der Informatik

Die Attributnotation kommt in der Informatik häufig zum Einsatz. Beispielsweise können in JSON innerhalb einer Mengenklammer beliebig viele (jedoch nur endlich viele) Attribute angeben werden. Als Attributnamen werden Zeichenketten (Strings) verwendet, die mit Anführungszeichen " markiert sind. Der zugehörige Attributwert wird vom Attributnamen durch einen Doppelpunkt abgetrennt.

```
{"name": "Anton", "geburtsjahr": 1961, "ehefrau": "Berta"}
...
```

Weitere Beispiele für Datenstrukturen, in denen Mengen von Schlüssel/Wert-Paaren zum Einsatz kommen:

C/C++: Datentyp `struct` (die Länge kann zur Laufzeit *nicht* verändert werden)
Pascal: Datentyp `record` (die Länge kann zur Laufzeit *nicht* verändert werden)
diverse Sprachen: Hashtabelle (auch *hash map*, *hash array*, *assoziatives Array* etc.; die Länge kann zur Laufzeit verändert werden)
SQL: Tabellenzeilen, d.h. Elemente von Relationen (die Länge kann zur Laufzeit durch Schema-Evolution verändert werden)
JavaScript: Objekte (die Länge kann zur Laufzeit verändert werden, sofern dies nicht explizit mittels `Object.freeze` „untersagt“ wird)
Java und viele andere Sprachen: Objekte (die Länge kann zur Laufzeit i. Allg. *nicht* verändert werden)

4.1.2 Attributzugriff

Um mittels des Attributnamens auf einen Attributwert zuzugreifen, haben sich in der Informatik zwei syntaktische Konstrukte etabliert (obwohl es durchaus noch Sprachen gibt, die andere syntaktische

Konstrukte verwenden):

Die Indexnotation: `t1["name"]`, `t5["ehefrau"]` etc.
Die Punktnotation: `t1.name`, `t5.ehefrau` etc.

In der Mathematik sind dagegen folgende Konstrukte üblich:

Die Funktionsauswertung: [math]\rm{t1}(\rm{name})[/math], [math]\rm{t5}(\rm{ehefrau})[/math] etc.
Die Projektionsfunktion [math]π[/math]: [math]π_{\rm{name}}(\rm{t1})[/math], [math]π_{\rm{ehefrau}}(\rm{t5})[/math] etc.
Die Indexnotation: [math]\rm{t1}_{\rm{name}}[/math], [math]\rm{t5}_{\rm{ehefrau}}[/math] etc.

4.2 Listennotation

Für Tupel, deren Indexbereich eine Menge von [math]n[/math] aufeinanderfolgenden natürlichen Zahlen ist (i. Allg. [math]\{i| 0 \lt i \le n\}[/math], oder, wie in vielen Programmiersprachen üblich, [math]\{i| 0 \le i \lt n\}[/math]), bietet sich die in der Mathematik gebräuchliche Listennotation an. Bei dieser werden die Schlüssel nicht explizit angegeben, sondern implizit durch die Position der Elemente festgelegt:

[math]t_6 := (555, 333)[/math]

[math]t_7 := (333, 555)[/math]

[math]t_8 := (555, 333, 555)[/math]

[math]t_9 := (0, 1, 1, 2, 3, 5, 8, \ldots)[/math] (Fibonacci-Zahlen)

Die Tupel [math]t_6[/math] bis [math]t_9[/math] unterscheiden sich alle voneinander. In Tupel [math]t_6[/math] steht an Position 1 das Element [math]555[/math], während in Tupel [math]t_7[/math] an Position 1 das Element [math]333[/math] steht. Tupel [math]t_8[/math] unterscheidet sich von Tupel [math]t_6[/math] und [math]t_7[/math], da die Indexbereiche ([math]\{1,2\}[/math] bei Tupel [math]t_6[/math] und [math]t_7[/math]; [math]\{1,2,3\}[/math] bei Tupel [math]t_8[/math]) nicht übereinstimmen.

Tupel [math]t_6[/math] und [math]t_7[/math] sind 2-Tupel, Tupel [math]t_8[/math] ist um ein Element länger, die Länge von Tupel [math]t_9[/math] beträgt ω. Tupel [math]t_9[/math] enthält also abzählbar unendlich Elemente. Da ein unendlich großes Tupel nicht mehr explizit angegeben werden kann, muss die Definition entweder anschaulich erfolgen (siehe obige Definition von Tupel [math]t_9[/math]) oder mittels einer Rechenvorschrift:

[math] \rm{fib(n)} := \begin{cases} 0 & n = 0\\ 1 & n = 1\\ \rm{fib}(n-1) + \rm{fib}(n-2) & \text{otherwise} \end{cases}\\ t_9 := (\rm{fib}(0), \rm{fib}(1), \rm{fib}(2), \rm{fib}(3), \ldots)\\ t_9(i) := \rm{fib}(i) [/math]

4.2.1 Listennotation in der Informatik

In der Informatik, wie beispielsweise in JSON, kommt die Listennotation ebenfalls zu Einsatz:

```
t6 = [555, 333]
t7 = [333, 555]
t8 = [555, 333, 555]
```

Unendliche lange Tupel können in JSON nicht definiert werden. Allerdings ist es in JavaScript möglich, mittels Generatoren beliebig große Tupel zu generieren. Der Generator selbst kann als unendlich langes Tupel aufgefasst werden:

```
// Generator für Fibonacci-Zahlen (BigInt).
function* fib()
{ let a = 0n, b = 1n; // 0n = BigInt(0), 1n = BigInt(1), ...
  yield a;
  yield b;
  while (true)
  { [a, b] = [b, a+b];
    yield b;
  }
}

// Hilfsfunktion zum Materialisieren der
// ersten n Elemente eines Generators.
function take(n, p_iterable)
{ let tuple = [], i = 0;
  while (i++ < n)
  { tuple.push(p_iterable.next().value);  }
  return tuple;
}

let f = fib();
console.log(f.next().value, f.next().value, f.next().value,
f.next().value, f.next().value);
// -> 0n, 1n, 1n, 2n, 3n
console.log(take(1000, fib()));
// -> [0n, 1n, 1n, 2n, 3n, 5n, 8n, 13n, 21n ...] // tausend Elemente
```

In der Informatik finden sich weitere Datenstrukturen, in denen Elemente sequenziell abgespeichert werden, d.h., bei denen Tupel nicht als Menge von Attributen, sondern als Folge von Elementen aufgefasst werden:

Arrays oder Felder (häufig mit fixer Länge)
Listen in zahlreichen Ausprägungen (i. Allg. mit variabler Länge)
Streams (evtl. sogar unendlich lang)

4.2.2 Attributzugriff

Um mittels der Attributposition auf einen Attributwert zuzugreifen, hat sich in der Informatik ein syntaktisches Konstrukt etabliert:

Die Indexnotation: `t[1]`, `t[2]` etc.

Bei verketteten Listen, Streams etc. ist der Zugriff auf ein Element an einer bestimmten Position dagegen manchmal nur dadurch möglich, dass man sich mittels einer Funktion oder Methode (z. B. `next`) von einem Element zu nächsten „hangelt“ (Traversierung), bis man beim gewünschten Element angekommen ist (siehe z. B. die Funktion `take` im vorangegangenen Abschnitt).

In der Mathematik sind dagegen folgende Konstrukte üblich:

Die Funktionsauswertung: [math]t(1)[/math], [math]t(2)[/math] etc.
Die Indexnotation: [math]t_1[/math], [math]t_2[/math] etc.
Die Projektionsfunktion [math]π[/math]: [math]π_1(t)[/math], [math]π_2(t)[/math] etc.

4.2.3 Spezielle Tupel mit Positionsattributen

Länge [math]n[/math]	Name	Attributnotation	Listennotation
0	Leeres Tupel	[math]\{\}[/math]	[math]()[/math]
1	Singel	[math]\{(1,5)\}[/math]	[math](5)[/math]
2	(geordnetes) Paar	[math]\{(1,5), (2,3)\}[/math]	[math](5,3)[/math]
3	Triple	[math]\{(1,5), (2,3), (3,8)\}[/math]	[math](5,3,8)[/math]
4	Quadrupel	[math]\{(1,5), (2,3), (3,8), (4,π)\}[/math]	[math](5,3,8,π)[/math]
5	Quintupel	[math]\vdots[/math]	[math]\vdots[/math]
6	Sextupel		
7	Septupel		
8	Oktupel		
[math]\vdots[/math]	[math]\vdots[/math]		
100	Centupel		
n	n-Tupel		
ω oder [math]\aleph_0[/math]	ω- oder oder [math]\aleph_0[/math]-Tupel	[math]\{(i,i^2)| i \in \mathbb N\}[/math]	[math](i^2)_{i \in \mathbb N}[/math]
[math]2^{\aleph_0}[/math]	[math]2^{\aleph_0}[/math]-Tupel	[math]\{(r,r^2)| i \in \mathbb R\}[/math]	[math](r^2)_{r \in \mathbb R}[/math]

Jede endliche oder abzählbar unendliche Folge, Sequenz oder Familie kann als Tupel in Listennotation aufgefasst werden. Jede (mathematische) Funktion kann als Tupel aufgefasst werden.

5 Geschichte

5.1 Definition „Tupel“ (Bourbaki (1939)[3])

Die Mathematiker der Gruppe Bourbaki definieren zunächst das geordnete Paar (**couple**) nach Kazimierez Kuratowski und erwähnen, dass es das Paaraxiom erfüllt (S. E II.7):

> *On dit que le terme [math]\{\{x\},\{x,y\}\}[/math] est le **couple formé de [math]x[/math] et de [math]y[/math]**, et on le note de façon abrégée [math](x, y)[/math], de sorte que la relation [math](x, y) = (x', y')[/math] est equivalente a «[math]x = x'[/math] et [math]y = y'[/math]».*

Im Anschluss daran definieren sie das **geordnete Tripel** (**triplet**) mit Hilfe zweier Paare (S. E II.9):

> *... un élément [math]((x, y), z)[/math] de [math]A \times B \times C[/math] s'écrit aussi [math](x, y, z)[/math] et s'appelle un triplet.*

Sie weisen außerdem daraufhin, dass sich dies für vier und mehr Elemente verallgemeinern lässt.

Auf Seite E IV.5 führt Bourbaki den Begriff **Tupel** (**multiplet**) für den Term [math](s_1,\ldots,s_p)[/math] ein und verweisen dabei explizit auf die obige Definition.

Zusammenfassung: Der Begriff **Tupel** (**multiplet**) wird von Bourbaki folgendermaßen definiert:

> *[math](x,y) := \{\{x\},\{x,y\}\}[/math]*
> *Wenn [math]n \ge 3[/math], dann [math](x_1,\ldots,x_n) := ((x_1,\ldots,x_{n-1}), x_n)[/math]*

5.2 Definition „Familie“ (Bourbaki (1939)[4])

Die Mathematikergruppe Bourbaki definiert zunächst **funktionelle Relationen** (**Relations fonctionelles**) als *Relationen* [math]R[/math], die rechtseindeutig sind (S. E I.40):

> *[math](\forall y)(\forall z)(((y|x)R \,\text{et}\, (z|x)R) \Rightarrow (y = z))[/math]*

Im Anschluss an die Paardefinition (siehe vorangegangenen Abschnitt) zeigt das Mathematikerteam, dass jede Relation eindeutig durch eine Menge von Paaren repräsentiert wird. Diese Menge bezeichnen sie als **Graph** (**graphe**) der Relation:

> *[math](\exists G)(G \,\text{est un graphe et}\, (\forall x)(\forall y)(R \Leftrightarrow ((x, y) \in G)))[/math] Le graphe [math]G[/math] est alors unique en vertu de l'axiome d'extensionalite, et s'appelle le **graphe de [math]R[/math]***

Anschließend definieren sie die eigenlichen **Funktionen** (**fonctions**) als *funktionelle Relationen* mit Definitionsbereich [math]A[/math] und Wertebereiche [math]B[/math] (S. E II.13):

> *Autrement dit, une correspondance [math]f = (F, A, B)[/math] est une fonction si, pour taut [math]x[/math] appartenant à l' ensenzble de départ [math]A[/math] de [math]f[/math], la relation [math](x,y) \in F[/math] est fonctionnelle en [math]y[/math] (I, p. 41);*

[math]F[/math] ist als Graf der Funktion eine Menge von Paaren. Dies ist die typische mengentheoretische Definition des Funktionsbegriffs.

Zu guter Letzt führt die Gruppe den Begriff **Familie** als Alternative für den Begriff **Funktion** ein (S. E II. 16):

> *Si [math]f[/math] est une application de [math]A[/math] dans [math]B[/math], la fonction [math]f[/math] est égale à la fonction [math]x \rightarrow f(x) (x \in A, f(x) \in B)[/math], qu'on écrit sirnplement [math]x \rightarrow f(x)[/math], ou aussi [math](f_x)_{x \in A}[/math] c'est surtout quand on utilise la dernière notation qu'on parle de « famille d'éléments » au lieu de « fonction ».*

Die Begriffe *Familie* und *Funktion* unterscheiden sich nur syntaktisch. Die Syntax [math](f_x)_{x \in A}[/math] wurde in Anlehnung an die Tupelsyntax gewählt und betont damit die Verwandschaft zwischen diesen beiden Begriffen. Der Begriff *Familie* kann also als Alternative für den Begriff *Tupel* aufgefasst werden. Diese Definition hat den Vorteil hat, dass auch leere, einelementige und nicht-endliche Indexbereiche unterstützt werden.

5.3 Definition „Tupel“ (Gödel (1940)[5])

> *Dfn [math]\lt xy\gt = \{\{x\}\{xy\}\}[/math]*
> *Dfn [math]\lt x_1,\ldots,x_n\gt = \lt x_1\lt x_2\ldots x_{n}\gt \gt [/math] [Anm. WK: wobei [math]n \ge 3[/math]]*
> *Dfn [math]\lt x\gt = x[/math]*

Wegen der Rechtsassoziativität gilt folgender Satz (wie man laut Gödel per Induktion nachweist):

[math]\lt x_1,\ldots,x_n\lt x_{n+1}\ldots x_{n+p}\gt \gt = \lt x_1\ldots x_{n}x_{n+1}\ldots x_{n+p}\gt \gt [/math]

Anmerkung:
Diese Definition unterschiedet sich nicht wesentlich von der oben angeführten Tupel-Definition der Mathematikergruppe Bourbaki: Die Klammerung ist rechts- an Stelle von linksassoziativ und das einelementige Tupel wurde als Spezialfall ergänzt.

5.4 Definition „List“ (McCarthy (1960)[6])

S-expressions are then defines as follows:

Atomic symbols are S-expressions.
If [math]e_1[/math] and [math]e_2[/math] are S-expressions, so is [math](e_1 \cdot e_2)[/math]-

...

*An S-expression is then simply an ordered pair, the terms of which may be atomic symbols or simpler S-expressions. We can represent a list of arbitrary length in terms of S-expressions as follows. The li**st***

[math](m_1, m_2, \cdots, m_n)[/math]

is represented by the S-expresion

[math](m_1 \cdot (m_2 \cdot (\cdots (m_n \cdot \text{NIL}) \cdots)))[/math]

Here [math]\text{NIL}[/math] is an atomic symbol used to terminate lists.

Anmerkung:
McCarthy erwähnt, dass es auch einelementige Listen gibt: [math](m) = (m \cdot \text{NIL})[/math]

5.5 Definition „List“ (McCarthy et al. (1965)[7])

Seite 2:

An S-expression is either an atomic symbol or it is composed of these elements in the following order: a left parenthesis, an S-expression, a dot, an S-expression, and a right parenthesis.

Notice that this definition is recursive.

Seite 4:

Any S-expression can be expressed in terms of the dot notation. However, LISP has an alternative form of S-expression called the list notation. The list (m_1 m_2 . . . m□) can be defined in terms of dot notation. It is identical to (m_1 . (m_2 . (. . . . (m□ . NIL). . .))).

The atomic symbol NIL serves as a terminator for lists. The null list ( ) is identical to NIL.

5.6 Definition „Property List“ (McCarthy et al.

(1965)[8])

Seite 39:

In LISP 1.5 wird eine (interne) Datenstruktur **property list** definiert, die dazu verwendet wird, atomaren Symbolen eine ganze Liste von **Eigenschaften** (**properties**) zuzuordnen. Dabei wird jede *Eigenschaft* (*property*) von einem atomaren Symbol eingeleitet, welches **Indikator** (*indicator*) genannt wird.

Seite 58:

In LISP 1.5 gibt es eine Pseudofunktion „define“, um atomaren Symbolen Properties zuzuordnen:

> *The argument of define, x, is a list of pairs*
>
> *((u₁ v₁) (u₂ v₂) ... (u□ v□))*
>
> *where each u is a name and each v is a λ-expression for a function. For each pair, define puts an EXPR on the property list for u pointing to v. The function of define puts things on at the front of the property list.*

5.7 Common Lisp

5.7.1 Definition „Association List“ (Steele (1990), S431[9])

*An **association list**, or **a-list**, is a data structure used very frequently in Lisp. An a-list is a list of pairs (conses); each pair is an association. The **car** of a pair is called the **key**, and the **cdr** is called the **datum**.*

Übersetzung (W. Kowarschick):
*Eine **Assoziationsliste** oder **A-Liste** ist eine Datenstruktur, die in Lisp sehr oft benutzt wird. Eine A-Liste ist eine Liste von Paaren (Cons-Zellen); jedes Paar ist eine Assoziation. Das car-Element [WK: das erste Element] eines Paare wird **Schlüssel** genannt und das cdr-Element [WK: das zweite Element] wird **Datum** genannt.*

5.7.2 Definition „Property List“ (Steele (1990), S. 238 – 239[10])

*Since its inception, Lisp has associated with each symbol a kind of tabular data structure called a **property list** (**plist** for short). A property list contains zero or more entries; each entry associates with a key (called the **indicator**), which is typically a symbol, an arbitrary Lisp object (called the **value** or, sometimes, the **property**). There are no duplications among the indicators; a property list may only have one property at a time with a given name. In this way, given a symbol and an indicator (another symbol), an associated value can be retrieved.*

A property list is very similar in purpose to an association list. The difference is that a property list is an object with a unique identity; the operations for adding and removing property-list entries are destructive operations that alter the property list rather than making a new one. Association lists, on the other hand, are normally augmented non-destructively (without side effects) by adding new entries to the front (see `acons` and `pairlis`).

Übersetzung (W. Kowarschick):
*Von Anfang an wurde in Lisp jedes Symbol mit einer Art tabellenartiger Datenstruktur verknüpft, die **Propertyliste** [WK: **Eigenschaftsliste**] genannt wird (kurz **P-Liste**). Eine Propertyliste enthält null oder mehr Einträge: Jeder Eintrag verknüpft einen Schlüssel (**Indikator** genannt), welcher typischerweise ein Symbol ist, mit einem beliebigen Lisp-Objekt (**Wert** genannt oder manchmal auch **Property** [WK: **Eigenschaft**]). Unter den Indikatoren gibt es keine Duplikate; eine Propertyliste kann zu jedem Zeitpunkt nur eine Eigenschaft mit einem gegebenen Namen haben. Auf diese Art kann für ein gegebenes Symbol und einen gegebenen Indikator (ein anderes Symbol) auf einen [WK: mit dem Symbol] verknüpften Wert zugegriffen werden.*

Der Zweck einer Propertyliste ist dem einer Assoziationsliste sehr ähnlich. Der Unterschied ist, dass eine Propertyliste ein Objekt mit einer eindeutigen Identität ist; die Operationen zum Hinzufügen und Löschen von Propertylisten-Einträgen sind destruktive Operationen, die die Propertyliste verändern anstatt eine neue zu erstellen. Assoziationslisten werden demgegenüber nicht-destruktiv erweitert (ohne Seiteneffekte), indem neue Einträge vorne hinzugefügt werden (vgl. `acons` und `pairlis`).

5.8 Definition „Tupel“ (Schmidt (1966)[1])

Schmidt definiert Tripel, Quadruppel ..., [math]n[/math]-Tupel genauso wie Bourbaki (siehe oben). Allerdings verwendet er eine eigene Definition des geordneten Paares, die auch echte Klassen zulässt:

> *[math](x,y) := \{\,\{\{x\}\}: x \in a\,\} \,\cup\, \{\,\{\emptyset, \{x\}\}: x \in b\,\}[/math]*
> *Wenn [math]n \ge 3[/math], dann [math](x_1,\ldots,x_n) := ((x_1,\ldots,x_{n-1}), x_n)[/math]*

Anmerkung:
Damit ist es z. B. möglich, das Monoid [math](\Omega, +, *)[/math] der Ordinalzahlen als Tupel zu definieren (die Klasse [math]\Omega[/math] ist eine Unmege).

5.9 Definition „Familie“ (Schmidt (1966)[11])

Schmidt definiert Funktionen [math]f[/math] als Relation (d. h. als Teilklasse von [math]\mathcal{V} \times \mathcal{V}[/math]), die folgende Eindeutigkeitsbedingung erfüllt (S. 119):

> *[math]\bigwedge x \bigwedge y \bigwedge z ((x,y) \in f \wedge (x,z) \in f \Rightarrow y=z)[/math]*

Den Wert der Funktion [math]f[/math] an der Stelle [math]x[/math] berechnet er folgendermaßen:

> *[math]f(x) := \bigcap\{y|(x,y) \in f\}[/math]*

Das heißt, eine Funktion ist eine Klasse von geordneten Paaren (wobei die Paare selbst nur Mengen als Elemente enthalten können), bei denen kein Element des Definitionsbereichs doppelt vorkommt.

Er führt dann (S. 122) den Begriff **Familie** oder **Folge** als Synonym für den Begriff *Funktion* ein. Den *Definitionsbereich* nennt er in diesem Fall **Indexbereich**. Ein Element eines Indexbereichs heißt **Index**. An Stelle der Schreibweise [math]f(x)[/math] verwendet er die Indexschreibweise [math]f_x[/math].

Anmerkungen:
Im Prinzip unterscheidet sich diese Definition nicht wesenlich von der Definition von Bourbaki. Schmidt unterscheidet allerdings nicht zwischen Funktion und Funktionsgraph. Außerdeist sein Funktionsbegriff in der Hinsicht allgemeiner, dass eine Funktion nicht notwendigerweise eine Menge von Paaren ist, sondern auch eine echte Klasse von Paaren sein kann.

Auf der einen Seite ist der Begriff Schmidt-Familie universeller als der Begriff Schmidt-Tupel, da der Indexbereich auch leer, einelementig und beliebig groß sein kann. Auf der anderen Seite ist der Begriff Schmidt-Tupel universeller als der Begriff Schmidt-Familie, da ein Tupel auch echte Klassen als Elemente enthalten kann.

5.10 Definition „Tupel“ (Ebbinghaus (2003)[12])

Ebbinghaus definiert zunächst das geordnete Paar

[math](x,y) := \{\{x\},\{x,y\}\}[/math]

nach Kazimierez Kuratowski und verallgemeinert es dann für beliebige [math]n[/math]-Tupel ([math]n \ge 1[/math]):

(i) [math](x) := x[/math]
(ii) Für [math]n \ge 1[/math] sei [math](x_0,\ldots,x_n) := ((x_0,\ldots,x_{n-1}), x_n)[/math]

Er formuliert den folgenden Satz:

[math](x_0,\ldots,x_n) = (y_0,\ldots,y_n) \leftrightarrow x_0 = y_0 \wedge \ldots \wedge x_n = y_n[/math]

Einen Beweis gibt er nicht an, da sich dieser leicht *durch metaprachliche Induktion* ergäbe.

Anmerkung:
Diese Definition unterschiedet sich nicht von der Definition der Mathematikergruppe Bourbaki (siehe oben). Es wurde lediglich das einelementige Tupel als Spezialfall ergänzt, in derselben Weise, wie dies Gödel vorgeschlagen hat (siehe oben).

5.11 Definition „Familie“ (Ebbinghaus (2003)[13])

TO BE DONE

6 Quellen

Schmidt (1966): Jürgen Schmidt; Mengenlehre – Grundbegriffe; Reihe: B.I.Hochschultaschenbücher; Band: 1; Nummer: 56; Verlag: Bibliographisches Institut AG; Adresse: Mannheim; ISBN: B0000BUJC6; 1966; Quellengüte: 5 (Buch), S. 100
Glubrecht, Oberschelp, Todt (1983): Jürgen-Michael Glubrecht, Arnold Oberschelp und Günter Todt; Klassenlogik; Verlag: Bibliographisches Institut; Adresse: Mannheim, Wien, Zürich; ISBN: 3-411-01634-5, 978-3411016341; 1983; Quellengüte: 5 (Buch)
Bourbaki (1939): Nicolas Bourbaki; Théorie des ensembles; Verlag: Hermann; Adresse: Paris; 1939; Quellengüte: 5 (Buch), S. E II.7, E II.9, E IV.5
Bourbaki (1939), S. E I.40, S. E II.13, S. E II.16
Gödel (1940): Kurt Gödel; The Consistency of the Continuum Hypothesis; Verlag: Princeton University Press; ISBN: 0-691-07927-7; Web-Link; 1940; Quellengüte: 5 (Buch), S. 4
McCarthy (1960): John McCarthy; Recursive Functions of Symbolic Expressions and Their

Computation by Machine, Part I; in: Communications of the ACM; Band: 3; Nummer: 4; Seite(n): 184-195; Verlag: Association for Computing Machinery; Adresse: New York; Web-Link 0, Web-Link 1; 1960; Quellengüte: 5 (Artikel), S. 187
McCarthy et. al. (1965): John McCarthy, Paul W. Abrahams, Daniel J. Edwards, Timothy P. Hart und Michael I. Levin; LISP 1.5 Programmer's Manual; Verlag: The MIT Press; Adresse: Cambridge, Massachusetts; Web-Link; 1965 (Buch), S. 2, S. 4
McCarthy et. al. (1965), S. 39, S. 58
Steele (1990): Guy L. Steele Jr.; Common Lisp – The Language; Verlag: Addison Wesley Longman; Adresse: Bonn; ISBN: 3-8273-1023-7; Web-Link; 1996 (Buch), S. 431, https://www.cs.cmu.edu/Groups/AI/html/cltl/clm/node153.html
Steele (1990), S .238 – 239, https://www.cs.cmu.edu/Groups/AI/html/cltl/clm/node108.html
Schmidt (1966), S. 119, S. 120, S. 122
Ebbinghaus (2003): Heinz-Dieter Ebbinghaus; Einführung in die Mengenlehre; Reihe: Hochschultaschenbuch; Auflage: 4; Verlag: Spektrum Akademischer Verlag; Adresse: Heidelberg, Berlin; ISBN: 3-8274-1411-3; 2003; Quellengüte: 5 (Buch), S. 59 – 60
Ebbinghaus (2003), S. 55, S. 59 – 60

7 Siehe auch

Kowarschick (MMDB-Skript): Wolfgang Kowarschick; Vorlesung Multimedia-Datenbanksysteme – Sommersemester 2018; Hochschule: Hochschule Augsburg; Adresse: Augsburg; Web-Link; 2018; Quellengüte: 4 (Skript)

Kategorien:
Mengenlehre
Datenmanagement
Glossar

Diese Seite wurde zuletzt am 7. Oktober 2019 um 14:04 Uhr bearbeitet.
Inhalt verfügbar unter CC BY-SA 4.0.