5 – BINÄRE ENTSCHEIDUNGS-DIAGRAMME (BDDS)

Sommersemester 2009

Dr. Carsten Sinz, Universität Karlsruhe

Datenstruktur BDD

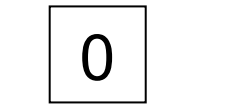
- 1986 von R. Bryant vorgeschlagen
- zur Darstellung von aussagenlogischen Formeln (genauer: Booleschen Funktionen)
  - Boolesche Fkt.: f: $\mathbf{B}^n \rightarrow \mathbf{B}$ (für $n \in \mathbf{N}$)
- f wird repräsentiert als gerichteter azyklischer Graph (DAG)
  - „Baum mit Sharing“
  - Blätter: Boolesche Konstanten 0/1;
    innere Knoten: markiert mit Variablen, genau zwei Nachfolger (über 0-/1-Kante)

BDD: Semantik

- Terminalknoten stellen konstante Funktion 0 bzw. 1 dar
- Innere Knoten interpretiert als:

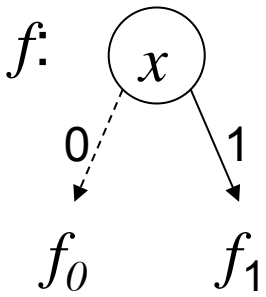
$$f = \text{if } x \text{ then } f_1 \text{ else } f_0$$
$$= (x\ ?\ f_1 : f_0)$$
$$= (x \Rightarrow f_1) \wedge (\neg x \Rightarrow f_0)$$

[f_0 und f_1 wiederum BDD-Knoten]



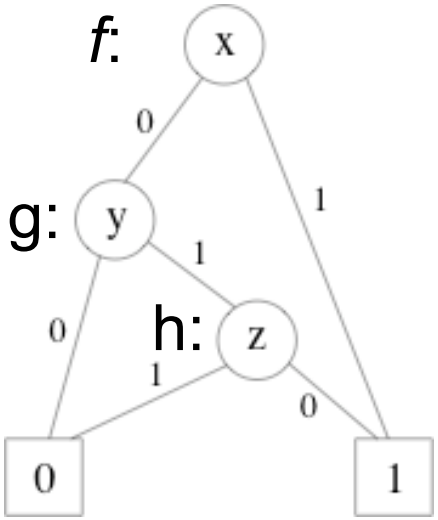
Konstante 0

Konstante 1



Innerer Knoten mit
- Entscheidungsvariable x
- f_0 an 0-Kante und
- f_1 an 1-Kante

BDD: Beispiel



BDD-Darstellung
von $x \vee (y \wedge \neg z)$

f = if x then 1 else g

= if x then 1 else (if y then h else 0)

= if x then 1 else

(if y then (if z then 0 else 1) else 0)

= if x then 1 else (if y then ¬z else 0)

= if x then 1 else (y⇒¬z)∧(¬y⇒0)

= if x then 1 else (y⇒¬z)∧y

= if x then 1 else ¬z∧y

= (x⇒1)∧(¬x⇒¬z∧y)

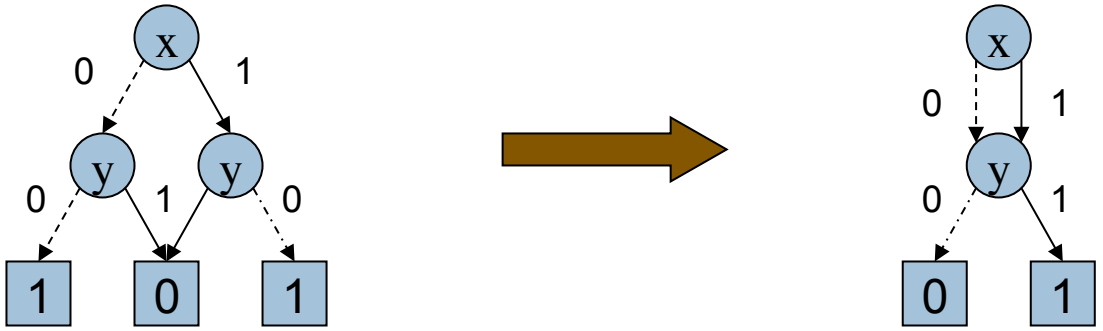
= x∨(¬z∧y)

Reduced Ordered BDDs (ROBDDs)

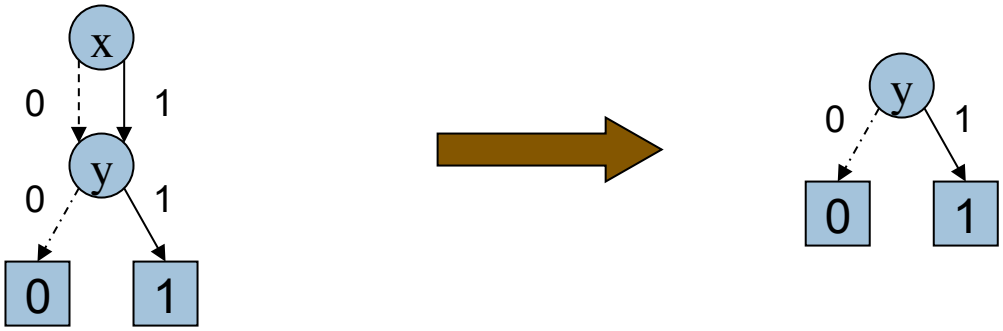
1. Totale Ordnung auf Variablen: $\prec$
   - Bedingung: Variable des Elternknotens muss kleiner sein als Variable beider Kindknoten
   - Auf jedem Pfad kommt jede Variable höchstens einmal vor
2. Reduktionen:
   a) Isomorphe Teilbäume dürfen nicht mehrfach vorkommen, mehrere Vorkommen werden auf eines reduziert.
   b) Knoten f mit $f_0 = f_1$ werden gelöscht und Kanten, die auf f zeigen auf f_0 (bzw. f_1) umgebogen.

ROBDDs: Beispiel Reduktionen

Reduktion a) [gleiche Teilbäume nur 1x]



Reduktion b) [gleiche Nachfolger, d.h. $f_0=f_1$]

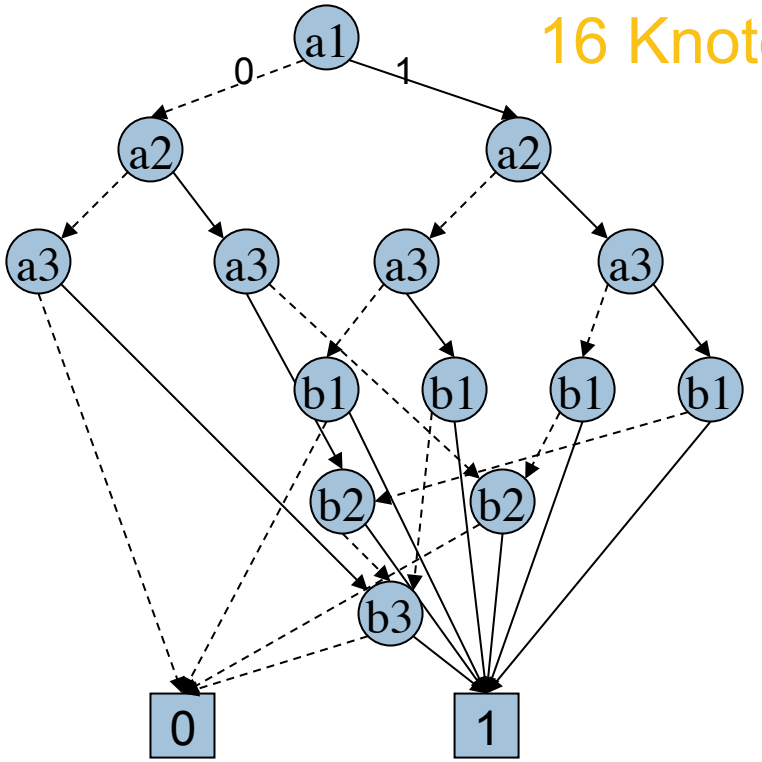


ROBDDs: Wichtige Eigenschaften

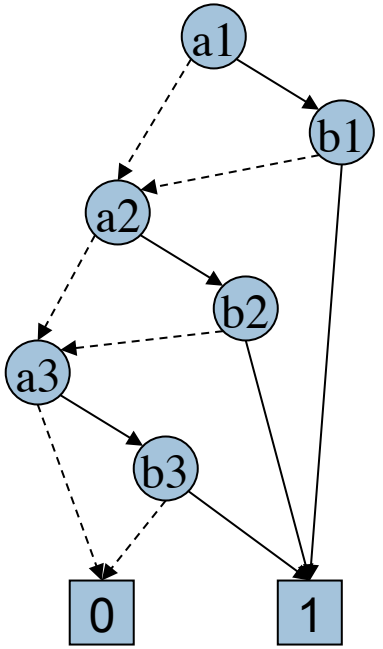
- ROBDD *b* zu gegebener Booleschen Funktion *f* eindeutig, d.h.:
  - *b* ist eindeutiger Repräsentant aller zu *f* äquivalenten Formeln.
  - Spezialfall: Unerfüllbare (allgemeingültige) Formeln werden durch 0-Knoten (1-Knoten) repräsentiert.
- Größe des ROBDDs kann stark von der gewählten Variablenordnung $\succ$ abhängen.

Abhängigkeit von Variablenordnung

Bsp. $f = a_1*b_1+a_2*b_2+a_3*b_3$



16 Knoten



8 Knoten

$a_1 < a_2 < a_3 < b_1 < b_2 < b_3$

$a_1 < b_1 < a_2 < b_2 < a_3 < b_3$

Generierung von (RO)BDDs

1. „Top-Down“-Ansatz
   - Zerlegung der Eingabeformel f unter gleichzeitiger Generierung des BDDs, d.h.
     a) Selektion der kleinsten in f vorkommenden Variablen
     b) Berechnung von f_0 und f_1
     c) Rekursive BDD-Transformation von f_0 und f_1
2. „Bottom-Up“-Ansatz
   - BDDs für atomare Formeln (Variablen/0/1)
   - Operationen zur Generierung komplexer BDDs aus einfacheren (z.B. BDD-Disjunktion, BDD-Konjunktion)

Top-Down-Ansatz: Grundlagen

- **Definition:** Restriktion

  Sei F eine aussagenlogische Formel, x eine Variable und $b \in \{0,1\}$. Die Restriktion $F|_{x=b}$ ist dann rekursiv wie folgt definiert:

$$0|_{x=b} = 0$$

$$1|_{x=b} = 1$$

$$y|_{x=b} = \begin{cases} 1 & \text{falls } x = y \text{ und } b = 1 \\ 0 & \text{falls } x = y \text{ und } b = 0 \\ y & \text{sonst} \end{cases}$$

$$(\neg G)|_{x=b} = \neg(G|_{x=b})$$

$$(G \vee H)|_{x=b} = G|_{x=b} \vee H|_{x=b}$$

$$(G \wedge H)|_{x=b} = G|_{x=b} \wedge H|_{x=b}$$

Top-Down-Ansatz

- Shannon-Expansion:

$$f = (x \wedge f|_{x=1}) \vee (\neg x \wedge f|_{x=0})$$

$$= (x \Rightarrow f|_{x=1}) \wedge (\neg x \Rightarrow f|_{x=0})$$

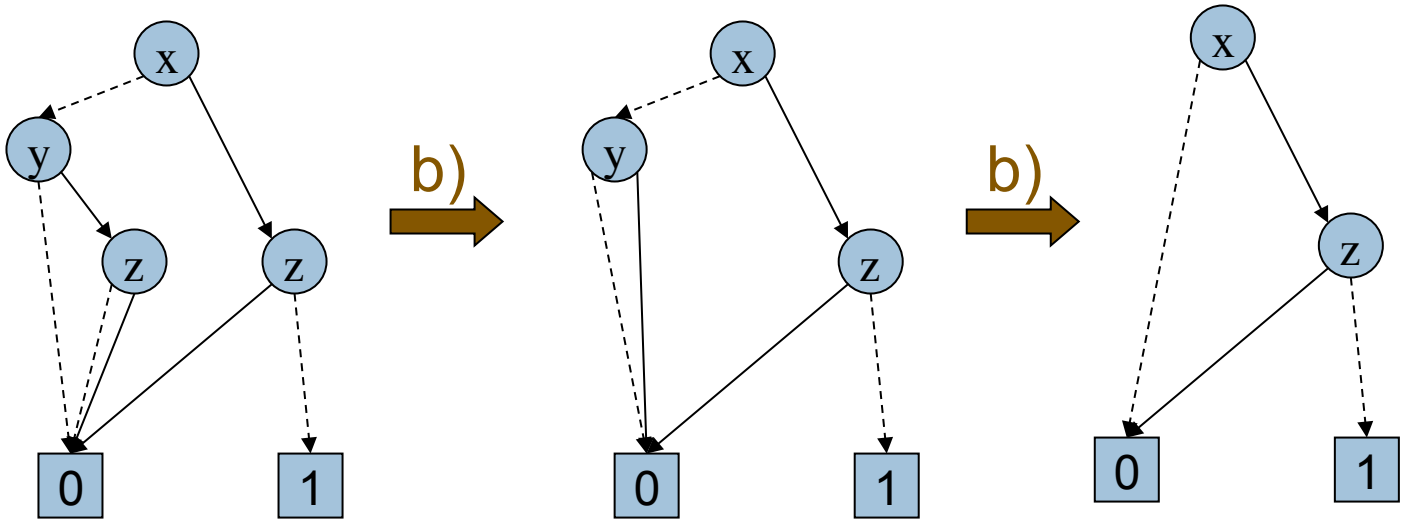
  - f_0 und f_1 werden Kofaktoren von f (bezüglich *x*) genannt.
  - Benannt nach Claude Shannon (1916-2001)
- Top-Down-Ansatz berechnet BDDs mittels rekursiver Shannon-Expansion
  - d.h. $f_1 = f_{x=1}$, $f_0 = f_{x=0}$ für einen BDD-Knoten mit Variable *x*
  - Nachträgliche Reduktion erforderlich.

Top-Down-Ansatz: Nachträgliche Reduktion

Bsp: $f = (x \vee y) \wedge (x \vee \neg y \vee z) \wedge \neg z$, $z > y > x$

$f|_{x=0} = (0 \vee y) \wedge (0 \vee \neg y \vee z) \wedge \neg z = y \wedge (\neg y \vee z) \wedge \neg z$

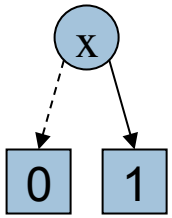
$f|_{x=1} = (1 \vee y) \wedge (1 \vee \neg y \vee z) \wedge \neg z = \neg z$



Bottom-Up-Ansatz

- BDDs für atomare Formeln plus Konstruktoren für komplexe Formeln
  - Boolesche Konstanten:

  - Variablen:



  - Komplexe Formeln:
    - Mittels BDD-Algorithmen

BDD-Algorithmus: „BDD-and“

Algorithm 1: BDD-and(a, b)

1: if $a = 0$ or $b = 0$ then return 0
2: if $a = 1$ then return b else if $b = 1$ then return a
3: (x, a_0, a_1) = decompose(a); (y, b_0, b_1) = decompose(b)
4: if $x > y$ then return new-node(y, BDD-and(a, b_0), BDD-and(a, b_1))
5: if $x = y$ then return new-node(x, BDD-and(a_0, b_0), BDD-and(a_1, b_1))
6: if $x < y$ then return new-node(x, BDD-and(a_0, b), BDD-and(a_1, b))

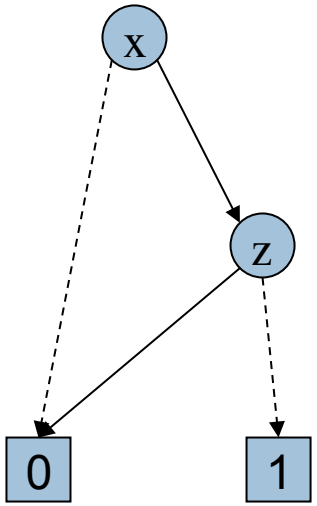
- (x,f_0,f_1) = decompose(f) zerlegt Knoten in Variable und Kofaktoren.
- new-node(x,f_0,f_1) legt neuen BDD-Knoten mit Variable x und Kofaktoren f_0 und f_1 an, sofern dieser noch nicht existiert. Ansonsten wird der bereits vorhandene Knoten verwendet.

Automatische Reduktion mittels *new-node*

- *new-node*(x,f_0,f_1) implementiert auch Reduktion:
  - Falls $f_0=f_1$, so wird f_0 zurückgegeben.
  - Falls ein Knoten (x,f_0,f_1) bereits angelegt wurde, so wird dieser zurückgegeben (implementiert mittels Hash-Tabelle)

Bottom-Up-Ansatz: Implementierung

- Hash-Tabelle zur Speicherung von nodes:



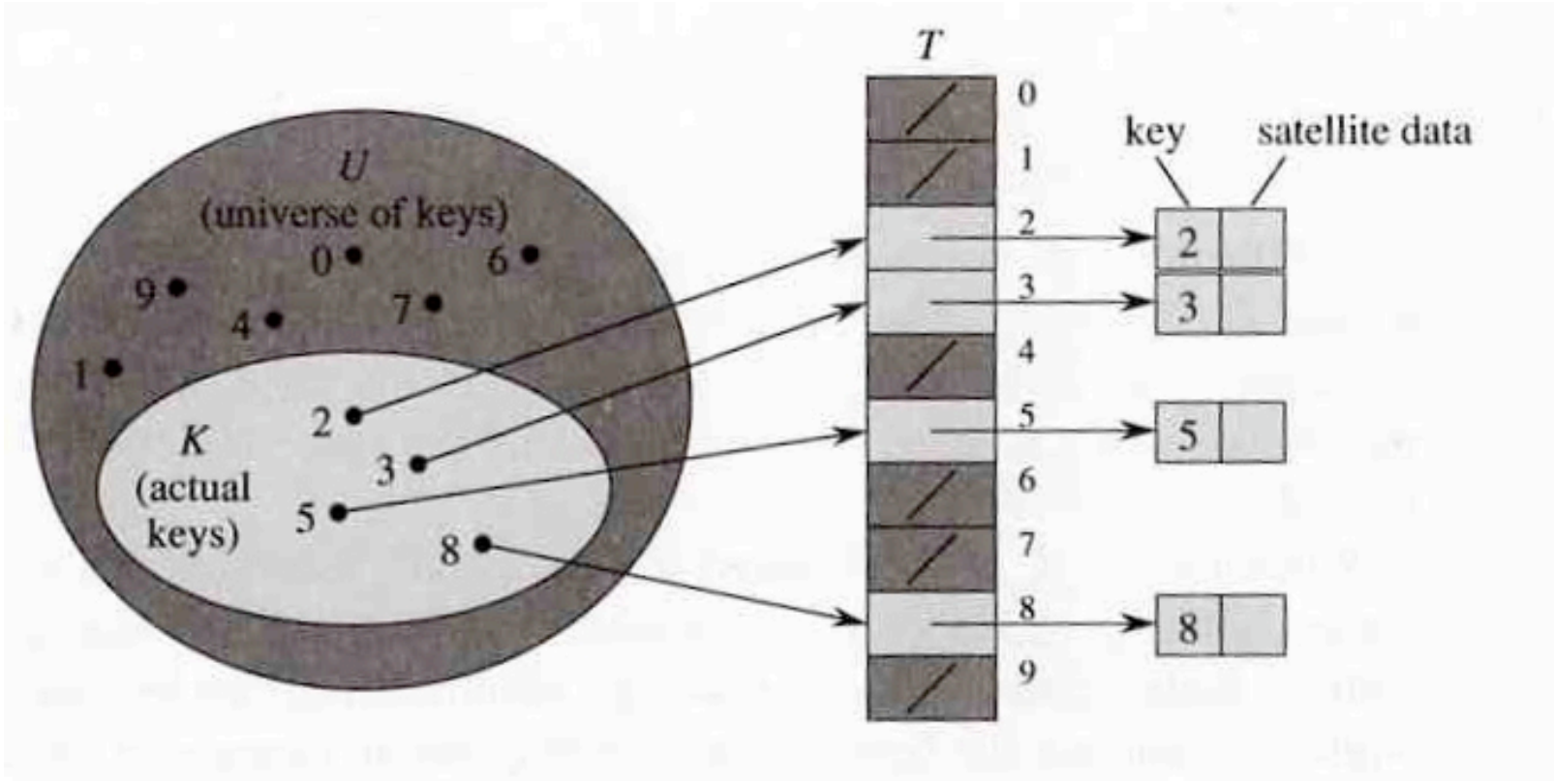
Idx	Var	Low	High
0	0	--	--
1	1	--	--
⋮			
6	z	1	0
⋮			
8	x	0	6
⋮			

- Weitere Hash-Tabelle zum Zwischenspeichern von Ergebnissen der BDD-Konstruktions-Algorithmen

Hashing (I)

- Verallgemeinerung der „array“-Datenstruktur
  - Array: direkte Adressierung der Elemente anhand von Index.
  - Hash-Tabelle: Berechnung des Indexes, an dem ein Element abgelegt wurde durch Hash-Funktion.
- Erlaubt Suchen, Einfügen und Löschen von Elementen in O(1).

Daten-Speicherung in Array



[Quelle: Corman/Leiserson/Rivest: Introduction to Algorithms]

Daten-Speicherung in Hash-Tabelle

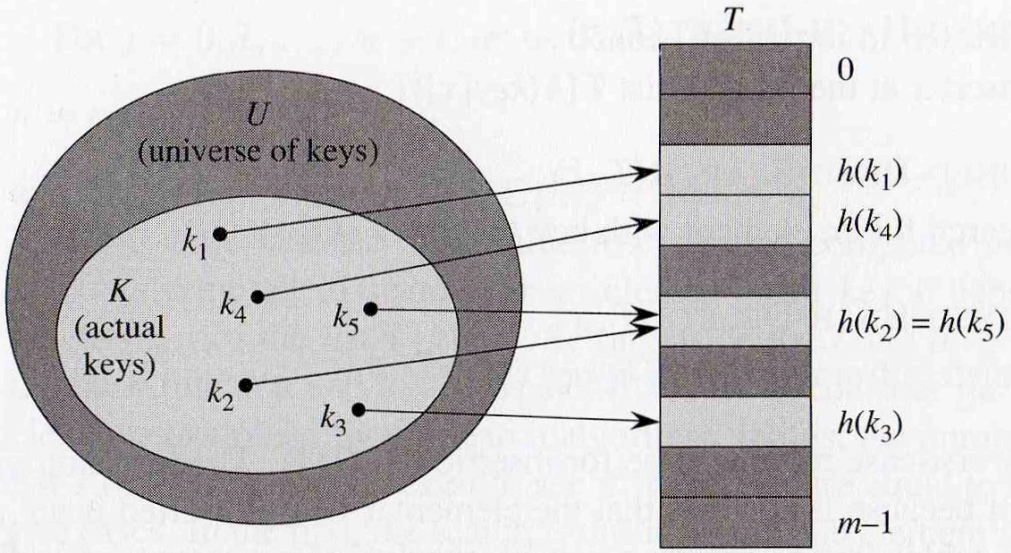


Figure 11.2 Using a hash function h to map keys to hash-table slots. Keys k_2 and k_5 map to the same slot, so they collide.

[Quelle: Corman/Leiserson/Rivest: Introduction to Algorithms]

Hash-Tabelle
Kollisions-Auflösung mittels *Chaining*

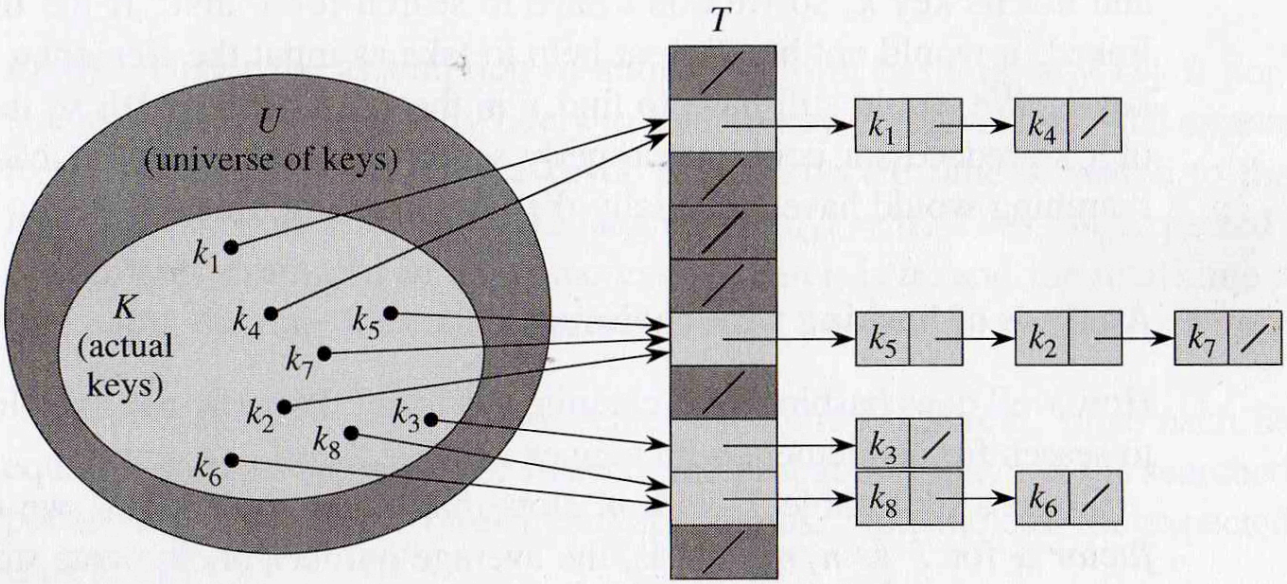


Figure 11.3 Collision resolution by chaining. Each hash-table slot $T[j]$ contains a linked list of all the keys whose hash value is j. For example, $h(k_1) = h(k_4)$ and $h(k_5) = h(k_2) = h(k_7)$.

[Quelle: Corman/Leiserson/Rivest: Introduction to Algorithms]

Hash-Funktionen

- Berechnen Indizes in Hash-Tabelle
- Gewünschte Eigenschaften:
  - möglichst gute Verteilung auf alle Indizes
- Problem:
  - zu speichernde Elemente im Voraus nicht bekannt
- Typische Hash-Funktionen:
  - $h(k) = k \bmod m$ (Divisions-Methode)
  - $h(k) = \lfloor m\,(k\,A \bmod 1) \rfloor$ mit $A \in [0,1]$
    (Multiplikations-Methode)

Hash-Tabellen: Kollisions-Auflösung

1. Chaining
2. Open Addressing
   - linear probing: $h(k,i) = (h'(k) + i) \bmod m$
   - quadratic probing:
     $h(k,i) = (h'(k) + c\,i + d\,i^2) \bmod m$
   - double hashing:
     $h(k,i) = (h_1(k) + i\,h_2(k)) \bmod m$

BDD-Varianten (I)

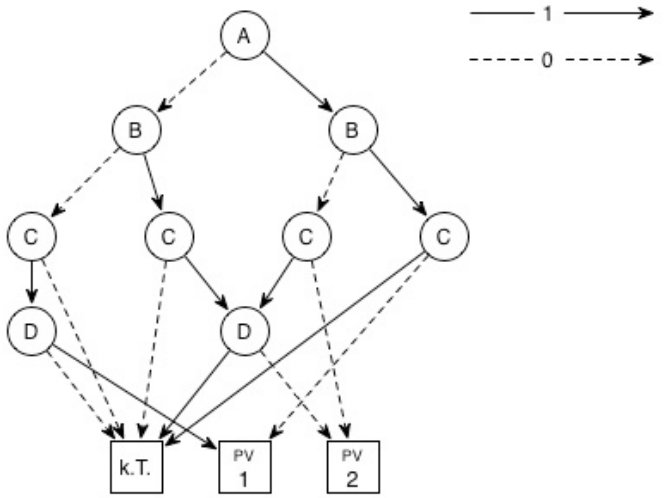
- FDDs: [Kebschull, Schubert, Rosenstiel; 1992]
  - Functional Decision Diagrams
  - (Positive) Davio-Expansion anstelle der Shannon-Expansion

$$f = f|_{x=0} \oplus x \wedge (f|_{x=0} \oplus f|_{x=1})$$

  - Reduktion b) ersetzt durch Reduktion c): Falls $f_1=0$, so ersetze f durch f_0.
- ZDDs: [Minato, 1992]
  - Zero-Suppressed Decision Diagrams
  - Zur Darstellung von Teilmengen einer endlichen Menge M
  - Unterschied BDD: fehlende Variablen auf Pfad werden als 0 interpretiert, nicht als don‘t care.

BDD-Varianten (II)

- MTBDDs:
  - Multi-Terminal BDDs
  - Einsatz z.B. in der Darstellung von Fahrzeugstücklisten



- ADDs:
  - Arithmetic Decision Diagrams
  - Ganze Zahlen in Terminalknoten
  - „Gewichtete Terme“

BDD-Pakete im Internet

- CUDD [http://vlsi.colorado.edu/~fabio/CUDD]
  - CU (University of Colorado) DD package
  - BDDs, ZDDs, ADDs
  - Dynamische Variablen-Umordnung
  - C / C++
- Buddy [http://buddy.sourceforge.net/]
  - Universität Kopenhagen
  - C++, Dynamische Variablen-Umordnung
  - Graphische Ausgabe von BDDs