

Automatische Dekodersynthese für den retargierbaren Befehlssatzsimulator Jahris

Statusvortrag zur Diplomarbeit

Ronald Rist

s0200738@mail.zih.tu-dresden.de

Dresden, 03.07.2013



Inhalt

1 Einleitung

2 Grundlagen

- *Formale Grundlagen*
- *Erstellung eines Dekoderbaumes*
- *Optimierung eines Dekoderbaumes*

3 Entwurf: Erweiterung der HPADL-Syntax

- *Definition von verschiedenen Befehlswordformaten*
- *Beschreibung von Befehlen*

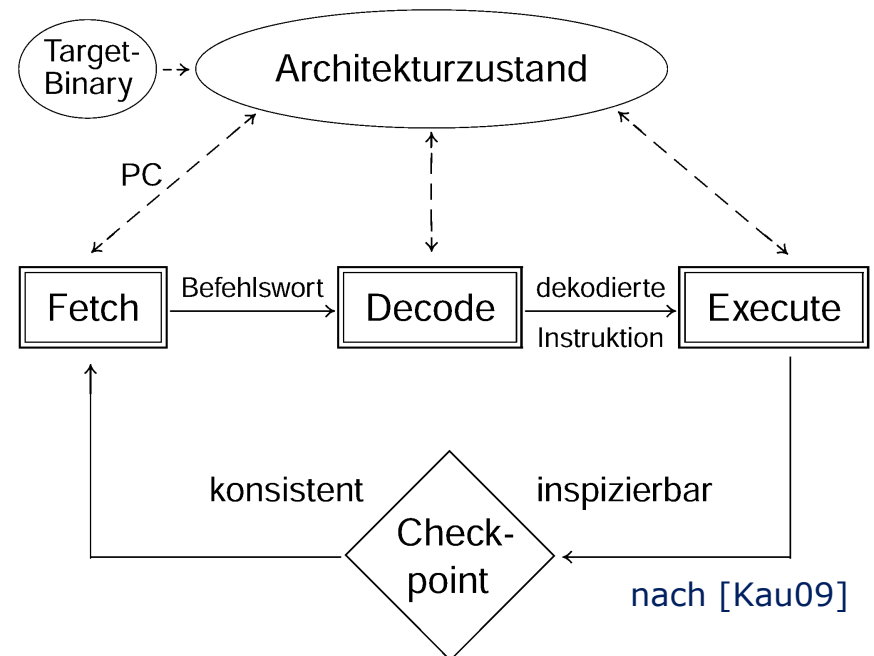
4 Ablaufplan

5 Ausgewählte Quellen

1 Einleitung

1.1 Hintergrund – retargierbarer Befehlssatzsimulator Jahris

- von Marco Kaufmann in Java entwickelt, plattformunabhängig
- Simulation verschiedenster Architekturen/Befehlssätze, diese werden in HPADL beschrieben
- Beschränkung auf befehlsgenaue Simulation
- ausgelegt auf hohe Performanz durch Just-in-Time-Compilierung größerer Codeabschnitte
- interpretierende schrittweise Simulation ebenfalls möglich



1.2 Ausgangspunkt



Test/Debug auf Zielsystem



Einehbarkeit?

???



Entwickler



Test/Debug im Simulator



Geschwindigkeit?



Entwickler

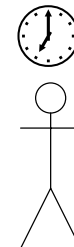
- Vereinbarkeit von Inspizierbarkeit und hoher Simulationsperformanz?
→ *in Jahris gewährleistet*
- **neue Problemstellung:** Wie kann Entwicklung von Modellen für Jahris vereinfacht werden?

1.3 Motivation

- Entwicklung von Architektursimulatoren bzw. Architekturbeschreibungen für retargierbare Architektursimulatoren:
(abstrakte) Abbildung der physischen Pipeline-Stufen

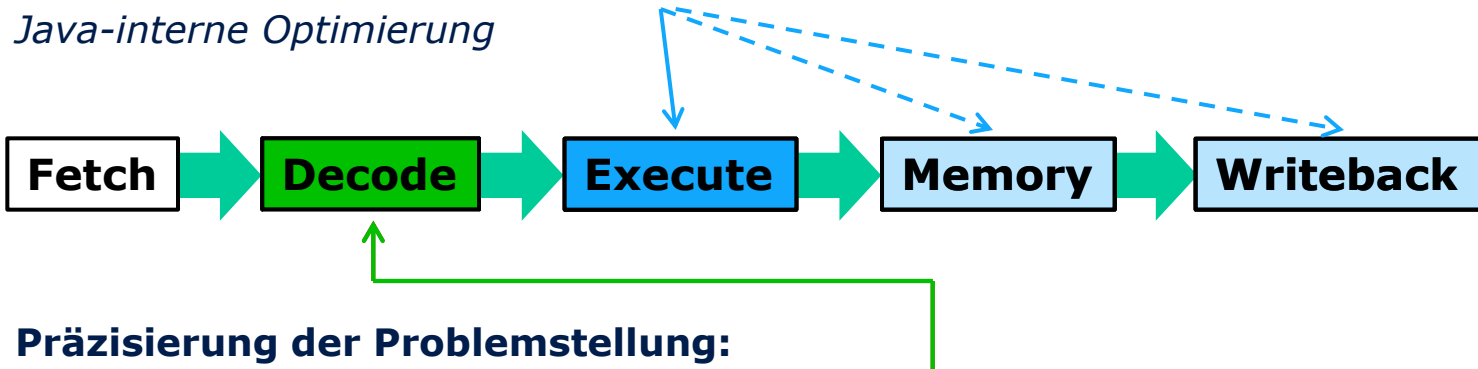


- Hauptfehlerquellen:
 - Fehler im Dekoderbaum
 - Fehler in der Befehlsimplementierung
- stellen gleichzeitig den Hauptzeitaufwand des Entwicklers dar (Erstellung sowie Optimierung von Hand)



1.3 Motivation (Fortsetzung)

- Jahris: Befehlsimplementierung durch Entwickler, jedoch *maschinelle* Optimierung der Befehle (bzw. deren Mikrooperationen) durch *Java-interne Optimierung*



- **Präzisierung der Problemstellung:**

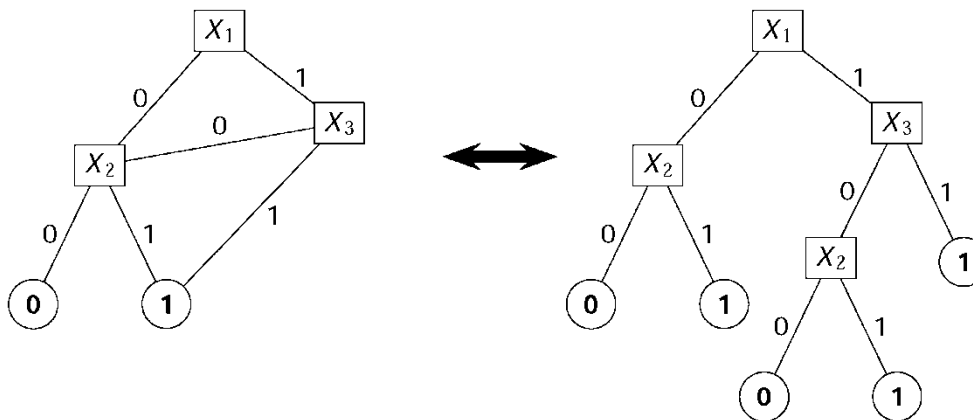
Kann Entwickler auch bei Erstellung des Dekoderbaumes entlastet werden?

- maschinelle Erzeugung
- ggf. automatische Optimierung
- Ausschluss des „Fehlerfaktors Mensch“

2 Grundlagen

2.1 Formale Grundlagen

- Abbildung von Befehlsdekodern in sog. *Entscheidungs*bäumen
- Zuordnung von Befehlsworten zu Operationen schrittweise hierarchisch
- *Baumstruktur*



nach [Mor82]

2.2 Erstellung eines Dekoderbaumes

- normierte Beschreibung aller verfügbaren Befehle der zu simulierenden Architektur
- **Aufgabe:**
Entwicklung/Adaptierung eines geeigneten Formats (XML, binär, ...?)
- Repräsentation als „Quelltext“ und Repräsentation im Speicher?

2.2 Erstellung eines Dekoderbaumes – Was soll abbildbar sein?

15	10	9	8	6	5	3	2	0	
0	0	0	1	1	0	A	Rm	Rn	Rd

(1) ADD | SUB Rd, Rn, Rm

15	10	9	8	6	5	3	2	0	
0	0	0	1	1	1	A	#imm3	Rn	Rd

(2) ADD | SUB Rd, Rn, #imm3

15	13	12	11	10	8	7	0
0	0	1	Op	Rd/Rn	#imm8		

(3) <Op> Rd, Rn, #imm8

15	13	12	11	10	6	5	3	2	0
0	0	0	Op	#sh	Rn	Rd			

(4) LSL | LSR | ASR Rd, Rn, #sh

15	10	9	6	5	3	2	0	
0	1	0	0	0	0	Op	Rm/Rs	Rd/Rn

(5) <Op> Rd/Rn, Rm/Rs

15	10	9	8	7	6	5	3	2	0	
0	1	0	0	0	1	Op	D	M	Rm	Rd/Rn

(6) ADD | SUB | MOV Rd/Rn, Rm

15	12	11	10	8	7	0
1	0	1	0	R	Rd	#imm8

(7) ADD Rd, SP | PC, #imm8

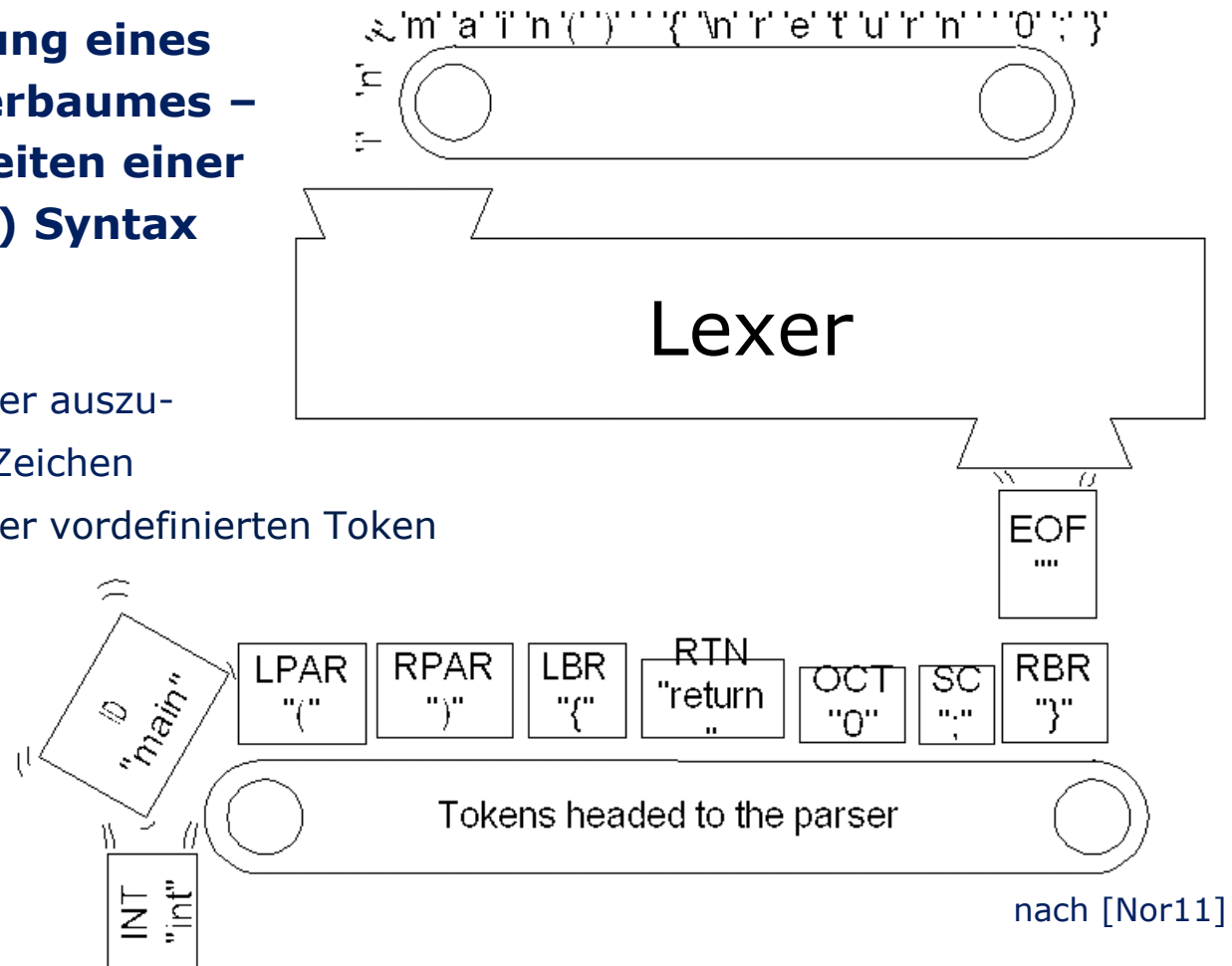
15	8	7	6	0					
1	0	1	1	0	0	0	0	A	#imm7

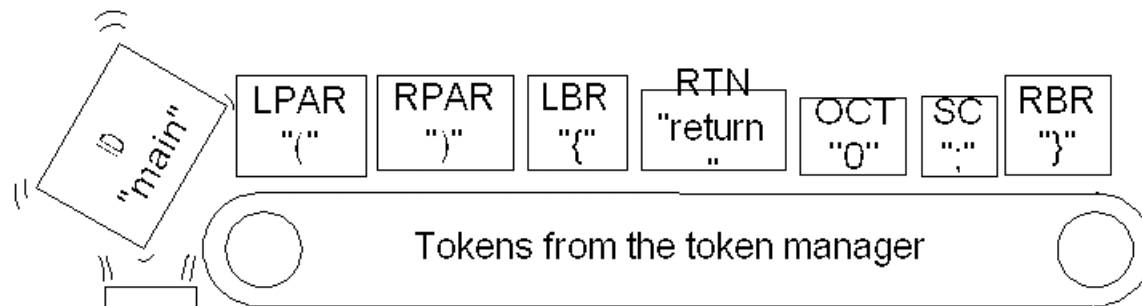
(8) ADD | SUB SP, SP, #imm7

nach [Fur00]

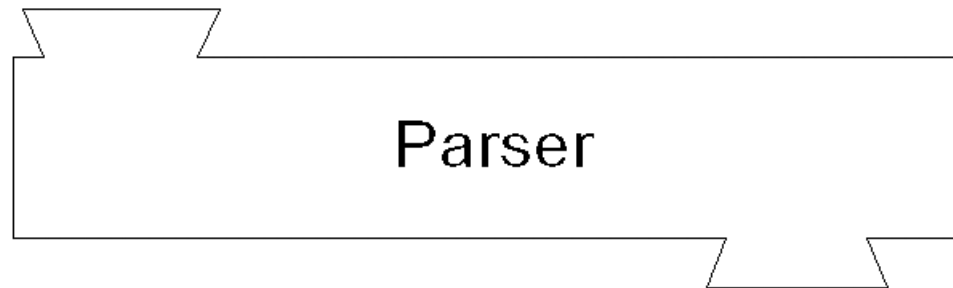
2.2 Erstellung eines Dekoderbaumes – Verarbeiten einer (neuen) Syntax

- Definition der auszulassenden Zeichen
- Definition der vordefinierten Token

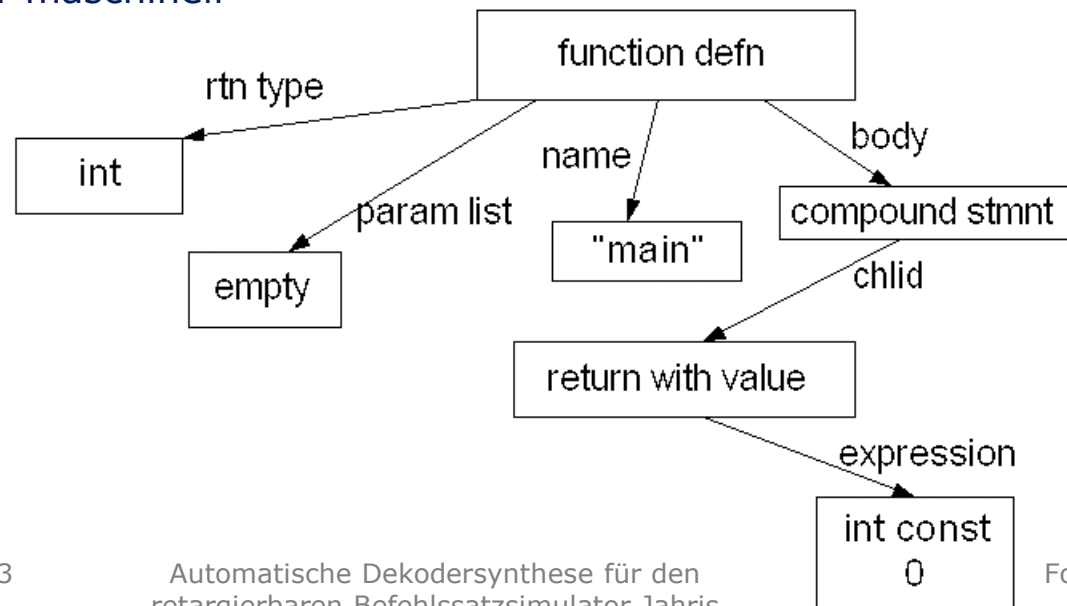




2.2 ... – bzw. Grammatik




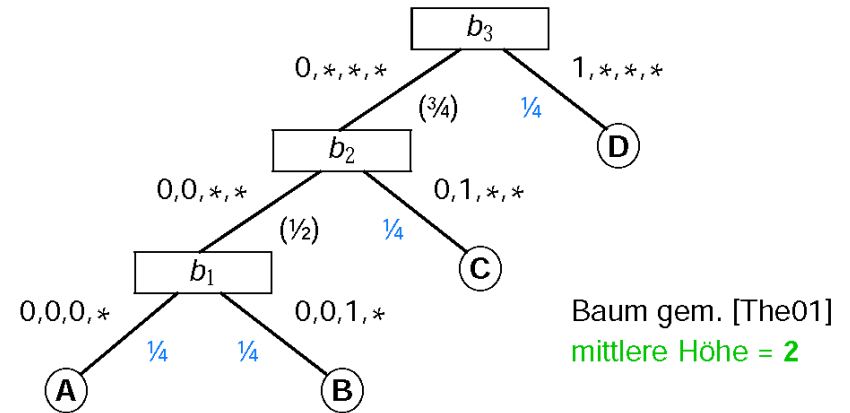
- Erzeugen einer maschinell
verwertbaren
Struktur




aus [Nor11]

2.2 Erstellung eines Dekoderbaumes – Kodierung

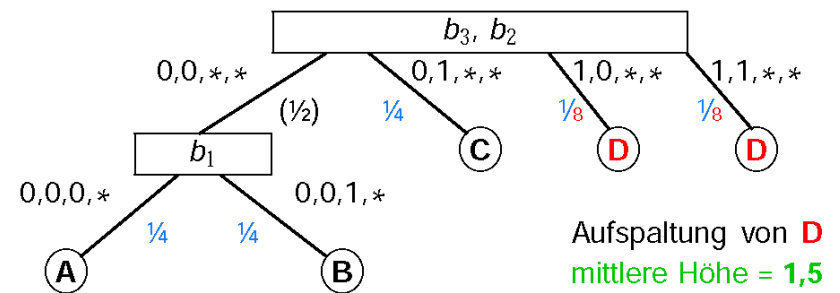
Befehl	Pattern	$p(.)$
	$b_3 \ b_2 \ b_1 \ b_0$	
A	0 0 0 *	$\frac{1}{4}$
B	0 0 1 *	$\frac{1}{4}$
C	0 1 * *	$\frac{1}{4}$
D	1 * * *	$\frac{1}{4}$



Befehl	Pattern	$p(.)$
	$b_3 \ b_2 \ b_1 \ b_0$	
A	0 0 0 *	$\frac{1}{4}$
B	0 0 1 *	$\frac{1}{4}$
C	0 1 * *	$\frac{1}{4}$
D	1 * * *	$\frac{1}{4}$

$(b_3 \ b_2)$		Pattern	$p(.)$
0	1		
0	0	A 0 0 0 *	$\frac{1}{4}$
0	1	B 0 0 1 *	$\frac{1}{4}$
1	0	C 0 1 * *	$\frac{1}{4}$
1	1	D 1 0 * *	$\frac{1}{8}$
1	1	D 1 1 * *	$\frac{1}{8}$

D geteilt



nach [Qin+03], modifiziert

2.3 Optimierung eines Dekoderbaumes

- Festlegen von Bewertungskriterien für *Effizienz*
 - maximale Baumtiefe (längster Pfad, ungünstigster Überprüfungsaufwand)
 - mittlere Baumtiefe (ggf. auch Bewertung von Pfaden mit voneinander abweichenden Auftrittswahrscheinlichkeiten)
 - Speichergröße des Baumes
 - ggf. *einstellbare* Kriterien
- Art und Weise der Optimierung?
 - „**brute-force**“: zeitaufwändig, aber: Findung optimalster Lösung garantiert
 - Backtracking zur Reduktion des Aufwands?
 - **beschleunigte Algorithmen**
 - u.U. 90% (oder 99%) des Optimums in einem Bruchteil der Zeit?

2.3 Optimierung eines Dekoderbaumes – Lösungsraum

- Anzahl möglicher Bäume einer Funktion

$$A_B(k) = \prod_{i=0}^{k-1} (k-i)^{2^i}$$

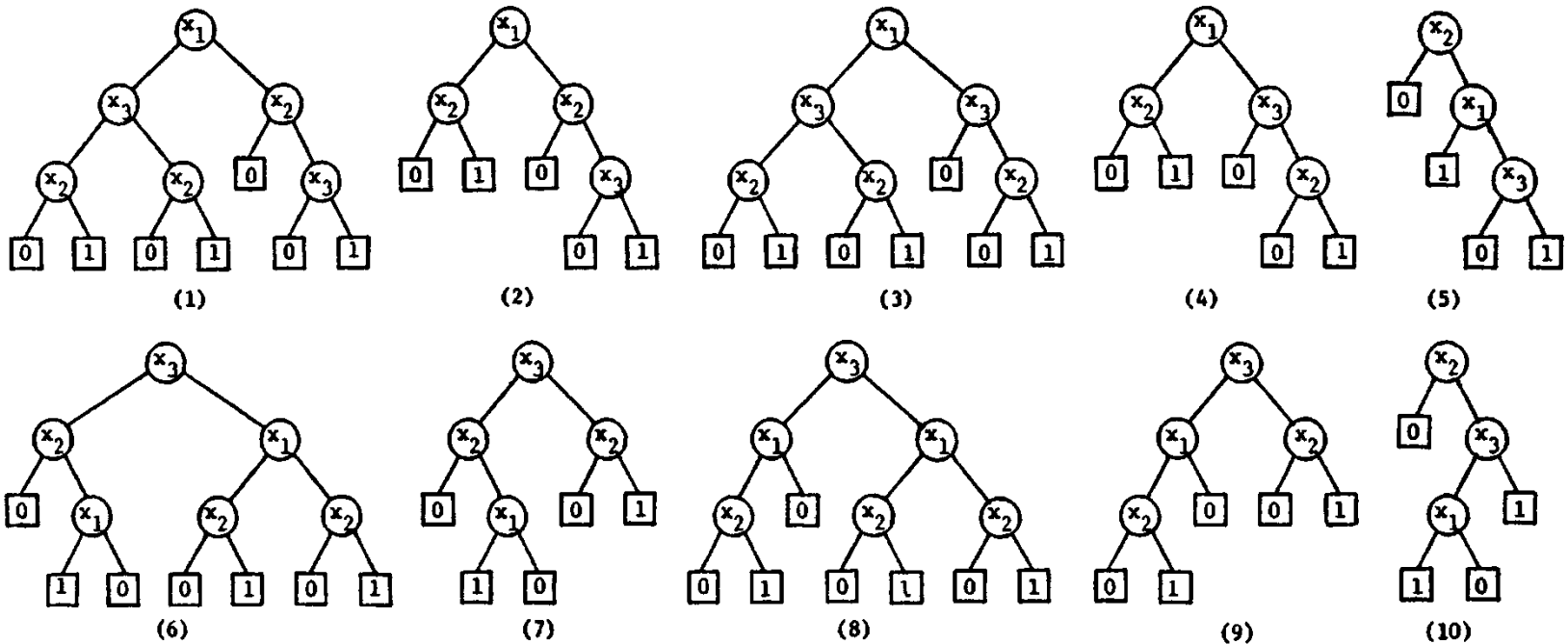
$$A_B(k+1) = (k+1) * (A_B(k))^2$$

Anzahl der Variablen k	Anzahl der mögl. Bäume A_B
1	1
2	2
3	12
4	576
5	1.658.880
6	16.511.297.126.400
...	...

nach [Mor82]

2.3 Optimierung eines Dekoderbaumes – Beispiel

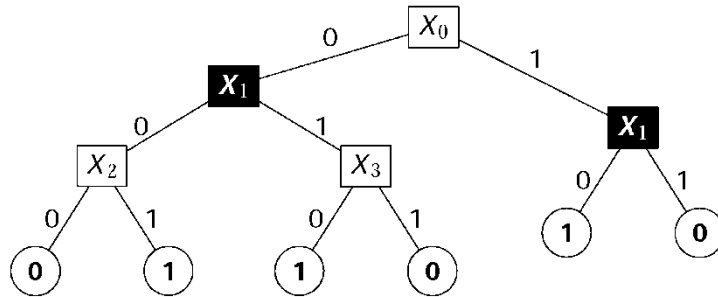
- **Beispiel:** Bäume der Funktion $\bar{x}_1 \& x_2 \parallel x_2 \& x_3$ mit 3 Variablen



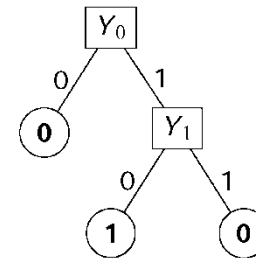
aus [Mor82]

2.3 Optimierung eines Dekoderbaumes – Baumkomposition

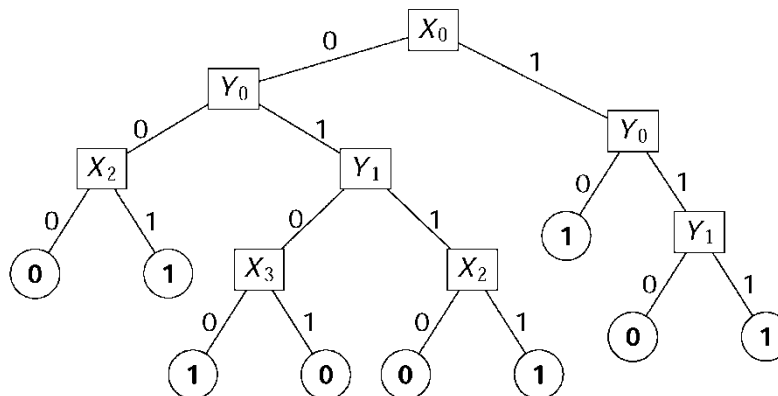
- **Beispiel:** Zusammensetzung von Unterbäumen



$$Z = f(X_0, X_1, X_2, X_3)$$



$$\begin{aligned}
 \mathbf{X_1} &= g(Y_0, Y_1) \\
 &= Y_0 \& \overline{Y_1}
 \end{aligned}$$



$$\begin{aligned}
 Z &= f(X_0, g(Y_0, Y_1), X_2, X_3) \\
 &= f(X_0, Y_0 \& \overline{Y_1}, X_2, X_3) \\
 &= h(X_0, X_2, X_3, Y_0, Y_1)
 \end{aligned}$$

nach [Mor82]

3 Entwurf: Erweiterung der HPADL-Syntax

3.1 Definition von verschiedenen Befehlswordformaten

```
format f1(opc) {  
    opc: int[4];  
    op1: int[4];  
    op2: int[4];  
    opD: int[4];  
}
```

```
format f2(opc,opc2) {  
    opc: int[4];  
    op1D: int[5];  
    opc2: int[2];  
    op2: int[3];  
    op3: int[2];  
}
```

```
format f3(opc) {  
    opc: int[6];  
    op1D: int[6];  
    op2: int[4];  
}
```

- der Identifikation dienende Opcode-Teile eindeutig vorgeben (vgl. Parameter)
- Operanden bereits hart typisieren → vereinfacht z.B. spätere Vorzeichenerweiterung

3.2 Beschreibung von Befehlen

```
f1(10) { rn=op1; rm=op2; rd=opD; ls1rn; ls2rm; ADD; sdrd; } //ADD8
```

```
f1(12) {                                     SUB;          } //SUB8
```

```
f1(5)  { if (opD==15) B; else ADD }
```

- direkte Verwendung von (bisherigem) HPADL-Code in der Befehlsdefinition

```
f2(7,1) { ... }
```

- effektive Definition von geteilten Opcodes (vgl. vorige Folie)
- **OFFEN:** effizientere Umsetzung des bisherigen Registerzugriffs möglich?
→ parametrisierbarer Aufruf, Umsetzung dennoch zur Dekodierzeit?

z.B. `ls1(op1);` → `ls1(4);` → `LS1R4;`

3.2 Beschreibung von Befehlen (Fortsetzung)

`f3(0x1D) { ... }`

`f3(0b011101) { ... }`

- vorherige Definition der Befehlswortabschnitte → vereinfachte Lesbarkeit bei „krummen“ Längen

`f3(0x3F) { fetch(16); ... }`

- Nachladen weiterer Befehlswortbestandteile
- **OFFEN:** Unterteilung in Formate-Block und Befehle-Block *ODER* Intermix mit Hilfe von Keywords (z.B. **format**, **instr**) ?

4 Ablaufplan

- Recherche und zusammenfassende Dokumentation
- Vorüberlegungen



Nächste Schritte:

- Modifizierung des HPADL-Parsers (*parser.jj*) zum Einlesen der neuen Syntax
- **Implementierung eines *reinen Dekoderbaum-Syntheseautomaten (Modifizierung des *resolver*)***
- Erstellung einer Beschreibung von Thumb im neuen Format (ggf. Modifikationen am neuen Format)
- Synthese dieser Beschreibung, Vergleich mit existierendem Modell, **Korrekturen!** (kurzes) Benchmark gegen existierendes Modell

MEILENSTEIN!

- **Dokumentation der Implementation**
- anschließend: Implementierung von Thumb2, ARM, u.a.
- Erweiterung des Synthesealgorithmus um Optimierung (geringere Relevanz)
- ausführliche, systematische Benchmarks und deren Dokumentation

5 Ausgewählte Quellen

- [Kau09] Kaufmann, M.: Erschließung von Just-in-Time-Compilierungstechniken in der Realisierung eines retargierbaren Architektursimulators, TU Dresden, 2009
- [Mor82] Moret, B. M. E.: Decision Trees and Diagrams, University of New Mexico, ACM, CSUR 1982, S. 593-623, ISSN 0360-0300
- [Qin⁺03] Qin, W.; Malik, S.: Automated Synthesis of Efficient Binary Decoders for Retargetable Software Toolkits, Princeton University, ACM, DAC 2003, S. 764-769, ISBN 978-1-58113-688-3
- [Nor11] Norvell, Th. S.: The JavaCC FAQ,
<http://www.engr.mun.ca/~theo/JavaCC-FAQ/javacc-faq-moz.htm>