Attributierte Grammatiken

AG spezifiziert abhängige Berechnungen im Strukturbaum

deklarativ: nur explizite Abhängigkeiten
keine implizite Ausführungsreihenfolge

Generator erzeugt **Plan für Baumdurchlauf**
mit Aufrufen der Berechnungen
spezifizierte Abhängigkeiten werden eingehalten
berechnete Werte werden durch den Baum transportiert

Ergebnis: Attributauswerter
(anwendbar für jeden Baum zur AG)

Strategien für Baumdurchlauf:	**AG-Klasse**
k-mal links-abwärts	**LAG (k)**
k-mal alternierend links/rechts-abwärts	**AAG (k)**
1-mal aufwärts	**SAG**
Besuchssequenzen: individueller Plan für jede Produktion der abstrakten Syntax	**OAG**

Generator analysiert Abhängigkeiten, prüft Anwendbarkeit der Strategie, erzeugt Plan.

Paß-orientierte Strategien (LAG, AAG, SAG) auch manuell anwendbar; Besuchssequenzen mit Generator.

Vorlesung Übersetzer WS 97/98 / Folie 46

Ziele:

- AG als Spezifikation für generierten Baumdurchlauf zeigen

in der Vorlesung:

- Zusammenhang Durchlaufstrategien und Abhängigkeiten zeigen.

nachlesen:

Kastens / Übersetzerbau, Abschnitt 5, 5.1

Verständnisfragen:

- Warum ist es nützlich, die Ausführungsreihenfolge der Berechnungen NICHT zu spezifizieren?

AG zu: Zielbaum erzeugen (U-45)

```
RULE:     Stmt ::= Variable ':=' Expr            COMPUTE
    Stmt.Code = MkAssign (Variable.Code, Expr.Code);
END;

RULE:     Variable ::= Variable '.' Selector    COMPUTE
    Variable[1].Code =
        MkSelect (Variable[2].Code, Selector.Key);
END;

RULE:     Variable ::= Variable '[' Expr ']'     COMPUTE
    Variable[1].Code =
        MkIndex (Variable[2].Code, Expr.Code);
END;

RULE:     Variable ::= UseIdent                  COMPUTE
    Variable.Code = MkAddr (UseIdent.Key);
END;

RULE:     Expr ::= UseIdent                      COMPUTE
    Expr.Code = MkCont (MkAddr (UseIdent.Key));
END;
```


Ziele:

Beispiel für einen AG-Ausschnitt in LIDO-Notation

Grundkonzepte von AGn

AG spezifiziert Berechnungen im Strukturbaum:

kontextfreie Baumgrammatik

Produktion + Berechnungen

p: Y ::= u g (...)

wird in jedem Knotenkontext p ausgeführt

AG spezifiziert Abhängigkeiten zwischen Berechnungen

Symbole + Attribute

X.b = f (Y.a) benutzt Ergebnis von g ( )

Y.a = g (...) vor f () ausgeführt

Nachbed. Vorbedingung

abhängige Berechnungen auch in benachbarten Kontexten:

r: X ::= v Y w X.b = f (Y.a)

p: Y ::= u Y.a = g (...)

Attribute können Abhängigkeiten spezifizieren, ohne einen Wert zu transportieren.

X.GotType = ResetTypeOf (...);

Y.Type = GetTypeOf (...) <- X.GotType;

vereinfachende Notationen (siehe AG-Sprache Lido):

- **Berechnungen zu Symbolen**
- **Zugriff auf Attribute entfernter Symbole**
- **links-rechts-Abhängigkeit zwischen Berechnungen**

Vorlesung Übersetzer WS 97/98 / Folie 47

Ziele:

- Grundverständnis für AGn vermitteln.
- Eine einfache Notation zeigen.

in der Vorlesung:

- weitere Beispiele

nachlesen:

Kastens / Übersetzerbau, Abschnitt 5, 5.1

Übungsaufgaben:

- Beispiel zur Einführung von attributierten Grammatiken in LIDO

AG Binärzahlen

Attribute:

- **v** Wert
- **lg** Anzahl der Ziffern in der Teilfolge
- **s** Skalierung

```
RULE p1:    D ::= L '.' L        COMPUTE
    D.v = ADD (L[1].v, L[2].v);
    L[1].s = 0;
    L[2].s = NEG (L[2].lg);
END;
RULE p2:    L ::= L B            COMPUTE
    L[1].v = ADD (L[2].v, B.v);
    B.s = L[1].s;
    L[2].s = ADD (L[1].s, 1);
    L[1].lg = ADD (L[2].lg, 1);
END;
RULE p3:    L ::= B              COMPUTE
    L.v = B.v;
    B.s = L.s;
    L.lg = 1;
END;
RULE p4:    B ::= '0'            COMPUTE
    B.v = 0;
END;
RULE p5:    B ::= '1'            COMPUTE
    B.v = Power2 (B.s);
END;
```


Ziele:

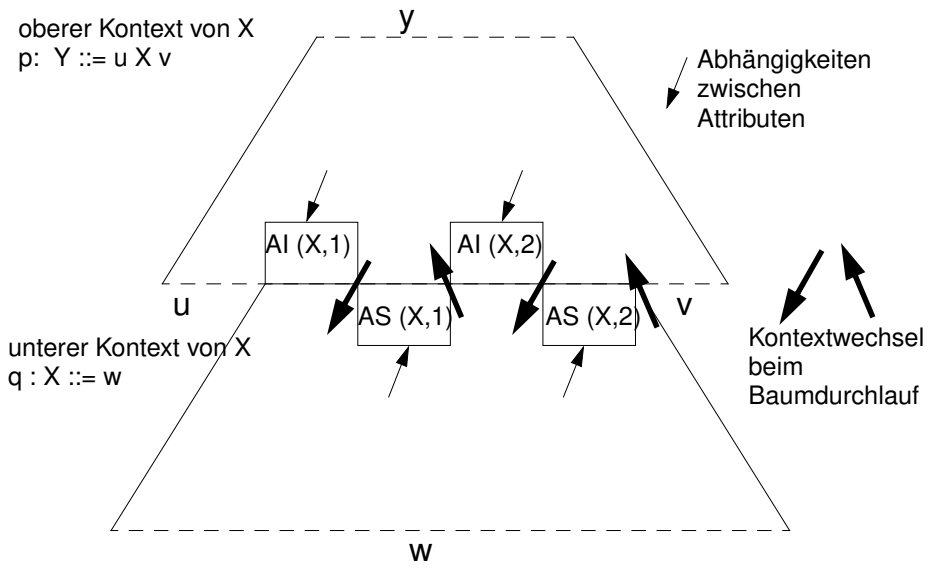
Beispiel zu Attributierten Grammtiken

Abhängigkeitsanalyse für AGn

Attributmenge zum Symbol X 2-geteilt:

AI (X) : inherited (erworben), in oberen Kontexten berechnet

AS (X): synthesized (abgeleitet), in unteren Kontexten berechnet.



Ziel: Zerlegung aller Attributmengen, so daß

AI (X, i) vor i-ten Besuch von X berechnet wird

AS (X, i) bei dem i-ten Besuch von X berechnet wird

Vorraussetzung für Zerlegung:

An keinem X-Knoten in keinem Strukturbaum gibt es direkte oder indirekte Abhängigkeiten entgegen der Folge

AI (X, 1), AS (X, 1), ..., AI (X, k), AS (X, k)

Vorlesung Übersetzer WS 97/98 / Folie 48

Ziele:

- Allgemeine Problemstellung der AG-Abhängigkeitsanalyse verstehen

in der Vorlesung:

- synthesized-inherited, oberer-unterer Kontext, Attributzerlegung erläutern

nachlesen:

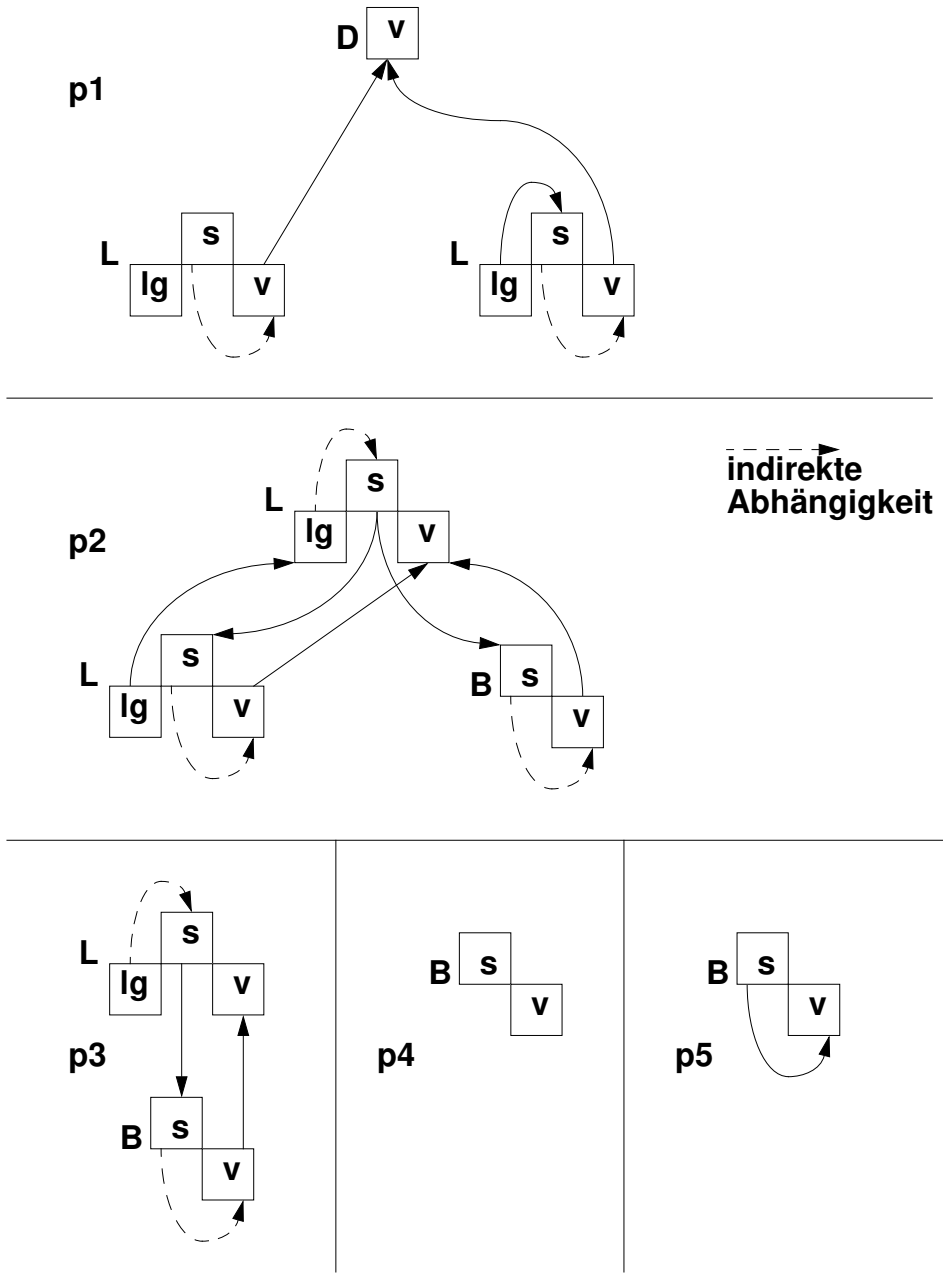
Kastens / Übersetzerbau, Abschnitt 5.2

Übungsaufgaben:

Konstruieren Sie möglichst kleine AGn mit folgenden Eigenschaften:

- Es gibt Strukturbäume mit und ohne Abhängigkeitszyklen und die Zyklen reichen über mehrere Kontexte.
- Für ein X gibt es eine Zerlegung mit k=2 aber nicht mit k=1.
- Es gibt keine Zerlegung, obwohl es keine Strukturbäume mit Zyklen gibt.

Abhängigkeitsgraphen zu AG Binärzahlen



Vorlesung Übersetzer WS 97/98 / Folie 48a

Ziele:

- Beispiel zu Zerlegungen (U-48)
- Beispiel zu LAG(k)-Algorithmus (U-50)

in der Vorlesung:

- Abhängigkeiten der Berechnungen in der AG als Graphen dargestellt

Besuchssequenzen

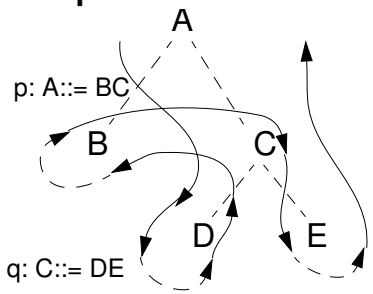
Eine Besuchssequenz (visit-sequence) vs_p zu jeder Produktion der Baumgrammatik:

$$p: X_o ::= X_1 \ldots X_i \ldots X_n$$

Folge von Operationen

↓ i, j	j-ter Besuch des i-ten Unterbaums
↑ j	j-te Rückkehr zum Vaterknoten
$eval_c$	Ausführung einer p zugeordneten Berechnung c

Beispiel im Baum:

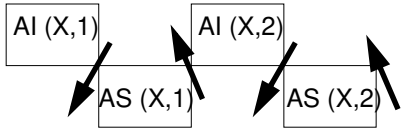


Besuchssequenzen

vs_p: ... ↓C,1 ...↓B,1 ...↓C,2 ...↑1

vs_q: ... ↓D,1 ... ↑1 ... ↓E,1 ... ↑2

Attributzerlegung garantiert korrekte Verzahnung



Implementierung:

je eine Prozedur für Besuchssequenz bis ↑

Aufruf (mit Verzweigung über Produktion) für ↓

Vorlesung Übersetzer WS 97/98 / Folie 49

Ziele:

- Baumdurchlauf gesteuert durch Besuchssequenzen
- Zusammenhang zu Attributzerlegung

in der Vorlesung:

- Prinzip erläutern

nachlesen:

Kastens / Übersetzerbau, Abschnitt 5.2.2

Übungsaufgaben:

- Zu einer kleinen Baumgrammatik Besuchssequenzen angeben, die bestimmte Aufgaben erfüllen.

Verständnisfragen:

- Skizzieren Sie Besuchssequenzen für die Operationen DefineIdn und KeyInEnv in einer Block-strukturierten Sprache mit Algol- bzw. C-Gültigkeitsregeln.

Besuchssequenzen für AG Binärzahlen

vs_{p1}: D ::= L '.' L

↓L[1],1; L[1].s=0; ↓L[1],2; ↓L[2],1; L[2].s=NEG(L[2].lg);

↓L[2],2; D.v=ADD(L[1].v, L[2].v); ↑1

vs_{p2}: L ::= L B

↓L[2],1; L[1].lg=ADD(L[2].lg,1); ↑1

L[2].s=ADD(L[1].s,1); ↓L[2],2; B.s=L[1].s; ↓B,1;

L[1].v=ADD(L[2].v, B.v); ↑2

vs_{p3}: L ::= B

L.lg=1; ↑1

B.s=L.s; ↓B,1; L.v=B.v; ↑2

vs_{p4}: B ::= '0'

B.v=0; ↑1

vs_{p5}: B ::= '1'

B.v=Power2(B.s); ↑1

Implementierung:

Prozedur vs<i><p> für jeden Abschnitt von vs_p bis ↑i

Aufruf mit Verzweigung für ↓X,i

Vorlesung Übersetzer WS 97/98 / Folie 49a

Ziele:

- Beispiel zu U-49

in der Vorlesung:

- Baumdurchlauf zeigen

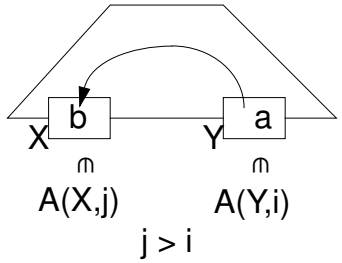
Verständnisfragen:

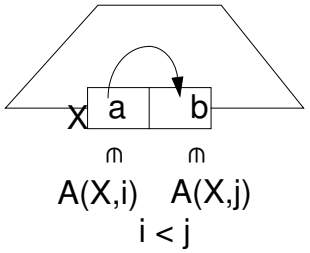
- Geben Sie an, welchen Bedingungen die Besuche in den Besuchssequenzen genügen müssen, damit sie korrekt verzahnen.
- Geben Sie an, welchen Bedingungen die Berechnungen in den Besuchssequenzen genügen müssen, damit die Zerlegungen eingehalten werden.
- Geben Sie an, welchen Bedingungen die Berechnungen in den Besuchssequenzen genügen müssen, damit die Abhängigkeiten eingehalten werden.

LAG (k)-Bedingung

Für alle X gibt es eine Zerlegung A (X,1), ..., A (X, k), so daß A (X, i) im i-ten links-abwärts-Durchgang berechnet werden können.

Notwendige und hinreichende Bedingung über Abhängigkeitsgraphen zu Produktionen:





Algorithmus: berechnet A (1), ..., A (k), falls existiert:

für i = 1, 2, ...
  A (i) := alle noch nicht zugeordneten Attribute
  streiche Attribute solange folgende Regeln anwendbar:

- **streiche X.b, wenn es in einem Kontext von einem Attribut aus A (i) nach obigem Muster abhängt**
- **streiche Z.c, wenn es von einem gestrichenen Attribut abhängt**

falls alle Attribute zugeordent: **AG ist LAG (k)**, k = i
falls A (i) leer: **AG ist nicht LAG (k)**

Vorlesung Übersetzer WS 97/98 / Folie 50

Ziele:

- Einfache Bedingung für links-abwärts Durchläufe verstehen.
- Algorithmus dazu.

in der Vorlesung:

- Bedingung motivieren.
- Algorithmus am Beispiel.

nachlesen:

Kastens / Übersetzerbau, Abschnitt 5.2.3

Übungsaufgaben:

- LAG (k)-Bedingung für AGn prüfen. Aufgabe 24

Generatoren für AGn

LIGA	Universität Paderborn, in Eli	OAG
FNC-2	INRIA	ANCAG
Synthesizer Generator	Cornell University	OAG, inkrementell
CoCo	Universität Linz	LAG(1)

Eigenschaften von LIGA

- **höhere Spezifikationssprache Lido**
- **Modularisierung, Wiederverwendung von AG-Komponenten**
- **Objekt-orientierte Konstrukte zur Abstraktion von Berechnungsmustern**
- **Berechnungen: Funktionsaufrufe; implementiert außerhalb der AG**
- **auch Abhängigkeiten für Seiteneffekt-Berechnungen**
- **Notationen für Zugriff auf entfernte Attribute**
- **generiert Besuchssequenz-gesteuerte Attributauswerter in C**
- **Speicheroptimierung für Attribute**

Vorlesung Übersetzer WS 97/98 / Folie 51

Ziele:

- Hinweis auf einige Generatoren
- einige Eigenschaften von LIGA

nachlesen:

Kastens / Übersetzerbau, Abschnitt 5.4