4. Berechnungen im Strukturbaum
Übersicht

Berechnungen im Srukturbaum für beliebige Zwecke, z. B.

- **Eigenschaften** der Eingabestrukturen **berechnen, prüfen**, z. B. Typen, Wertebereiche
- **Zusammenhänge** bestimmen, prüfen; z. B. Definition - Anwendung
- **Datenstrukturen**, **Zieltext** konstruieren

Modell: Attributierte Grammatik, Spezifikationssprache **Lido**, Generator **Liga**:

Abstrakte Syntax wird erweitert:

Attribute zu Nichtterminalsymbolen:
Daten zu den Baumknoten, z. B. Expr.Value Expr.Type TypeName.ClassFwd

zum Speichern von **Werten an den Knoten**, Propagieren von Werten durch den Baum, Spezifizieren von Abhängigkeiten zwischen Berechnungen

Berechnungen zu Produktionen (RULE) oder Nichtterminalsymbolen (SYMBOL):
Berechnen Attributwerte aus anderen **Attributen im Kontext** (RULE oder SYMBOL), oder bewirken Effekte, z. B. Prüfungen, Meldungen, Ausgabe

Jede Berechnung wird genau einmal für jeden Knoten dieses Typs augeführt.
An jedem Knoten wird jedes seiner Attribute genau einmal berechnet.
Die **Reihenfolge der Berechnungen und den Baumdurchlauf** ermittelt der **Generator** aus den Abhängigkeiten zwischen den Attributen.

Vorlesung Generierung von Software aus Spezifikationen WS 2002 / Folie 401

Ziele:

Grundbegriffe und Fakten

in der Vorlesung:

Themen der Folie erläutern (Beispiele folgen):

- Zwecke,
- Attributierte Grammatiken,
- Attribute und Berechnungen zur abstrakten Syntax,
- Auswertungsmodell

Abhängige Berechnungen

```
SYMBOL Expr, Opr: value: int SYNT;
SYMBOL Opr: left, right: int INH;
TERM Number: int;
```

´typisierte Attribute zu Symbolen

Terminalsymbol mit int-Wert

```
RULE: Root ::= Expr COMPUTE
   printf ("value is %d\n", Expr.value);
END;
```

SYNThesized Attribute werden im unteren Kontext berechnet, INHerited Attribute im oberen.

```
RULE: Expr ::= Number COMPUTE
   Expr.value = Number;
END;
```

Angabe ist meist optional.

```
RULE: Expr ::= Expr Opr Expr COMPUTE
   Expr[1].value = Opr.value;
   Opr.left = Expr[2].value;
   Opr.right = Expr[3].value;
END;
```

Reihenfolge der Berechnungen wird vom Generator aus den Abhängigkeiten bestimmt

```
RULE: Opr ::= '+' COMPUTE
   Opr.value = ADD (Opr.left, Opr.right);
END;
RULE: Opr ::= '-' COMPUTE
   Opr.value = SUB (Opr.left, Opr.right);
END;
```

Beispiel:

Berechnung und Ausgabe des Wertes eines Ausdruckes

Vorlesung Generierung von Software aus Spezifikationen WS 2002 / Folie 402

Ziele:

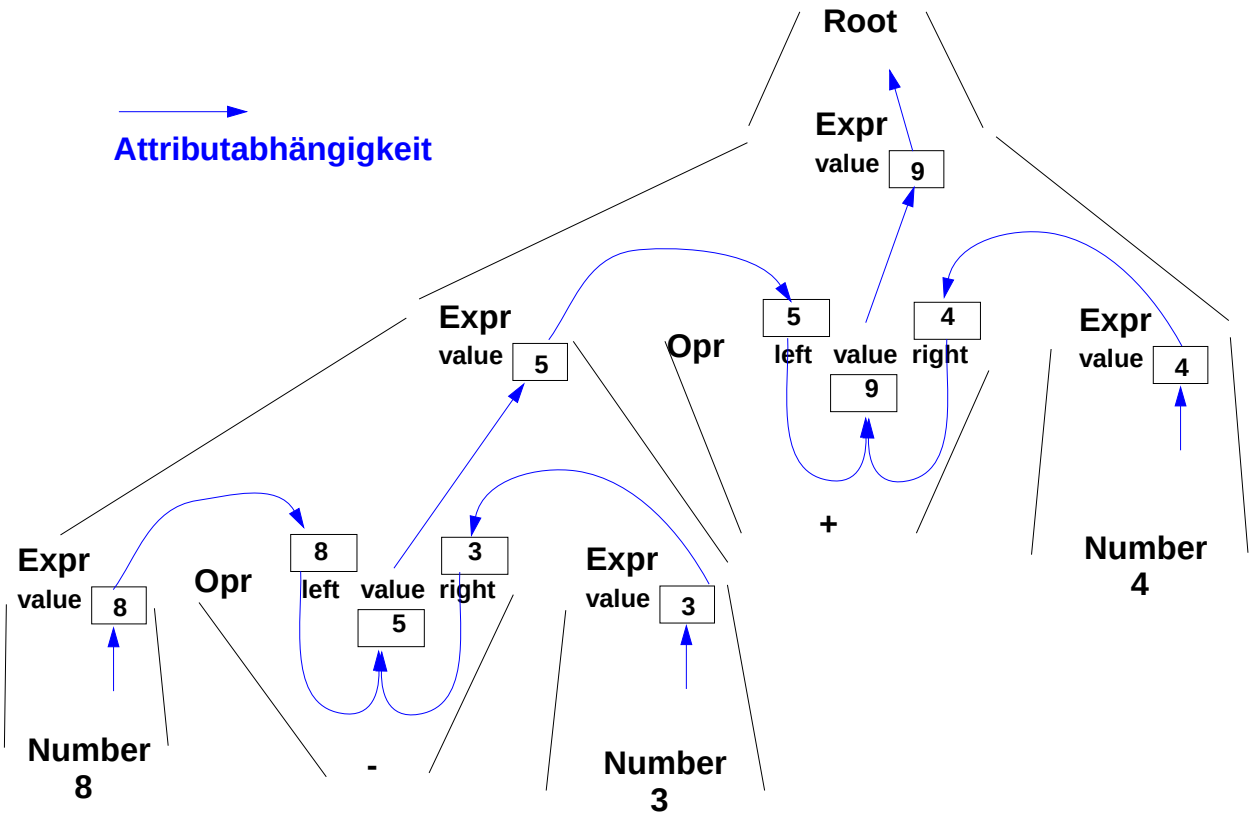
Lido-Notation einführen

in der Vorlesung:

Notation am Beispiel erläutern:

- typisierte Attribute,
- Berechnung mit Seiten-Effekt (print),
- Attributberechnungen,
- Berechnungsreihenfolge aus Abhängigkeiten,
- SYNT und INH Attribute

Ein attributierter Strukturbaum



Vorlesung Generierung von Software aus Spezifikationen WS 2002 / Folie 403

Ziele:

Attributwerte und -abhängigkeiten

in der Vorlesung:

Bild erläutern:

- RULE-Kontexte,
- Berechnungen darin,
- Pfeile zu Attributberechnungen,
- ein passender Baumdurchlauf.

Vor- und Nachbedingungen von Berechnungen

```
RULE: Root ::= Expr COMPUTE
   Expr.print = "yes";
   printf ("n") <- Expr.printed;
END;

RULE: Expr ::= Number COMPUTE
   Expr.printed =
      printf ("%d ", Number) <-Expr.print;
END;

RULE: Expr ::= Expr Opr Expr COMPUTE
   Expr[2].print = Expr[1].print;
   Expr[3].print = Expr[2].printed;
   Opr.print = Expr[3].printed;
   Expr[1].printed = Opr.printed;
END;

RULE: Opr ::= '+' COMPUTE
   Opr.printed =
      printf ("+ ") <- Opr.print;
END;
```

Attribute **`print`** und **`printed`** haben **keine Werte** (Typ **`VOID`**)

Sie beschreiben die Zustände als **Vor- und Nachbedingungen** der Berechnungen:

`Expr.print:`

Postfix-Ausgabe bis vor diesem Knoten ist erledigt

`Expr.printed:`

Postfix-Ausgabe bis einschließlich dieses Knotens ist erledigt

Beispiel:

Ausgabe des Ausdrucks in Postfix-Form

Vorlesung Generierung von Software aus Spezifikationen WS 2002 / Folie 404

Ziele:

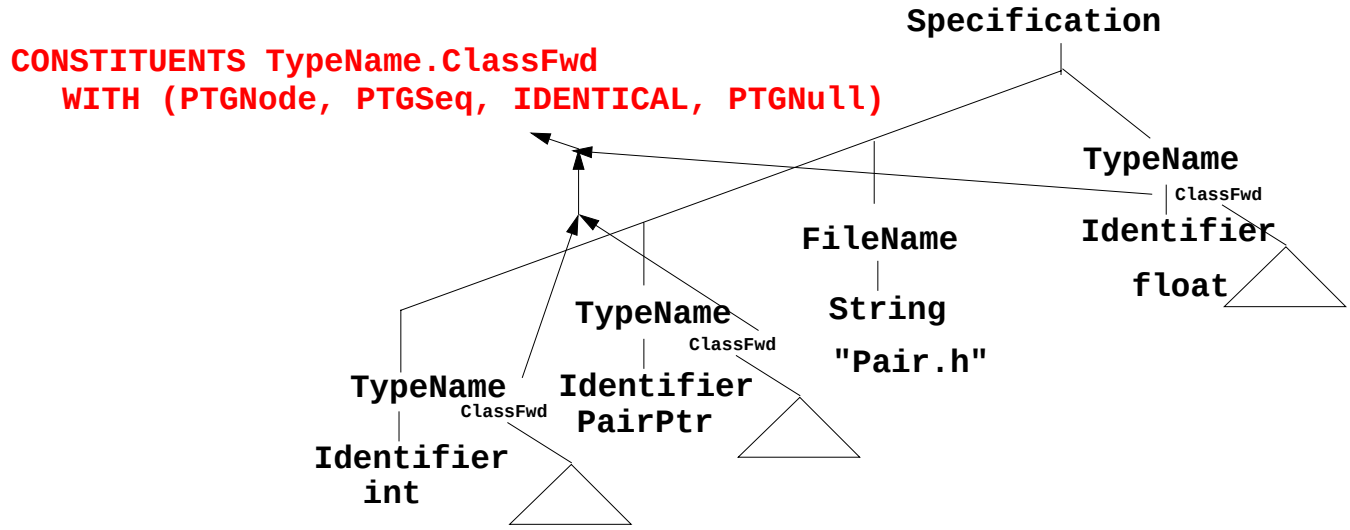
Spezifikation der Reihenfolge

in der Vorlesung:

Erläutern:

- Postfix-Ausgabe,
- Bedeutung und Verwendung der Attribute print und printed

Schema: Attributwerte aus Unterbaum zusammenfassen



CONSTITUENTS fasst Attribute aus Unterbaum zusammen, hier **TypeName.ClassFwd**

	WITH (PTGNode,	**PTGSeq,**	**IDENTICAL,**	**PTGNull)**
Bedeutung:	Typ	2-stellige Verknüp-fungs-funktion	1-stellige Funktion, auf jedes Attribut angewandt	0-stellige Funktion für optionale Teilbäume

Vorlesung Generierung von Software aus Spezifikationen WS 2002 / Folie 405

Ziele:

CONSTITUENTS wiederholen

in der Vorlesung:

- Zusammenfassung der Werte erläutern.
- Die 2-stellige Funktion muss assoziativ sein,
- Die 0-stellige Funktion muss neutral bzgl. der 2-stelligen sein.

Verständnisfragen:

Welche Attribute und Berechnungen müssen Sie einfügen, wenn Sie das Konstrukt ersetzen wollten?

Schema: Attribut eines entfernten Oberknotens benutzen

```
SYMBOL Block: depth: int;

RULE: Root ::= Block COMPUTE
   Block.depth = 0;
END;

RULE: Block ::= '(' Sequence ')' END;
RULE: Sequence LISTOF
        Definition / Statement END;
...

RULE: Statement ::= Block COMPUTE
   Block.depth =
      ADD (INCLUDING Block.depth, 1);
END;

TERM Ident: int;

RULE: Definition ::= 'define' Ident
COMPUTE
   printf("%s defined on depth %d\n",
      StringTable (Ident),
      INCLUDING Block.depth);
END;
```

Beispiel:

Berechnung der Tiefe geschachtelter Blöcke

`INCLUDING Block.depth` bezieht sich auf das **`depth`**-Attribut des nächsten Oberknotens **`Block`**

Das **`INCLUDING`**-Attribut wird **automatisch durch die Kontexte** zwischen seiner Definition im Oberknoten und seiner Benutzung im **`INCLUDING`**-Konstrukt **transportiert**.

Vorlesung Generierung von Software aus Spezifikationen WS 2002 / Folie 406

Ziele:

Benutzung des INCLUDING-Konstrukts

in der Vorlesung:

- Bedeutung erläutern,
- typische Anwendungen zeigen

Verständnisfragen:

Welche Attribute und Berechnungen müssen Sie einfügen, wenn Sie das Konstrukt ersetzen wollten?

Schema: Vorbedingungen aus Unterbaum zusammenfassen

```
SYMBOL Block: depth: int;

RULE: Root ::= Block END;

RULE: Block ::= '(' Sequence ')'
COMPUTE
   Block.DefDone =
      CONSTITUENTS Definition.DefDone;
END;

...

RULE: Definition ::= 'define' Ident
COMPUTE
   Definition.DefDone =
   printf("%s defined in line %d\n",
      StringTable (Ident), LINE);
END;

RULE: Statement ::= 'use' Ident
COMPUTE
   printf("%s used in line %d\n",
      StringTable (Ident), LINE)
      <- INCLUDING Block.DefDone;
END;
```

Beispiel:

Erst alle Definitionen dann alle Anwendungen ausgeben

Die Attribute **DefDone** haben **keine Werte**, sondern sie spezifizieren **nur Abhängigkeiten**

Das **CONSTITUENTS**-Konstrukt brucht **keine WITH-Klausel**, da **keine Werte transportiert** werden.

Typische Kombination von **CONSTITUENTS**-Konstrukt und **INCLUDING**-Konstrukt, um die Reihenfolge der Seiteneffekte zu spezifizieren.

Vorlesung Generierung von Software aus Spezifikationen WS 2002 / Folie 407

Ziele:

Benutzung des CONSTITUENTS-Konstrukts

in der Vorlesung:

- Bedeutung erläutern,
- typische Anwendungen zeigen

Verständnisfragen:

Welche Attribute und Berechnungen müssen Sie einfügen, wenn Sie das Konstrukt ersetzen wollten?

Schema: Abhängigkeiten links-abwärts durch den Baum

```
CHAIN print: VOID;

RULE: Root ::= Expr COMPUTE
   CHAINSTART HEAD.print = "yes";
   printf ("n") <- TAIL.print;
END;

RULE: Expr ::= Number COMPUTE
   Expr.print =
      printf ("%d ", Number) <-Expr.print;
END;

RULE: Expr ::= Expr Opr Expr COMPUTE
   Opr.print = Expr[3].print;
   Expr[1].print = Opr.print;
END;

RULE: Opr ::= '+' COMPUTE
   Opr.print =
      printf ("+ ") <- Opr.print;
END;
```

CHAIN spezifiziert **left-right depth-first** Abhängigkeit.

CHAINSTART im **Wurzel-Kontext der CHAIN**

Berechnungen werden mit Vor- und Nachbedingungen **in die CHAIN „eingehängt“**

CHAIN-Reihenfolge kann **überschrieben** werden.

Weggelassene CHAIN-Berechnungen werden **automatisch ergänzt**

Beispiel:

Ausgabe des Ausdrucks in Postfix-Form (vgl. GSS-4.4)

Vorlesung Generierung von Software aus Spezifikationen WS 2002 / Folie 408

Ziele:

Benutzung des CHAIN-Konstrukts

in der Vorlesung:

- Bedeutung erläutern,
- typische Anwendungen zeigen.
- Realisierung durch ein Paar von Attributen an jedem Symbol, durch das die CHAIN führt - eines INH und eines SYNT.

Verständnisfragen:

Welche Attribute und Berechnungen müssen Sie einfügen, wenn Sie das Konstrukt ersetzen wollten?

Berechnungen zu Symbolen

Berechnungen können **Symbolen zugeordnet** werden, wenn sie **für jedes Auftreten** des Symbols in Produktionen **gleich** sind.

```
SYMBOL Expr COMPUTE
   printf ("expression value %d in line %d\n", THIS.value, LINE);
END;
```

Symbol-Berechnungen können **INCLUDING**-, **CONSTITUENTS**-, **CHAIN**-Konstrukte enthalten:

```
SYMBOL Block COMPUTE
   printf ("%d uses occurred\n",
      CONSTITUENTS Usage.Count WITH (int, ADD, IDENTICAL, ZERO);
END;
```

SYNT.a bzw. **INH.a** gibt an, dass die Berechnung zum **unteren bzw. oberen Kontext** des Symbols gehört:

```
SYMBOL Block COMPUTE
   INH.depth = ADD (INCLUDING Block.depth);
END;
```

Berechnungen im **RULE-Kontext überschreiben Berechnungen im SYMBOL-Kontext**, z. B. für Rekursionsanfänge, Ausnahmen, oder Defaults:

```
RULE: Root ::= Block COMPUTE
   Block.depth = 0;
END;
```


Vorlesung Generierung von Software aus Spezifikationen WS 2002 / Folie 409

Ziele:

Prinzip der SYMBOL-Berechnungen erkennen

in der Vorlesung:

SYMBOL-Berechnungen an den Beispielen der Folie erläutern:

- THIS, SYNT, INH stehen in Berechnungen für das betreffende Symbol.
- In SYMBOL-Berechnungen kann nicht auf Attribute von Symbolen aus einem Regelkontext Bezug genommen werden.

Wiederverwendung von Berechnungen

```
CLASS SYMBOL IdOcc: Sym: int;
CLASS SYMBOL IdOcc COMPUTE
   SYNT.Sym = TERM;
END;
```

Berechnungen werden neuen **CLASS**-Symbolen zugeordnet.

```
SYMBOL DefIdent INHERITS IdOcc END;
SYMBOL UseIdent INHERITS IdOcc END;
```

Durch **INHERITS** werden sie an Baum-Symbole gebunden

```
CLASS SYMBOL OccRoot COMPUTE
   CHAINSTART HEAD.Occurs = 0;
   SYNT.TotalOccs = TAIL.Occurs;
END;
CLASS SYMBOL OccElem COMPUTE
   SYNT.OccNo = THIS.Occurs;
   THIS.Occurs = ADD (SYNT.OccNo, 1);
END;
```

Zusammenwirkende CLASS-Symbole, z. B. Auftreten eines syntaktischen Konstruktes zählen

```
SYMBOL Block      INHERITS OccRoot END;
SYMBOL Definition INHERITS OccElem END;

SYMBOL Statement INHERITS OccRoot END;
SYMBOL Usage     INHERITS OccElem END;
```

Paarweise Wiederverwendung

Auf diesem Prinzip beruht die Benutzung von Bibliotheksmodulen, z. B. zur Namensanalyse (siehe Kap. 6)

Vorlesung Generierung von Software aus Spezifikationen WS 2002 / Folie 410

Ziele:

Abstrakte Symbol-Rollen

in der Vorlesung:

- Notation und Beispiele erläutern.
- Hinweis auf Bibliothek von Spezifikationen.

Richtlinien zum Entwurf von Berechnungen im Baum

1. Zerlege die Aufgabe in **Teilaufgaben**, die klein genug sind, um durch wenige der u. g. Spezifikationsmuster gelöst zu werden.
   Entwickle für jede Teilaufgabe ein `.lido`-Fragment und erläutere es im umgebenden `.fw`-Text.

2. Arbeite den **zentralen Aspekt der Teilaufgabe** heraus und bilde ihn auf einen der folgenden Fälle ab:

   A. Der Aspekt wird in natürlicher Weise durch **Eigenschaften von einigen Programmkonstrukten** beschrieben,
   z. B. Typen von Ausdrücken, Schachtelungstiefe von Blöcken, Übersetzung der Anweisungen eines Blockes.

   B. Der Aspekt wird in natürlicher Weise durch **Eigenschaften von Programmobjekten** beschrieben, z. B. Relativadresse von Variablen, Benutzung von Variablen vor der Definition.

   Entwirf die Berechnung wie für A oder B beschrieben.

3. Im Schritt 2 können Anforderungen an **weitere Aspekte** der Teilaufgabe entstehen (benutzte Attribute, deren Berechnung noch nicht entworfen ist): Wiederhole Schritt 2 für diese.

Ziele:

Hilfen zum systematischen Entwurf

in der Vorlesung:

An Beispielen erläutern

A: Eigenschaften von Programmkonstrukten

Bestimme den **Typ der Werte**, die die Eigenschaft beschreiben. Führe **Attribute dieses Typs zu allen Symbolen** ein, die die **Programmkonstrukte** repräsentieren. Prüfe, welcher der folgenden Fälle für die Bestimmung der Eigenschaft am besten zutrifft.

A1: Jeder **untere Kontext** bestimmt die Eigenschaft in unterschiedlicher Weise:
Entwirf **RULE-Berechnungen**.

A2: Wie A1; aber **oberer Kontext**.

A3: Die Eigenschaft kann **unabhängig von den RULE-Kontexten** berechnet werden, wobei nur Attribute des Symbols oder über INCLUDING, CONSTITUENT(S), CHAIN erreichbare Attribute verwendet werden:
Entwirf eine **untere (SYNT) SYMBOL-Berechnung**.

A4: Wie A3. Aber es gibt **wenige Ausnahmen**, wo entweder untere oder obere (nicht beide!) RULE-Kontexte die Eigenschaft abweichend bestimmen:
Entwirf eine obere (INH) oder untere (SYNT) **SYMBOL-Berechnung** und **überschreibe sie in solchen RULE-Kontexten**.

A5: Wie A4; aber für **rekursive Symbole**: Die Fundierung der Rekursion ist die Ausnahme aus A4, z. B. Schachtelungstiefe von Blöcken.

Wenn keiner der Fälle passt, muss die Modellierung der Eigenschaft überdacht werden. Sie könnte zu komplex sein und eine weitere Zerlegung erfordern.

Ziele:

Hilfen zum Platzieren von Berechnungen

in der Vorlesung:

An Beispielen erläutern: