Attributierte Grammatiken

- Form:
  - kontextfreie Produktionsregeln
  - Attribute: eine Reihe von Variablen für jedes Grammatiksymbol
  - für jede Produktion:
    - *Auswertungsregeln*: bestimmen Berechnung der Attribute
    - *Bedingungen*: bestimmen die Gültigkeit der Produktion
- Zwei Anwendungen: (werden oft kombiniert)
  - Einschränkungen, die eine kontextfreie in eine kontextsensitive Grammatik überführen
    (durch das Verbot bestimmter Expansionen/Reduktionen)
  - Anreicherung einer Grammatik mit semantischer Information
    (zur semantischen Überprüfung und Codegenerierung)
- Attributarten: (Abb. 3.1)
  - synthetisiert: rechte Seite einer Produktion bestimmt linke Seite (bottom-up)
  - ererbt: linke Seite bestimmt rechte Seite (top-down)

Datenfluss zwischen Attributen

Bsp.: A→BCD

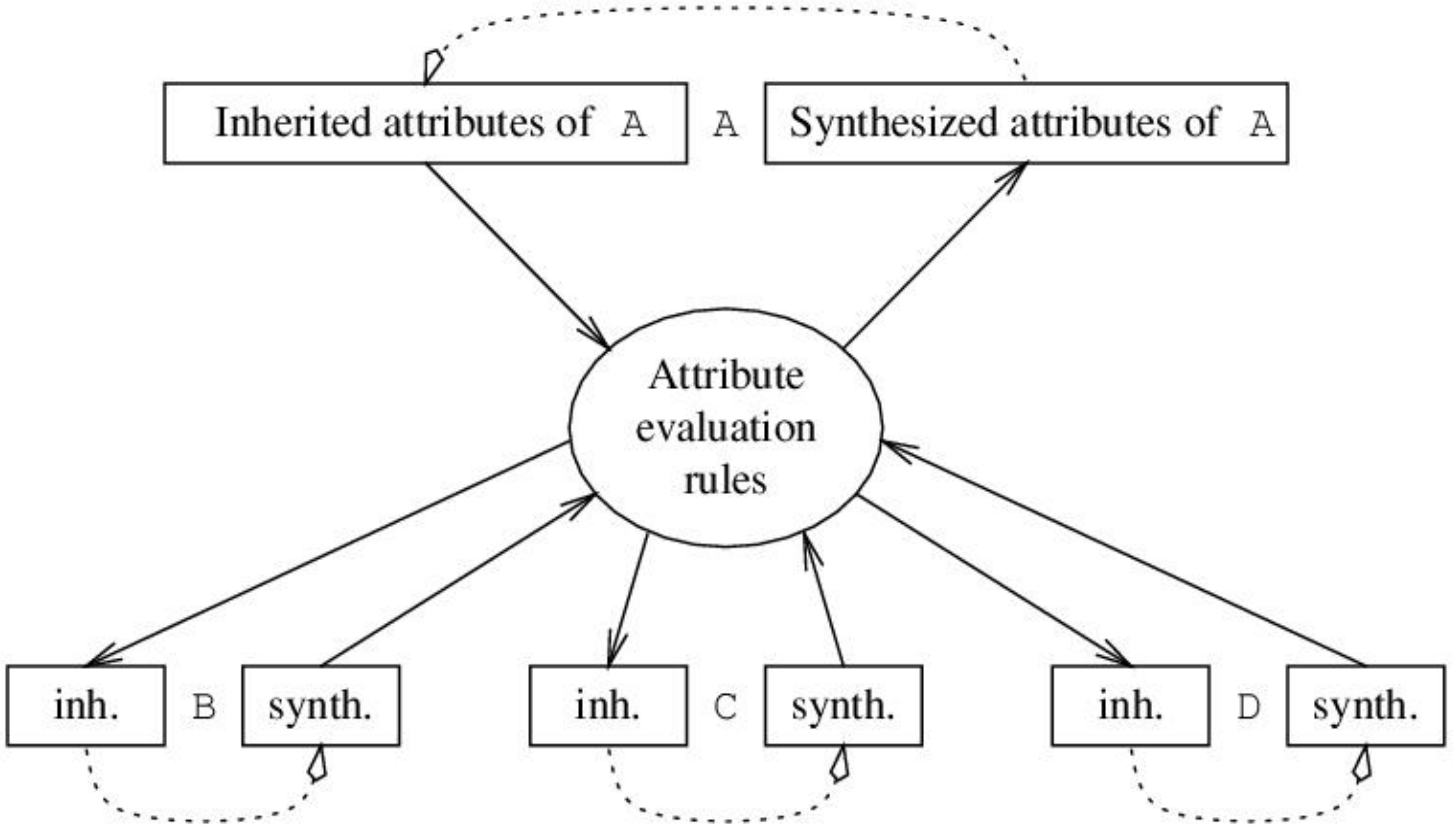

Figure 3.1 Data flow in a node with attributes.

Einführende Beispiele (1)

- Sprache: $\{x \in \mathbb{N} \mid x < 10^8\}$
- Grammatik:

Produktionen	Auswertungsregeln	Bedingungen
$\mathtt{Ds} \rightarrow \mathtt{Ds}_1\ \mathtt{D}$	$v(\mathtt{Ds}) := 10 \cdot v(\mathtt{Ds}_1) + v(\mathtt{D})$	$v(\mathtt{Ds}) < 10^8$
$\mathtt{Ds} \rightarrow \mathtt{D}$	$v(\mathtt{Ds}) := v(\mathtt{D})$	
$\mathtt{D} \rightarrow 0$	$v(\mathtt{D}) := 0$	
$\vdots$	$\vdots$	
$\mathtt{D} \rightarrow 9$	$v(\mathtt{D}) := 9$	

Einführende Beispiele (2)

- Sprache: $\{x^n y^n z^n \mid n>0\}$
- Grammatik: nur mit synthetisierten Attributen (S-attributiert)

Produktionen	Auswertungsregeln	Bedingungen
S → X Y Z		$s(X)=s(Y)=s(Z)$
X → x X_1	$s(X) := 1+s(X_1)$	
X → x	$s(X) := 1$	
Y → y Y_1	$s(Y) := 1+s(Y_1)$	
Y → y	$s(Y) := 1$	
Z → z Z_1	$s(Z) := 1+s(Z_1)$	
Z → z	$s(Z) := 1$	

- Fehlererkennung: nachdem der gesamte Parsebaum erstellt ist

Einführende Beispiele (3)

- Sprache: $\{x^n y^n z^n \mid n>0\}$
- Grammatik: auch mit ererbten Attributen (L-attributiert)

Produktionen	Auswertungsregeln	Bedingungen
S → X Y Z	$e(Y) := s(X);$ $e(Z) := s(X)$	
X → x X_1	$s(X) := 1+s(X_1)$	
X → x	$s(X) := 1$	
Y → y Y_1	$e(Y_1) := e(Y)-1$	$e(Y)>1$
Y → y		$e(Y)=1$
Z → z Z_1	$e(Z_1) := e(Z)-1$	$e(Z)>1$
Z → z		$e(Z)=1$

- Fehlererkennung: bevor der gesamte Parsebaum erstellt ist

Attributauswertung

- Aufgabe eines Auswerters:
  - Speicherallokation für die Attribute (im Syntaxbaum)
  - Belegung der Terminalknoten im Baum mit ihren Attributen
  - Propagation und Berechnung von Attributwerten entlang der Abhängigkeiten im Baum
  - Identifikation unauswertbarer Attribute (Inkonsistenzen, Zyklen)
- Vorgehensweisen:

  (A) Auswertung während des Parsens
  - Voraussetzung: S- oder L-Attributierung
  - Auswertungsreihenfolge statisch in den Compiler kodiert

  (B) Auswertung nach dem Parsen
  - Voraussetzung: zyklenfreie Attributabhängigkeiten
  - Auswertungsreihenfolge statisch oder dynamisch
- Implementierungsvarianten:
  - einfache Ausdruckssprache zur Kombination von Attributwerten
  - spezielle oder allgemeine Programmiersprache

Beispiel: Konstantendefinition

```
Constant_definition (INH old symbol table, SYN new symbol table) →
    'CONST'  Defined_identifier  '='  Expression  ';'
    ATTRIBUTE RULES:
        SET Expression .symbol table TO
            Constant_definition .old symbol table;
        SET Constant_definition .new symbol table TO
            Updated symbol table (
                Constant_definition .old symbol table,
                Defined_identifier .name,
                Checked type of Constant_definition (Expression .type),
                Expression .value
            );

Defined_identifier (SYN name) → ...

Expression (INH symbol table, SYN type, SYN value) → ...
```

Figure 3.2 A simple attribute rule for `Constant_definition`.

Beispiel: Konstantendefinition

- Attributierte Regel: (Abb. 3.2)
  - ererbtes Attribut: alte Symboltabelle
  - synthetisiertes Attribut: neue, erweiterte Symboltabelle
  - einzige Produktion:

    `'CONST' Defined_identifier '=' Expression ';'`
  - Auswertungsregeln:
    - vererbe die alte Symboltabelle an **`Expression`**
    - erweitere die alte zur neuen Symboltabelle
      (Eintrag: Typ und Wert des Ausdrucks, Name der Konstanten)
- Anmerkungen:
  - textuelle Reihenfolge der Auswertungsregeln beliebig
  - einziges Attribut von **`Defined_identifier`**: **`name`**

    (**`Identifier`** hat auch Art, Typ, Wert, Scope, etc.)
  - Kontextprüfung des Ausdruckstyps, keine einfache Übernahme!
    (Abfangen eines möglicherweise unpassenden Attributwertes)
- Abhängigkeiten: (Abb. 3.3)

Konstantendefinition/Abhängigkeitsgraph

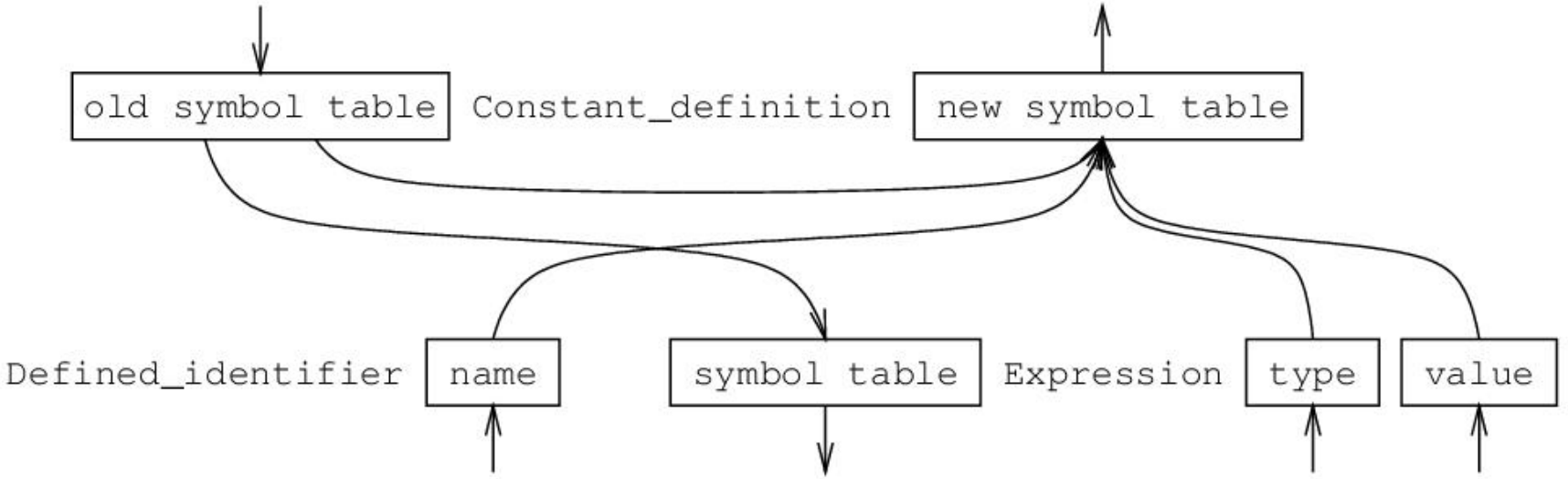

Figure 3.3 Dependency graph of the rule for `Constant_definition` from Figure 3.1.

Regel für `Expression → Number`

```
Expression (INH symbol table, SYN type, SYN value) →
    Number
    ATTRIBUTE RULES:
        SET Expression .type TO Number .type;
        SET Expression .value TO Number .value;
```

Figure 3.4 Trivial attribute grammar for `Expression`.

Datenflussgraph im Detail

Bsp.: `CONST pi = 3.14159265;`

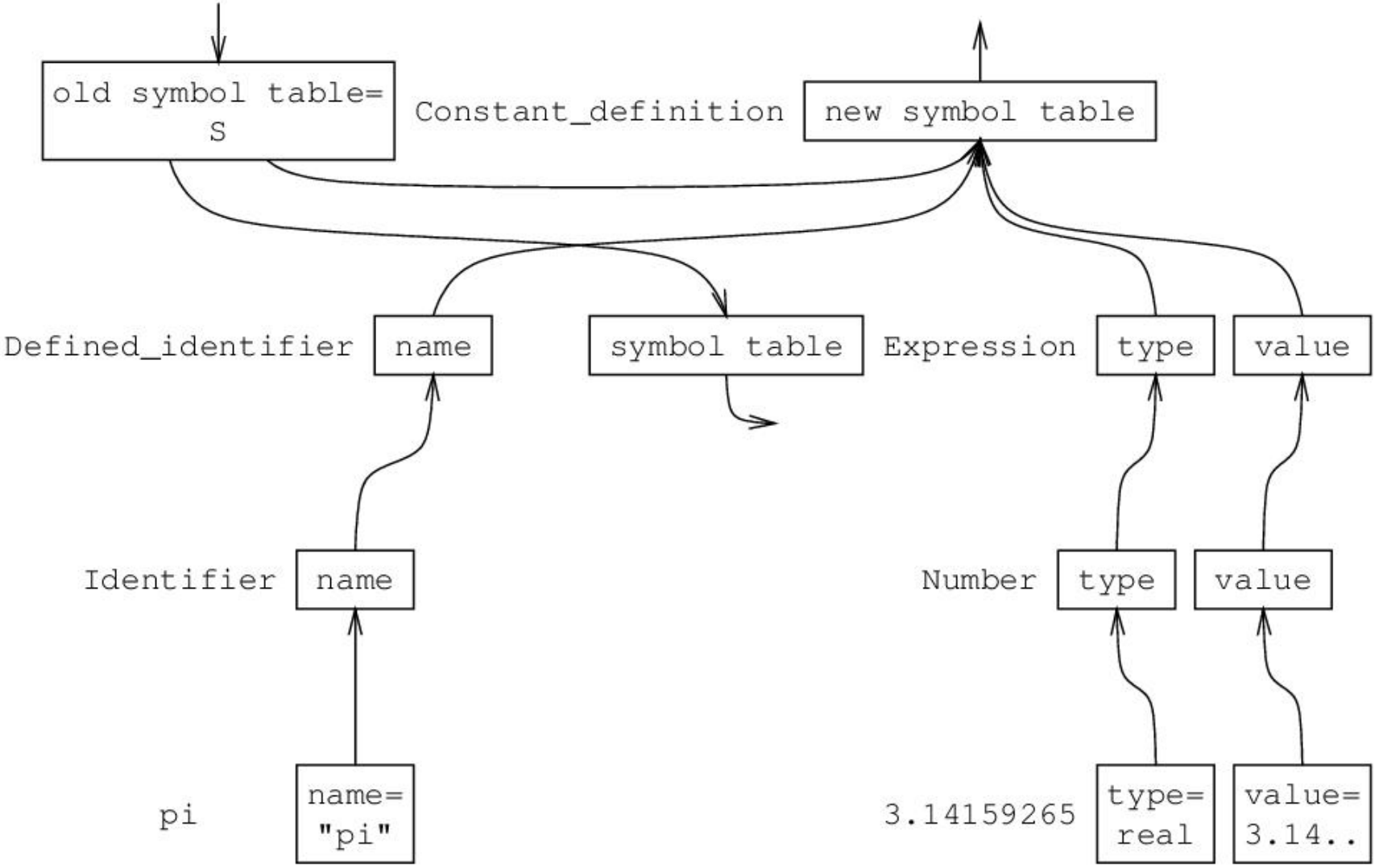

Figure 3.5 Sample attributed syntax tree with data flow.

Datenflussgraph nach Attributauswertung

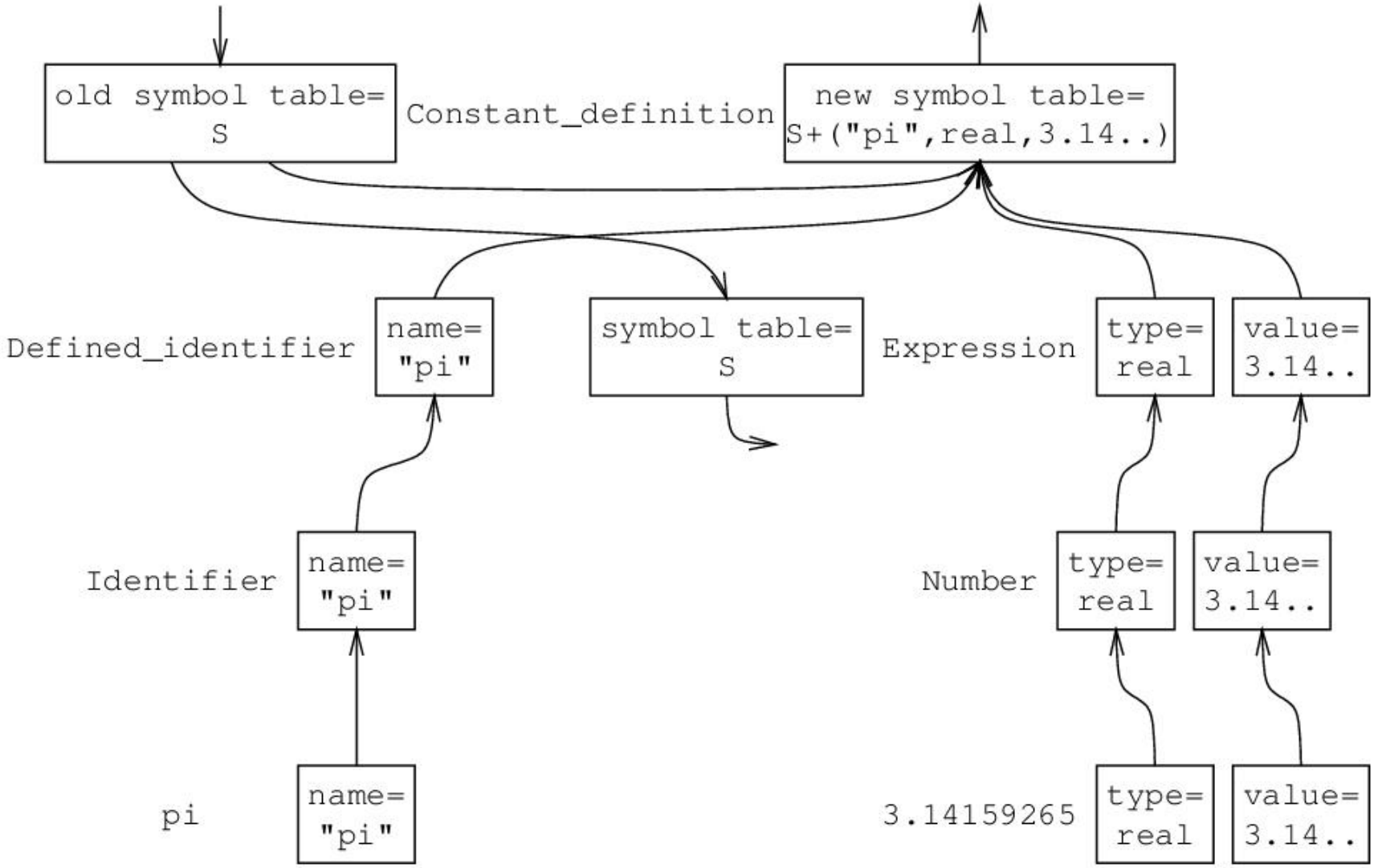

Figure 3.6 The attributed syntax tree from Figure 3.5 after attribute evaluation.

Dynamische Attributauswertung

- Bsp.: Oktal- und Dezimalkonstanten (**17D=21B**, **18B**: Fehler)

```
Number     → Digit_Seq Base_Tag
Digit_Seq  → Digit_Seq Digit | Digit
Digit      → Digit_Token   (0...9)
Base_Tag   → 'B' | 'D'
```

- Attributierung:
  - lange Version: (Abb. 3.8)
  - kurze Version: (Regel für **Digit_Seq**)

```
Digit_Seq(INH base, SYN value)
    →  Digit_Seq(base,value) Digit(base,value)
       ATTRIBUTE RULES:
          SET value
          TO Digit_Seq.value * base + Digit.value
    |  Digit(base,value)
```

- Datenfluss (Abb. 3.10-13)

  1. Durchlauf: “Synthetisierung” von **base**
  2. Durchlauf: Vererbung von **base**, Synthetisierung von **value**

```
Number(SYN value) →
    Digit_Seq  Base_Tag
    ATTRIBUTE RULES
        SET Digit_Seq .base TO Base_Tag .base;
        SET Number .value TO Digit_Seq .value;

Digit_Seq(INH base, SYN value) →
    Digit_Seq[1]  Digit
    ATTRIBUTE RULES
        SET Digit_Seq[1] .base TO Digit_Seq .base;
        SET Digit .base TO Digit_Seq .base;
        SET Digit_Seq .value TO
            Digit_Seq[1] .value * Digit_Seq .base + Digit .value;
|
    Digit
    ATTRIBUTE RULES
        SET Digit .base TO Digit_Seq .base;
        SET Digit_Seq .value TO Digit .value;

Digit(INH base, SYN value) →
    Digit_Token
    ATTRIBUTE RULES
        SET Digit .value TO Checked digit value (
            Value_of (Digit_Token .repr [0]) - Value_of ('0'),
            base
        );

Base_Tag(SYN base) →
    'B'
    ATTRIBUTE RULES
        SET Base_Tag .base TO 8;
|
    'D'
    ATTRIBUTE RULES
        SET Base_Tag .base TO 10;
```

Figure 3.8 An attribute grammar for octal and decimal numbers.

Fig. 3.8 zusammengefasst

`Number` → `Digit_Seq Base_Tag`	*base*(`Digit_Seq`) := *base*(`Base_Tag`); *value*(`Number`) := *value*(`Digit_Seq`)

`Digit_Seq` → `Digit_Seq`$_1$ `Digit`	*base*(`Digit_Seq`$_1$) := *base*(`Digit_Seq`);
	base(`Digit`) := *base*(`Digit_Seq`);
	value(`Digit_Seq`) := *value*(`Digit_Seq`$_1$)
	· *base*(`Digit_Seq`) + *value*(`Digit`)

`Digit_Seq` → `Digit`	*base*(`Digit`) := *base*(`Digit_Seq`); *value*(`Digit_Seq`) := *value*(`Digit`)
`Digit` → `Digit_Token`	*value*(`Digit`) := {Konversion und Check}
`Base_Tag` → `'B'`	*base*(`Base_Tag`) := 8
`Base_Tag` → `'D'`	*base*(`Base_Tag`) := 10

Zifferncheck für Zahlenbasis

```
FUNCTION Checked digit value (Token value, Base) RETURNING an integer:
    IF Token value < Base: RETURN Token value;
    ELSE Token value >= Base:
        ERROR "Token " Token value " cannot be a digit in base " Base;
        RETURN Base - 1;
```

Figure 3.9 The function `Checked digit value`.

Abhängigkeitsgraph für `Number`

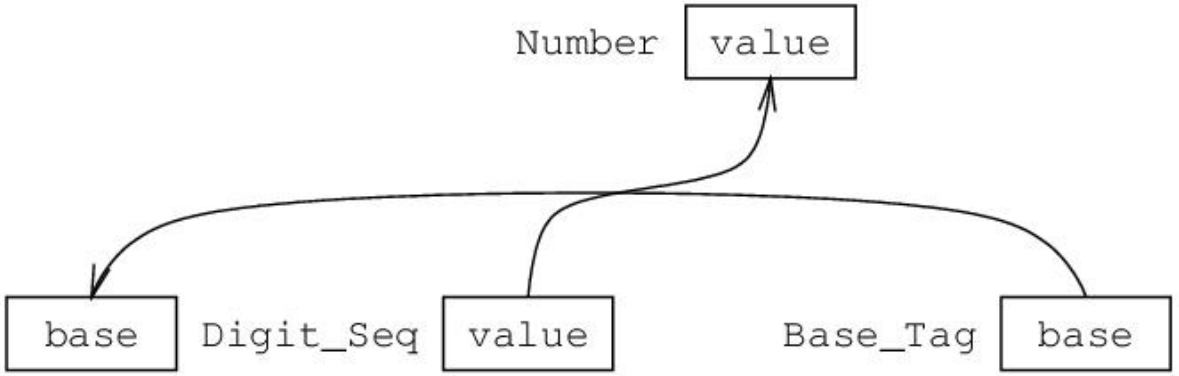

Figure 3.10 The dependency graph of `Number`.

Abhängigkeitsgraphen für `Digit_Seq`

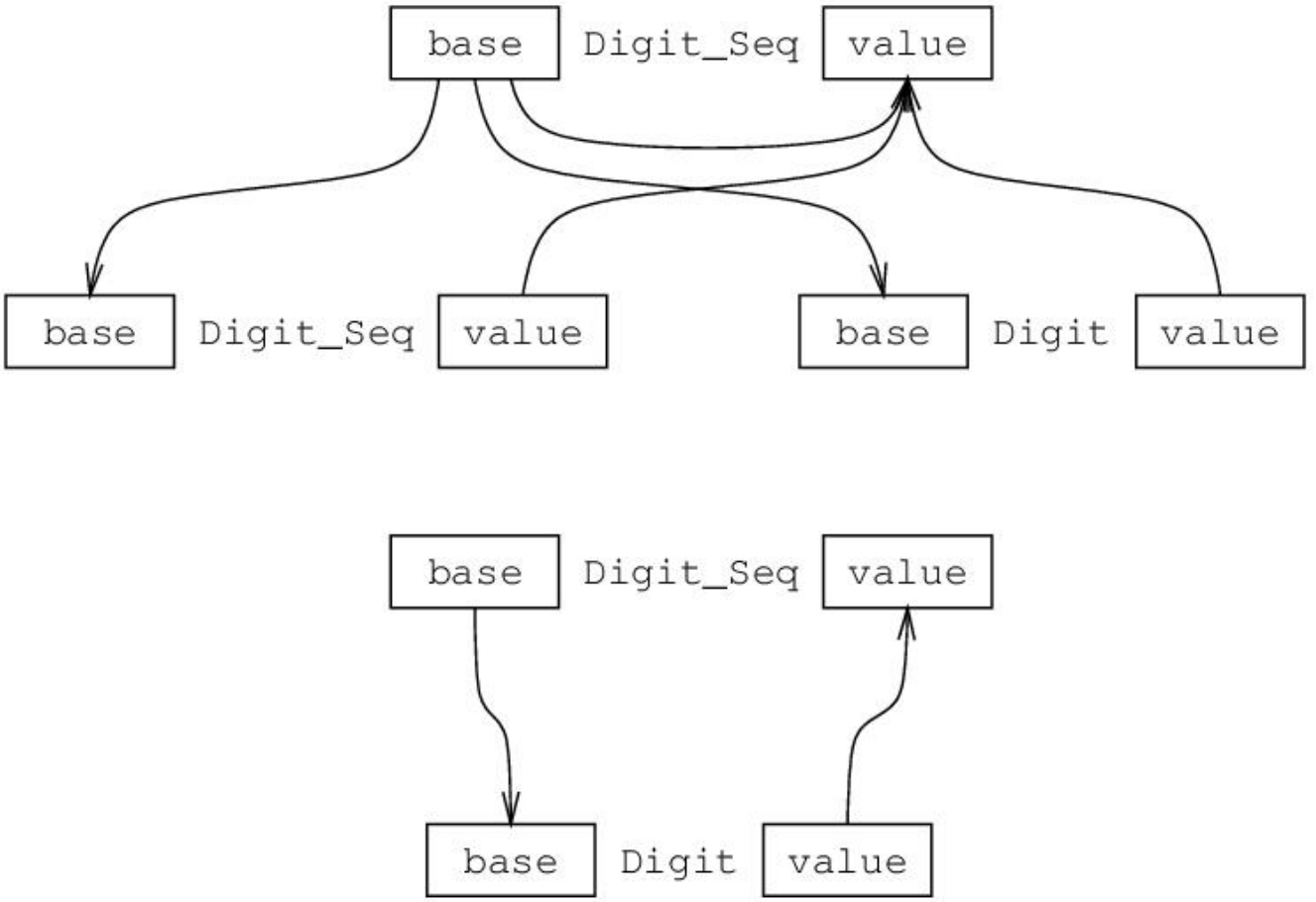

Figure 3.11 The two dependency graphs of `Digit_Seq`.

Weitere Abhängigkeitsgraphen

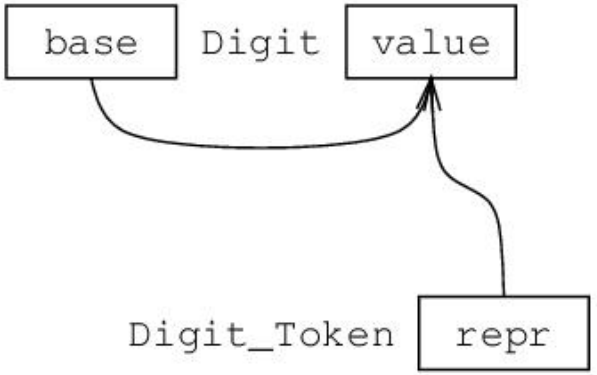

Figure 3.12 The dependency graph of `Digit`.

Figure 3.13 The two dependency graphs of `Base_Tag`.

Code: dynamischer Datenflussauswerter

- Zwei Routinen pro Grammatikregel: (Abb. 3.14)
  - Parameter:
    - Zeiger auf Knoten des linken Nichtterminals
    - Zeiger auf Tupel mit Zeigern auf die Knoten der Symbole der rechten Seite
  - Routine *Propagate*:
    - prüft, wo alle Eingaben besetzt sind, die Ausgaben noch nicht
    - führt entsprechende Auswertungen durch
  - Routine *Evaluate*: zwei Baumdurchläufe
    - propagiert Daten zur Eingabe in die Teilbäume
    - evaluiert die beiden Teilbäume
    - propagiert neue Ausgabewerte der Teilbäume
- Treiber: (Abb. 3.15)
  - macht wiederholt Baumdurchläufe, bis das Resultat ermittelt ist
  - bestätigt jeden Baumdurchlauf
  - gibt das Resultat aus
- Alternative: topologisches Sortieren des Flussgraphen

```
PROCEDURE Evaluate for Digit_Seq alternative_1 (
    pointer to digit_seq node Digit_Seq,
    pointer to digit_seq alt_1 node Digit_Seq alt_1
):
    // Propagate attributes:
    Propagate for Digit_Seq alternative_1 (Digit_Seq, Digit_Seq alt_1);

    // Traverse subtrees:
    Evaluate for Digit_Seq (Digit_Seq alt_1 .Digit_Seq);
    Evaluate for Digit (Digit_Seq alt_1 .Digit);

    // Propagate attributes:
    Propagate for Digit_Seq alternative_1 (Digit_Seq, Digit_Seq alt_1);

PROCEDURE Propagate for Digit_Seq alternative_1 (
    pointer to digit_seq node Digit_Seq,
    pointer to digit_seq alt_1 node Digit_Seq alt_1
):
    IF Digit_Seq alt_1 .Digit_Seq .base is not set
        AND Digit_Seq .base is set:
        SET Digit_Seq alt_1 .Digit_Seq .base TO Digit_Seq .base;

    IF Digit_Seq alt_1 .Digit .base is not set
        AND Digit_Seq .base is set:
        SET Digit_Seq alt_1 .Digit .base TO Digit_Seq .base;

    IF Digit_Seq .value is not set
        AND Digit_Seq alt_1 .Digit_Seq .value is set
        AND Digit_Seq .base is set
        AND Digit_Seq alt_1 .Digit .value is set:
        SET Digit_Seq .value TO
            Digit_Seq alt_1 .Digit_Seq .value * Digit_Seq .base
            + Digit_Seq alt_1 .Digit .value;
```

Figure 3.14 Data-flow code for the first alternative of `Digit_Seq`.

Iterierte Auswertung

```
PROCEDURE Run the data-flow attribute evaluator on node Number:
    WHILE Number .value is not set:
        PRINT "Evaluate for Number called";    // report progress
        Evaluate for Number (Number);

    // Print one attribute:
    PRINT "Number .value = ", Number .value;
```

Figure 3.15 Driver for the data-flow code.

```
Evaluate for Number called
Evaluate for Number called
Number .value = 375
```