Eigenschaften typischer Muster auf geordneten Attributgrammatiken

Christian Berg und Wolf Zimmermann

[1] christian.berg@informatik.uni-halle.de
[2] wolf.zimmermann@informatik.uni-halle.de
Institut für Informatik
Martin-Luther-Universität Halle-Wittenberg

Zusammenfassung. Bei der Implementierung von Spracheigenschaften und statischer Analysen mit Attributgrammatiken treten typische Muster auf. Diese Arbeit stellt typische Muster als Funktion auf Attributgrammatiken vor und untersucht deren Berechenbarkeitseigenschaften. Durch Beweis, dass typische Muster zerlegungserhaltend sind und deren Komposition abgeschlossen ist, wird erreicht, dass nahezu beliebige Attributgrammatiken aus typischen Mustern aufgebaut werden können. Das resultierende Laufzeitverhalten bei der Komposition wird durch die gezeigten Eigenschaften und die Konstruktion von Attributgrammatiken mit typischen Mustern nicht negativ beeinflusst.

1 Einleitung

Die von Knuth in [32] vorgestellten Attributgrammatiken haben sich als geeignet erwiesen Programmiersprachen aber auch Domänen-spezifische Sprachen zu implementieren[36,29,38]. Ein häufiger Kritikpunkt an Attributgrammatiken wurde von Koskimies in [34] wie folgt ausgedrückt:

„The concept of an attribute grammar is too primitive to be nothing but a basic framework, the ‚machine language' of language implementation"

Attributgrammatiken seien also vergleichbar mit Assembler bzgl. der Implementierung von Programmiersprachen. Andererseits existieren mit verschiedenen Erweiterungen, wie Vererbung aus [25], oder Bibliothekssystemen wie [26] und Attributen höherer Ordnung [43], die knappere Beschreibungen ermöglichen. Gemein ist diesen Erweiterungen, dass resultierende Attributgrammatik nicht notwendigerweise berechenbar sein muss. Die Prüfung ob eine beliebige Attributgrammatik berechenbar ist, ist ein schweres Problem[22].

In dieser Arbeit werden typische Muster als Alternative zu eben aufgeführten Erweiterungen vorgestellt. Ausgehend von der Präsentation der Gemeinsamkeiten vieler Spracheigenschaften in Abschnitt 2 und der Vorstellung von Attributgrammatiken und deren Berechenbarkeit in Abschnitt 3 werden typische Muster in Abschnitt 4 eingeführt. Abgeschlossen wird die Arbeit durch den Nachweis der Berechenbarkeit und der Abgeschlossenheit dieser in Abschnitt 5. Eine kurze Zusammenfassung gibt Abschnitt 6.

2 Verwandte Arbeiten

Gegenstand der Arbeiten [3,2,4] ist die Beschreibung einer Domänen-spezifischen Sprache (engl. domain specific language, DSL), die die Steuerung intelligenter Geräte innerhalb eines Gebäudes beschreiben soll. Während in diesen Arbeiten die Semantik nicht formal beschrieben ist, ist eine wichtige Aufgabe die Zuordnung von Aktionen zu Ereignissen auf Basis von Bezeichnern. Die Analyse von Bezeichnern, auch zur Verwendung in der Namensanalyse, wird auch in [13] benötigt bei der Industrieroboter durch eine DSL zur Laufzeit der Programme programmiert werden sollen.

Darüber hinaus existieren aus dem Bereich Domänen-spezifischer Sprachen weitere Beispiele wie [39] – eine Sprache zur Beschreibung der Signalverarbeitung bei der Bitoperationen und Tabellen generiert werden können – oder der Beschreibungssprache für Benutzerschnittstellen in Fahrzeugen aus [20]. Diesen Beispielen und vielen anderen ist gemein, dass eine Namensanalyse zwingend notwendig für weitere Analysen oder die Codegenerierung ist.

Ein häufig in dieser oder ähnlicher Art aufzufindendes Beispiel ist die Codegenerierung oder Analyse auf Basis einer Ausdrucksgrammatik. Beispiel 1.1 (Seite 3 stellt dieses Beispiel unter Ausnutzung der Beschreibung mit Attributgrammatiken vor.

Attributgrammatiken zur Sprachspezifikation wurden von Knuth in [32] eingeführt. Aufgrund der Komplexität bei der Bestimmung der Berechenbarkeit – siehe [22] – dieser sehr allgemeinen Attributgrammatiken wurden eingeschränkte Attributgrammatiken. Eine mögliche Einschränkung stellen Attributgrammatiken mit fester Besuchsstrategie dar, wie diese in [35,23] vorgestellt wurden, oder Attributgrammatiken mit statisch bzw. teilweise statisch bestimmter Berechnungsstrategie, wie diese in [27] oder [31] vorgestellt wurden. Die von Kastens in [27] vorgestellten Attributgrammatiken wurden bereits in vielen Arbeiten und Werkzeugen herangezogen bzw. verwendet[29,30,10,41,28].

Die häufig erwähnte Kritik an Attributgrammatiken – zu hoher Spezifikationsumfang – führte zu einer Reihe von Erweiterungen mit dem Ziel diese Komplexität zu verringern, wie bspw. entfernten Attributen oder verschiedenen Arten der Termersetzung, siehe u. a. [18,11,15,17,40,43]. Boyland beschreibt in [11], dass für entfernte Attribute die Bestimmung der Berechenbarkeit nicht nur schwer (d. h. algorithmisch exponentielle Laufzeit erfordert) sondern sogar unentscheidbar ist. Arbeiten zur Termersetzung, wie bspw. [43,40] lassen sich durch einfache Konventionen innerhalb des Gerüsts der geordneten Attributgrammatiken anwenden, für andere Ansätze, wie [15] ist keine Arbeit bekannt die solche Eigenschaften untersucht.

Einfache Ansätze wie Vererbung oder Modulsysteme sowie einfache Methoden der Textersetzung wie [29,38,25] basieren darauf, dass das Resultat im Grunde vollständig neu betrachtet werden muss, somit sind potentielle Garantien vor Kombination nicht nutzbar.

```
1  -- Namensanalyse
2  rule Root ::= Expr
3  attr Expr.envIn = []
4
5  rule Expr ::= VarDef Expr Expr
6  attr Expr2.envIn = Expr1.envIn
7       Expr3.envIn = ((VarDef.name, Expr2.value):Expr1.envIn)
8       cond VarDef.name ∉ Expr1.envIn =>error "Already defined " ++ VarDef.name
9
10 rule Factor ::= VarRef
11 attr Factor.value = Factor.envIn[VarRef.name]
12      cond VarRef.name ∉ Factor.envIn =>error "Unknown Variable " ++ VarRef.name
13
14 -- Konstantenberechnung
15 rule Root ::= Expr
16 attr report "Output value = " ++ Expr.value
17
18 rule Term ::= Term Factor
19 attr Term1.value = Term2.value * Factor.value
20
21 rule Expr ::= Expr Term
22 attr Expr1.value = Expr2.value + Term.value
23
24 rule Expr ::= VarDef Expr Expr
25 attr Expr1.value = Expr3.value
26
27 -- Identitätsausgabe
28 rule Root ::= Expr
29 attr report "Identity Code = " ++ Expr.output
30
31 rule Expr ::= VarDef Expr Expr
32 attr Expr1.output = "let " ++ VarDef.name ++ " = " ++ Expr2.output ++
33                     " in " ++ Expr3.output
34
35 rule Term ::= Term Factor
36 attr Term1.output = Term2.output ++ " * " ++ Factor.output
37
38 rule Expr ::= Expr Term
39 attr Expr1.output = Expr2.output ++ " + " ++ Term.output
```

In den ersten 12 Zeilen dieses Beispiels wird der semantisch relevante Teil der Namensanalyse präsentiert, indem eine Art Definitionstabelle in Haskell-artiger Syntax leer initialisiert wird (Zeile 3) und mit den Bezeichnern, denen mittels **`let`**-Ausdrücken Werte zugewiesen werden (Zeile 7), die Werte aus dieser Definitionstabelle zur weiteren Berechnung weiter verwendet werden (Zeile 11). Semantisch geprüft werden Definition und Benutzung (Zeilen 8 und 12). Im folgenden Abschnitt von Zeile 14 bis 26 wird eine Art Konstantenfaltung durchgeführt und das Ergebnis dieser ausgegeben. Eine der Codegenerierung angelehnte Identitätsausgabe erfolgt in den letzten Zeilen des Beispiels.
Der Umfang des vollständigen Beispiels gemeinsam mit der Definition der benötigten Attribute und Definition der Grammatik umfasst über 110 Zeilen mit mehrheitlich Kopieranweisungen. Werden die in [38] und [29] vorgestellten Paradigmen angwandt, verringert sich der Quellumfang auf knapp unter 100 Zeilen. Die Differenz beträgt in diesem Fall weniger als 20 Zeilen.

Beispiel 1.1 – Semantisch relevanter Teil der Attributierungsregeln zur Ausgabegenerierung, Namensanalyse und Wertberechnung für eine Ausdrucksgrammatik

3 Attributgrammatiken

Grundlage für Attributgrammatiken sind kontextfreie Grammatiken. Kontextfreie Grammatiken lassen sich als (erweiterte) Backus-Naur-Form[5] ((E)BNF) oder in Form von Syntaxdiagrammen (siehe z. B. [44,45,8]) darstellen.

Definition 1. *Ein Tupel* $G \triangleq (N, T, P, Z)$ *wobei*

- N *eine endliche Menge an Nichtterminalen,*
- T *eine endliche Menge Terminale,*
- $P \subseteq N \times (N \cup T)^*$ *die Menge der Produktionen und*
- $Z \in N$ *dem (ausgezeichnetem)* Startsymbol

heißt ***kontextfreie Grammatik*** *genau dann, wenn* $N \cap T = \emptyset$

Die Menge der Produktionen induziert eine Ableitungsrelation. Ist $p \in P$ mit $p\colon X ::= u\ Y\ v$ für $X \in N$, $Y \in \Sigma, u, v \in \Sigma^*$ und $\Sigma = N \cup T$; dann ist Y aus X *ableitbar*, geschrieben als $X \Rightarrow Y$. Für die Relation der Ableitbarkeit sind transitiver und reflexiv-transitiver Abschluss wie üblich definiert.

Folgende Definition von Attributgrammatiken folgt der Darstellung aus [32] und [44]. Dies gilt ebenso für die darauf folgenden.

Definition 2. *Eine* ***attributierte Grammatik*** *ist ein Tupel* $AG \triangleq (G, A, R, B)$*, wobei*

- $G \triangleq (N, T, P, Z)$ *eine kontextfreie Grammatik ist, die eine abstrakte Syntax definiert,*
- $A \triangleq \biguplus_{X \in \Sigma} A_X$ *eine endliche Menge von* ***Attributen****,*
- $R \triangleq \biguplus_{p \in P} R_p$ *eine endliche Menge* ***Attributierungsregeln*** *der Form* $a_0 \leftarrow f(a_1, \cdots, a_k)$*, wobei* f *eine Funktion ist und*
- $B \triangleq \biguplus_{p \in P} B_p$ *eine endliche Menge von* ***Bedingungen*** *der Form* $\varphi(a_1, \cdots, a_k)$*, wobei* φ *ein Prädikat ist.*

Für alle $a_0 \leftarrow f(a_1, \cdots, a_k) \in R_p$ *einer Produktion* $p \in P, p\colon \mathtt{X}_0 ::= \mathtt{X}_1 \cdots \mathtt{X}_n$ *muss gelten:* $a_i \in A_{\mathtt{X}_0} \cup \cdots A_{\mathtt{X}_n}, i = 0, \cdots, k$ *und für alle* $\varphi(a_1, \cdots, a_k) \in B_p$ *für eine Produktion* $p \in P, p\colon \mathtt{X}_0 ::= \mathtt{X}_1 \cdots \mathtt{X}_n$ *muss gelten:* $a_i \in A_{\mathtt{X}_0} \cup \cdots A_{\mathtt{X}_n}, i = 1, \cdots, k$*.*

Die Definition von Attributgrammatiken nach dieser Art erlaubt allerdings schnell inkonsistente oder nicht berechenbare Attributgrammatiken, sodass folgende Definitionen notwendig sind.

Definition 3. *Eine Attributgrammatik* $AG \triangleq (G, A, R, B)$ *heißt* ***konsistent*** *genau dann, wenn für alle abstrakten Syntaxbäume* B *folgende Bedingungen erfüllt sind:*

- *für jedes Attribut* $a \in A$ *in jedem Knoten* k *höchstens eine definierende Regel* $\mathtt{X}.a \leftarrow e \in R$ *existiert und*
- *der* $Typ(k) = \mathtt{X}$ *ist.*

Die Attributgrammatik heißt ***vollständig****, wenn in jedem Knoten k mindestens eine Regel $\mathtt{X}.a \leftarrow e \in R$ existiert.*

In dieser Definition sind Knoten K vom Typ X genau dann, wenn es eine Regel in der abstrakten Syntax $G \triangleq (N, T, P, Z)$ existiert, sodass zum Aufbau des Knotens K im abstrakten Syntaxbaum T eine Produktion mit linker Seite X angewendet wurde. Technische Details zum Aufbau abstrakter Syntaxbäume sind [1,24] zu entnehmen.

Ziel dieser Arbeit ist der Nachweis, dass typische Muster abgeschlossen für verschiedene Arten der Berechenbarkeit von Attributgrammatiken sind. Dafür ist die Definition dieser Arten der Berechenbarkeit notwendig. Um für eine Attributgrammatik und eine potentielle Eingabe, d. h. einen abstrakten Syntaxbaum, zu bestimmen ob diese zusammen berechenbar sind, hilft folgende Definition:

Definition 4. *Für eine attributierte Grammatik $AG \triangleq (G, A, R, B)$ mit abstrakter Syntax $G \triangleq (N, T, P, Z)$ und einem abstrakten Syntaxbaum T von G mit Knoten $K_0, \cdots, K_n$ heißt eine Liste $[K_{i_0}.a_0, \cdots, K_{i_m}.a_m]$ aller Attribute aller Knoten von T* ***berechenbare Reihenfolge*** *genau dann, wenn jedes Attribut jedes Knotens genau einmal in der Liste vorkommt und für jedes Attribut $K_{i_j}.a_j$ eine Definition $K_{i_j} \leftarrow f(K_{i_{j_1}}.a_{j_1}, \cdots, K_{i_{j_q}}.a_{j_q})$ der entsprechenden Produktion p der Attributgrammatik existiert, sodass $j_1, \cdots, j_q < j$ ist.*

Implizit ist in der Definition der berechenbaren Reihenfolge bereits der Begriff der Abhängigkeit von Attributen enthalten:

Definition 5. *Sei $AG \triangleq (G, A, R, B)$ eine attributierte Grammatik mit abstrakter Syntax $G \triangleq (N, T, P, Z)$. Für jede Produktion $p\colon \mathtt{X}_0 ::= \mathtt{X}_1 \cdots \mathtt{X}_n \in P$ mit Attributierungsregel $\mathtt{X}_j.b \leftarrow f(\cdots \mathtt{X}_i.a \cdots)$ heißt $\mathtt{X}_i.a \rightarrow \mathtt{X}_j.b \subseteq A \times A$* ***lokale Abhängigkeit der Produktion*** *p. Für eine Produktion $p \in P$ heißt $DG_p \triangleq (A_p, DDP_p)$ mit $A_p \triangleq A_{\mathtt{X}_0} \cup \cdots \cup A_{\mathtt{X}_n}$ und $DDP_p \triangleq \{\mathtt{X}_i.a \rightarrow \mathtt{X}_j.b : \mathtt{X}_j.b \leftarrow f(\cdots \mathtt{X}_i.a \cdots) \in R_p\}$* ***lokaler Abhängigkeitsgraph zu*** *p.*

Auf Basis des Abhängigkeitsgraphen von Produktionen lassen sich erste berechenbare Klassen von Attributgrammatiken definieren.

Definition 6. *Eine attributierte Grammatik $AG \triangleq (G, A, R, B)$ mit abstrakter Syntax $G \triangleq (N, T, P, Z)$ heißt* ***lokal azyklisch*** *genau dann, wenn für alle Produktionen $p \in P$ die Graphen DG_p nach Definition 5 azyklisch sind.*

Definition 5 und Definition 6 führen zu der Definition wohldefinierter Attributgrammatik – jenen Attributgrammatiken für die eine berechenbare Reihenfolge (siehe Def. 4) existiert.

Definition 7. *Eine konsistente attributierte Grammatik AG heißt* ***wohldefiniert****, genau dann, wenn für jeden abstrakten Syntaxbaum T eine berechenbare Reihenfolge existiert.*

Lemma 1. *([44,42]) Eine konsistente attributierte Grammatik AG ist genau dann wohldefiniert, wenn sie vollständig ist und für jeden abstrakten Syntaxbaum T der Abhängigkeitsgraph DT_T azyklisch ist.*

Durch die Bestimmung in welcher Reihenfolge Attribute auswertbar sind, lassen sich in der Praxis effiziente Evaluatoren und Codegeneratoren für Attributgrammatiken implementieren.

Definition 8. *Für eine attributierte Grammatik* $AG \triangleq (G, A, R, B)$ *mit abstrakter Syntax* $G \triangleq (N, T, P, Z)$ *und allen Symbolen* $\mathtt{X} \in \Sigma$ *heißt die Partitionierung von* $A_{\mathtt{X}} = A_{\mathtt{X}}(1) \uplus \cdots \uplus A_{\mathtt{X}}(m_{\mathtt{X}})$ ***zulässige Zerlegung*** *genau dann, wenn für alle Symbole* $\mathtt{X}$ *gilt* $A_{\mathtt{X}}(i) \subseteq AS_{\mathtt{X}}$ *für* $i = m, m-2, \cdots$ *und* $A_{\mathtt{X}}(i) \subseteq AI_{\mathtt{X}}$ *für* $i = m-1, m-3, \cdots$.

Definition 9. *Eine attributierte Grammatik* $AG \triangleq (G, A, R, B)$ *mit abstrakter Syntax* $G \triangleq (N, T, P, Z)$ *heißt* ***zerlegbar*** *genau dann, wenn sie lokal azyklisch ist und für jedes Symbol* $\mathtt{X} \in \Sigma$ *eine zulässige Zerlegung* $A_{\mathtt{X}} = A_{\mathtt{X}}(1) \uplus \cdots \uplus A_{\mathtt{X}}(m_{\mathtt{X}})$ *existiert, sodass für jeden abstrakten Syntaxbaum* T *eine berechenbare Reihenfolge existiert so, dass die Attribute des Knoten* K *mit* $Typ(K) = \mathtt{X}$ *in der Reihenfolge* $A_{\mathtt{X}}(1), \cdots, A_{\mathtt{X}}(m_{\mathtt{X}})$ *berechnet werden können.*

Definition 10. *Eine zerlegbare attributierte Grammatik* $AG \triangleq (G, A, R, B)$ *mit zulässiger Zerlegung* $A_{\mathtt{X}} = A_{\mathtt{X}}(1) \uplus \cdots \uplus A_{\mathtt{X}}(m_{\mathtt{X}})$ *für alle Symbole* $\mathtt{X} \in \Sigma$ *heißt* ***zerlegt****.*

Eine Art von Attributgrammatiken bei der bereits ein Großteil zur Berechenbarkeit vor Aufbau des abstrakten Syntaxbaums erzeugt werden kann ist folgende:

Definition 11. *([31]) Sei* $AG \triangleq (G, A, R, B)$ *eine attributierte Grammatik mit abstrakter Syntax* $G \triangleq (N, T, P, Z)$*. Für alle Produktionen* $p \in P$ *mit* $p : \mathtt{X}_0 ::= \mathtt{X}_1 \cdots \mathtt{X}_n$ *heißt* $NDDP_p = DDP_p^+ \setminus \{(\mathtt{X}_i.a, \mathtt{X}_j.b) : \mathtt{X}_i.a$ *und* $\mathtt{X}_j.b$ *werden in* p *definiert* $\}$ ***normalisierte direkte Abhängigkeit der Produktion*** p*. Für alle Symbole* $\mathtt{X} \in \Sigma$ *sind die* ***induzierten Attributabhängigkeiten*** *das kleinste System von Mengen* $IDS_{\mathtt{X}} \subseteq A_{\mathtt{X}} \times A_{\mathtt{X}}$*,* $IDP_p \subseteq A \times A$*, welches folgende Gleichungen erfüllt:*

1. $NDPP_p \subseteq IDP_p$
2. $IDS_{\mathtt{X}} = \{\mathtt{X}.a \to \mathtt{X}.b \colon \exists q \in P$ *sodass* $\mathtt{X}.a \to \mathtt{X}.b \in IDP_q^+\}$
3. $IDP_p = IDP_p \cup IDS_{\mathtt{X}_0} \cup \cdots \cup IDS_{\mathtt{X}_n}$

IDS sind die ***induzierten Abhängigkeiten zwischen Symbolattributen****, IDP sind die* ***induzierten Abhängigkeiten zwischen Attributvorkommen****.*

Die Attributgrammatik AG heißt ***absolut azyklisch*** *genau dann, wenn für alle Nichtterminale* $\mathtt{X} \in N$ *und alle Produktionen* $p \in P$ $IDS_{\mathtt{X}}$ *und* IDP_p *azyklisch sind.*

Diese Definition legt implizit eine Halbordnung der Attribute fest. Kann in jedem Kontext eines Symbols einer Attributgrammatik für die mit diesem Symbol assoziierten Attribute eines Auswertereihenfolge, d.h. Zerlegung, angegeben werden, die diese Halbordnung enthält, dann heißt diese Attributgrammatik *geordnet.*

Definition 12. *Sei* $AG \triangleq (G, A, R, B)$ *eine attributierte Grammatik mit abstrakter Syntax* $G \triangleq (N, T, P, Z)$. *Für alle Symbole* $\mathtt{X} \in \Sigma$ *seien*

1. $T_{\mathtt{X}}(-1) \triangleq \emptyset$
2. $T_{\mathtt{X}}(0) \triangleq \emptyset$
3. $T_{\mathtt{X}}(2k-1) \triangleq \{a \in AS_{\mathtt{X}}$ *so, dass* $\forall b \in A_{\mathtt{X}} : \mathtt{X}.a \to \mathtt{X}.b \in IDS_{\mathtt{X}} \Rightarrow \exists j \leq 2k-1$ *mit* $\mathtt{X}.b \in T_{\mathtt{X}}(j)\}$
4. $T_{\mathtt{X}}(2k) \triangleq \{a \in AI_{\mathtt{X}}$ *so, dass* $\forall b \in A_{\mathtt{X}} : \mathtt{X}.a \to \mathtt{X}.b \in IDS_{\mathtt{X}} \Rightarrow \exists j \leq 2k$ *mit* $\mathtt{X}.b \in T_{\mathtt{X}}(j)\}$
5. $A_{\mathtt{X}}(i) \triangleq T_{\mathtt{X}}(m-i+1) \setminus T_{\mathtt{X}}(m-i-1)$ *mit* $i = 1, \cdots, m$

wobei $m \in \mathbb{N}$ *minimal und* $T_{\mathtt{X}}(m-1) \cup T_{\mathtt{X}}(m) = A_{\mathtt{X}}$. *Die attributierte Grammatik heißt* ***geordnet*** *(OAG) genau dann, wenn sie mit den Partitionen* $A_{\mathtt{X}} = A_{\mathtt{X}}(1) \uplus \cdots \uplus A_{\mathtt{X}}(m_{\mathtt{X}})$ *zerlegt ist und für alle Produktionen* $p \in P$ *der erweiterte Abhängigkeitsgraph* $EDP_p \triangleq IDP_p \cup \{\mathtt{X}.a \to \mathtt{X}.b : \exists h, k \in \mathbb{N}$ *mit* $\mathtt{X}.a \in A_{\mathtt{X}}(h) \wedge \mathtt{X}.b \in A_{\mathtt{X}}(k) \wedge h < k\}$ *azyklisch ist.*

Kastens zeigt in [27] wie durch Hinzufügen zusätzlicher Abhängigkeiten jede zerlegbare Attributgrammatik in eine geordnete Attributgrammatik überführt werden kann.

Im Allgmeinen ist die Kombination zweier geordneter Attributgrammatiken über derselben abstrakten Syntax nicht notwendigerweise wieder geordnet[37,27,12].

Ziel dieser Arbeit ist es einen Formalismus vorzustellen, der jedoch genau dies ermöglicht.

4 Typische Muster und deren Äquivalenzen

Ein typisches Muster ist eine Funktion im Raum der Attributgrammatiken über eine abstrakte Syntax.

Definition 13. *Sei* $G \triangleq (N, T, P, Z)$ *eine abstrakte Syntax und* $\mathsf{AG}_G \triangleq (G, \mathsf{A}, \mathsf{R}, \mathsf{B})$ *die Menge aller Attributgrammatiken mit abstrakter Syntax* G. *Eine Funktion* $\mathcal{M}_G \colon \mathsf{AG}_G \to \mathsf{AG}_G$ *heißt* ***typisches Muster*** *über der abstrakten Syntax* G *genau dann, wenn für alle zerlegbaren Attributgrammatiken* $AG \in \mathsf{AG}_G$ *die resultierende Attributgrammatik* $AG' = \mathcal{M}_G(AG)$ *zerlegbar ist.*

Die Definition dieser Muster lässt sich durch Elementweise Definition der Transferfunktionen über den veränderten Mengen erreichen, sodass
$\mathcal{M}_G \triangleq (\mathcal{M}_{G,A}, \mathcal{M}_{G,R}, \mathcal{M}_{G,B})$ verwendet werden kann. Zur einfacheren Argumentation werden die Bedingungen von Attributgrammatiken nicht beachtet. Diese lassen sich durch Attribute emulieren[44]. Die praktische Bedeutung solcher Bedingungen liegt in der Möglichkeit diese „Attribute“ als Abbruchkriterium bspw. der Übersetzung zu interpretieren. Im weiteren werden daher nur die beiden Unterfunktionen $\mathcal{M}_{G,A}$ und $\mathcal{M}_{G,R}$ betrachtet. Im folgenden wird auf die Verwendung des Index G verzichtet da dieser konstant ist und aus dem Kontext klar ist.

Die Änderung einer Attributgrammatik im Sinne der Transformation durch Musteranwendung lässt sich im Allgemeinen durch verschiedene Kombinationen des Hinzufügens und Entfernen von Attributen bzw. Attributierungsregeln darstellen. Somit können diese „Transferfunktionen“ über Attributgrammatiken wie folgt definiert werden.

Lemma 2. *Sei* $AG \in \mathsf{AG}, AG \triangleq (G, A, R, B)$ *eine zerlegbare Attributgrammatik. Für jede solche Attributgrammatik existiert ein Muster* $\mathcal{M} \triangleq (\mathcal{M}_A, \mathcal{M}_R)$ *sodass die resultierende Attributgrammatik* $AG' = (G, \mathcal{M}_A(A), \mathcal{M}_R(R), B)$ *zerlegbar ist, wobei*

- $\mathcal{M}_A(A) = (A \setminus \mathcal{M}_{A,-}(A)) \cup \mathcal{M}_{A,+}(A)$ *und*
- $\mathcal{M}_R(R) = (R \setminus \mathcal{M}_{R,-}(R)) \cup \mathcal{M}_{R,+}(R)$

ist.

Beweis. Durch Angabe eines Musters: Sei $\mathcal{M}_{A,-}(A) = \emptyset$, $\mathcal{M}_{R,-}(R) = \emptyset$ *und* $\mathcal{M}_{A,+}(A) = \emptyset$ *und* $\mathcal{M}_{R,+}(R) = \emptyset$. *Sei die resultierende Attributgrammatik nicht zerlegbar, dann steht dies im Widerspruch zur Vorraussetzung.*

Der Beweis zu Lemma 2 stellt eine triviale Lösung dar. Diese Lösung – keine Änderung der Attributgrammatik – steht immer zur Verfügung. Für eine praktisch relevante Beschreibung von Mustern müssen diese unabhängig von der konkreten Attributgrammatik, auf der diese angewendet werden, beschreibbar sein.

Zur Beschreibung eines Musters unabhängig von einer kontextfreien Grammatik bzw. unabhängig von der Attributgrammatik auf die Muster angewendet werden sollen ist es notwendig folgende Definition aufzustellen:

Definition 14. *Seien* a_i, $0 \leq i \leq n$, *für* $n \in \mathbb{N}$ *Variablen der Sorte* $\mathfrak{A}$, *sogenannte Musterattribute, dann sind* ***Musterattributwertterme*** $t_{[\![\mathfrak{A}]\!]}$ *wie folgt induktiv definiert:*

- c *ist ein Musterattributwertterm, wobei* c *eine Konstante ist;*
- a_i *ist ein Musterattributwertterm und*
- *für* $t^{(1)}_{[\![\mathfrak{A}]\!]}, \cdots, t^{(k)}_{[\![\mathfrak{A}]\!]}$, $k \in \mathbb{N}$ *Musterattributwertterme ist* $f(t^{(1)}_{[\![\mathfrak{A}]\!]}, \cdots, t^{(k)}_{[\![\mathfrak{A}]\!]})$ *ein Musterattributwertterm wobei* f *eine beliebige Funktion ist.*

Aus der funktionalen Programmierung bekannt ist, dass beliebige k-stellige Funktionen durch eine Funktion mit genau einem Argument darstellbar sind. Im folgenden werden daher Funktionen mit mehreren Argumenten wie in der Literatur üblich dargestellt ohne dabei auf Currying zurückzugreifen. Zu diesem Verfahren siehe auch [6,21].

Ausgehend von Musterattributwerttermen lassen sich die eigentlichen Musterterme beschreiben, dabei werden diese Terme in eine Musterattributierungsregel eingebettet und bilden gemeinsam einen Musterattributierungsterm.

Definition 15. *Sei $t_{[\![\mathfrak{A}]\!]}$ eine Musterattributwertterm über Musterattribute a_i, wobei $0 \leq i \leq n$, $n \in \mathbb{N}$. Darüber hinaus seien r_j, $0 \leq j \leq m$, $m \in \mathbb{N}$, Variablen der Sorte $\mathfrak{R}$, sogenannte Musterregeln, dann heißt eine Attributierungsregel*

$$\textbf{rule } r_j \textbf{ attr } a_i \leftarrow t_{[\![\mathfrak{A}]\!]}$$

***Musterattributierungsterm** genau dann, wenn a_i nicht in $t_{[\![\mathfrak{A}]\!]}$ vorkommt.*

Die Abbildung dieser Terme auf konkrete Attributgrammatiken lässt sich mittels Prädikaten und Abbildungsrelationen erreichen. Die Idee hinter der Verwendung von Prädikaten ist, dass durch Anwendung eines Prädikats die Beschreibung der Abbildung vergleichsweise unabhängig gestaltet werden kann.

Definition 16. *Sei* **rule** r_j **attr** $a_i \leftarrow t_{[\![\mathfrak{A}]\!]}$ *ein Musterattributierungsterm mit r_j, $0 \leq j \leq m, m \in \mathbb{N}$ Variablen der Sorte $\mathfrak{R}$ und a_i Variablen der Sorte $\mathfrak{A}$ mit $0 \leq i \leq n, n \in \mathbb{N}$; $t_{[\![\mathfrak{A}]\!]}$ einem Musterattributwertterm. Für eine Attributgrammatik $AG \triangleq (G, A, R, B)$ mit abstrakter Syntax $G \triangleq (N, T, P, Z)$ heißt das Prädikat $p_{AG}(r_j, a_i, t_{[\![\mathfrak{A}]\!]}, q)$ **abbildendes Prädikat** genau dann, wenn für alle Produktionen $q \in P$ der Attributgrammatik das Prädikat zu wahr oder falsch auswertet und $\bar{p}_{AG}(r_j, a_i, t_{[\![\mathfrak{A}]\!]}, q)$ **löschendes Prädikat** analog. Dabei gilt, wird p zu wahr ausgewertet, dann impliziert dies, dass $\bar{p}$ nicht zu wahr ausgewertet wird.*

Intuitiv klar ist, dass bei zwei Prädikaten für eine Produktion q zusammen mit der letzten Bedingung von Definition 16 genau drei Kombinationen möglich sind. Eine Produktion kann dann abgebildet, gelöscht oder ignoriert werden. Letzteres würde somit beschreiben, dass ein Musterattributierungsterm nicht auf die Produktion abgebildet wird. Im Falle dass das löschende Prädikat zu wahr auswertet wird die abgebildete Attributierungsregel entfernt bzw. ersetzt.

Durch die Prädikate werden somit die „passenden“ Produktionen für ein Muster ausgewählt. Das eigentliche Hinzufügen (oder Entfernen) von Attributierungsregeln und Attributen mittels Substitution der Musterattribute durch die in der Attributgrammatik vorkommenden Attribute[3]. Dabei können Musterattribute auch mit dem „leeren Attribut“ substituiert werden, sodass dieses Musterattribut entfernt wird.

Definition 17. *Sei $\mathfrak{R}$ die Menge aller Musterattributierungsterme $t_{j,[\![\mathbb{R}]\!]}$ der Form* **rule** r_j **attr** $a_i \leftarrow t_{[\![\mathfrak{A}]\!]}$ *für $0 \leq i \leq n, 0 \leq j \leq m$ für $m, n \in \mathbb{N}$. Für eine zerlegbare Attributgrammatik $AG \triangleq (G, A, R, B)$ und abstrakter Syntax $G \triangleq (N, T, P, Z)$ mit abbildenden und löschenden Prädikaten $p_{AG}(r_j, a_i, t_{[\![\mathfrak{A}]\!]}, q)$ und $\bar{p}_{AG}(r_j, a_i, t_{[\![\mathfrak{A}]\!]}, q)$ für Produktionen $q \in P$ der Attributgrammatik heißt die Menge $\mathfrak{R}'_{AG} \triangleq \{ r_j \sigma_{r_j,q} \colon \sigma_{r_j,q} [\mathtt{a}_1/a_1 \cdots \mathtt{a}_n/a_n], r_j \in \mathfrak{R}, q \in P, p_{AG}(r_j, a_i, t_{[\![\mathfrak{A}]\!]}, q)$ werte zu wahr aus und $\bar{p}_{AG}(r_j, a_i, t_{[\![\mathfrak{A}]\!]}, q)$ zu falsch $\}$ **auf** AG **abgebildete Musterattributierungsterme**. Das Attribut $\mathtt{a}_k$ heißt **in** r_j **abgebildete Attribut** genau dann, wenn in einer Substitution $\sigma_{r_j,q}$ $\mathtt{a}_k/a_k$ enthalten ist.*

[3] Hierbei werden praktisch zwei Substitutionen angewendet. Zur einfacheren Präsentation beschränkt sich die Präsentation jedoch auf die Substitution der Attribute.

Halten abgebildete Musterattributierungsterme eine Reihe von Eigenschaften ein, so lässt sich zeigen, dass deren Anwendung auf eine Attributgrammatik der Anwendung eines Musters entspricht. Folgende Definition legt die dafür notwendigen Eigenschaften fest:

Definition 18. *Seien $\mathfrak{R}$ die Menge aller Musterattributierungsterme und $\mathfrak{R}'_{AG}$ die Menge aller abgebildeten Musterattributierungsterme für eine zerlegbare Attributgrammatik $AG \triangleq (G, A, R, B)$ mit abstrakter Syntax $G \triangleq (N, T, P, Z)$ und den Substitutionen $\sigma_{r_j,q}$ für $q \in P$, heißt eine Substitution **setzend** genau dann wenn für $a_i \leftarrow t_{[\![\mathfrak{A}]\!]}$ das abgebildete Attribut $\mathtt{a}_i$ sonst nicht in $\sigma_{r_j,q}$ vorkommt und eine der folgende Bedingungen für bestehende Attributierungsregeln $r \in R_q$ der Form* **rule**$\mathtt{X}$ $::=$ u **attr**$\mathtt{a}_i \leftarrow e$*,gilt:*

- $\mathtt{a}_i$ *kommt nicht im Ausdruck e vor und alle abgebildeten Attribute* $\mathtt{a}_k$ *in* $\sigma_{r_j,q}$ *kommen bereits in e vor oder*
- $\mathtt{a}_i$ *kommt nicht im Ausdruck e vor und alle abgebildeten Attribute* $\mathtt{a}_l$ *in* $\sigma_{r_j,q}$ *die nicht bereits in e vorkommen sind nicht in A enthalten*

oder keine Attributierungsregel für $\mathtt{a}_i$ *in* R_q *existiert.*

Mit Hilfe dieser Eigenschaften lässt sich nun zeigen, dass solche Musterattributierungsterm, die abgebildeten Musterattributierungsterme und setzende Substitutionen ein Muster auf einer Attributgrammatik definieren.

5 Eigenschaften Typischer Muster

Theorem 1. *Sei $AG \triangleq (G, A, R, B)$ mit abstrakte Syntax $G \triangleq (N, T, P, Z)$ eine zerlegbare Attributgrammatik und $\mathfrak{R}$ eine Menge von Musterattributierungstermen mit setzenden Substitutionen $\sigma_{r_j,q}$ für alle $q \in P$, so definiert $\mathfrak{R}'_{AG}$ ein Muster.*

Beweis. Mittels Konstruktion der resultierenden Attributgrammatik
$AG' \triangleq (G, A', R', B)$ mit

- $A'_q = A_q \cup A_{q,+}$ *wobei* $A_{q,+} = \{\mathtt{a}_{k,j} \colon \exists r_j \in \mathfrak{R}, \mathtt{a}_k$ *abgebildetes Attribut in* $\sigma_{r_j,q}$ *und* $\mathtt{a}_k \notin A\}$
- $R'_q = (R_q \setminus R_{q,-}) \cup R_{q,+}$

wobei

$$R_{q,-} = \{r \colon r \in R_q \wedge r : \textbf{rule } r \textbf{ attr } \mathtt{a}_i \leftarrow e \wedge \exists r_j \in \mathfrak{R} \colon \mathtt{a}_i \text{ abgebildetes Attribut in } \sigma_{r_j,q}\}$$
$$R_{q,+} = \{r_j\sigma_{r_j,q} \colon r_j \in \mathfrak{R}, q \in P\}$$

und Nachweis der Zerlegbarkeit für die resultierende Attributgrammatik AG'.

Angenommen AG' sei nicht zerlegbar, dann ist, ohne Beschränkung der Allgemeinheit in q eine der folgenden Bedingungen verletzt:

1. *der lokale Abhängigkeitsgraph für q ist zyklisch oder*
2. *die Zerlegung für eines der in q vorkommenden Symbole nicht zulässig ist.*

Fall 1 kann nach Definition 18 nur eintreten, wenn AG bereits nicht zerlegbar war und steht somit im Widerspruch zur Voraussetzung; es werden höchstens neue Attribute einer bestehenden Attributierungsregel als zusätzliche Abhängigkeit mitgegeben.

Sei $\mathtt{X} \in (N \cup T)$ das Symbol für die die Zerlegung nicht zulässig ist. Es gilt wieder verschiedene Fälle zu betrachten:

1. *$\mathtt{a}_i$ ist ein abgebildetes Attribut, sodass $\mathtt{a}_i \in A_+$ – der Fall der hinzugefügten Attributierung oder*
2. *$\mathtt{a}_i$ ist das abgebildetes Attribut zu a_i mit zugeordneter Attributierungsregel* **rule** r_j **attr**$a_i \leftarrow t_{[\![\mathfrak{A}]\!]}$, *sodass $q \in R_{q,-}$ und $r_j\sigma_{r_j,q} \in R_{q,+}$ gilt – der Fall der Ersetzung einer Attributierungsvorschrift,*

Ohne Beschränkung der Allgemeinheit sei der maximale Partitionierungsindex in AG für $\mathtt{X}$ $m_\mathtt{X}$ mit der Partitionierung $A_\mathtt{X} = A_\mathtt{X}(1) \uplus \cdots \uplus A_\mathtt{X}(m_\mathtt{X})$. Im ersten Fall wird $m_\mathtt{X}$ um 2 erhöht, sodass für synthetisierte Attribute $\mathtt{a}_i \in A_\mathtt{X}(m_{\mathtt{X}+2})$ gilt und $A_\mathtt{X}(m_{\mathtt{X}+1}) = \emptyset$ und für ererbte Attribute $\mathtt{a}_i \in A_\mathtt{X}(m_{\mathtt{X}+1})$ sowie $A_\mathtt{X}(m_{\mathtt{X}+2}) = \emptyset$ gilt. Ist die Partitionierung für AG' dann nicht zulässig, so muss dies bereits in AG der Fall gewesen sein, wiederum im Widerspruch zur Voraussetzung.

Nach Definition 18 werden höchstens noch nicht bestehende Attribute bei der Ersetzung den Abhängigkeiten hinzugefügt, somit kann das Verfahren für das Hinzufügen von Attributen angewendet werden wofür bereits gezeigt wurde, dass AG' nur dann nicht zerlegbar ist, wenn dies bereits bei AG der Fall war.

Für den praktischen Aufbau von Mustern aus Basismustern – dem Ziel dieser Arbeit – ist es notwendig die dafür benötigte Komposition als zerlegungserhaltend zu zeigen:

Theorem 2. *Seien $AG \triangleq (G, A, R, B)$ eine zerlegbare Attributgrammatik mit abstrakter Syntax $G \triangleq (N, T, P, Z)$ und zwei Muster $\mathcal{M}_{G,1}$ und $\mathcal{M}_{G,2}$ gegeben mittels Musterattributierungstermen, dann ist die resultierende Attributgrammatik $AG'' \triangleq (\mathcal{M}_{G,2} \circ \mathcal{M}_{G,1})(AG)$ bestimmt als $\mathcal{M}_{G,2}(\mathcal{M}_{G,1}(AG))$ zerlegbar.*

Beweis. Angenommen AG'' wäre nicht zerlegbar, dann existieren folgende Fälle:

1. *die ursprüngliche Attributgrammatik AG ist nicht zerlegbar,*
2. *die nach Anwendung von $\mathcal{M}_{G,1}$ resultierende Attributgrammatik $AG' \triangleq \mathcal{M}_{G,1}(AG)$ ist nicht zerlegbar oder*
3. *AG'' ist nicht zerlegbar und die anderen Fälle sind nicht eingetreten.*

In Fall 1 ist bereits die Voraussetzung nicht erfüllt. In Fall 2 wird die Voraussetzung, dass $\mathcal{M}_{G,1}$ ein Muster ist verletzt und im letzten Fall, Fall 3, kann dies nur der Fall sein, wenn einer der anderen Fälle bereits eingetreten ist oder $\mathcal{M}_{G,2}$ kein Muster ist; dies steht im Widerspruch zu den Voraussetzungen.

Mit diesen beiden Thesen lässt sich zeigen dass Muster aufgebaut werden können. Die genaue Darstellung von Mustern unabhängig von einer Attributgrammatik nur bezüglich einer abstrakten Syntax bzw. unabhängig von der abstrakten Syntax sprengt den Rahmen dieser Arbeit. Bereits in [**?**,**?**,9] wurde eine frühe Form typischer Muster angewendet. Dabei wurde gezeigt, dass Laufzeit und Kompaktheitsgrad anderen Ansätzen, wie bspw. OCL in nichts nachstehen.

6 Zusammenfassung

In dieser Arbeit wurden typische Muster als Möglichkeit der knapperen Beschreibung der Sprachsemantik präsentiert. Es wurde gezeigt, dass und wie sich aus einer Reihe simpler Basismuster und deren Komposition neue Muster erstellen lassen. Ein wesentlicher Vorteil typischer Muster gegenüber üblichen Verfahren, wie bspw. in [29,38] oder [11] ist, dass die Berechenbarkeit garantiert ist. Werden typische Muster auf eine geordnete Attributgrammatik angewandt, ist sichergestellt, dass das Resultat mindestens zerlegbar ist. Eine zulässige Zerlegung bzw. eine lineare Anordnung, lässt sich ebenfalls ermitteln. Bereits Kastens hat in [27] gezeigt, dass jede zerlegbare Attributgrammatik durch Einführung zusätzlicher Abhängigkeiten in eine geordnete Attributgrammatik überführt werden kann.

Typische Muster überbrücken die große Lücke zwischen den grobkörnigen Modulansätzen wie diese in [25] und [7] vorgestellt werden und einfachen Erweiterungen für die [29,38,34] einen Überblick geben. Typische Muster können grundsätzlich wie entfernte Attribute oder Attribute höherer Ordnung oder inkrementeller Auswertung verwendet werden. Die Eigenschaften der in dieser Arbeit vorgestellten typischen Muster erlauben jedoch die Verwendung mit geordneten Attributgrammatiken, welche aufgrund ihrer Eigenschaften ein sehr günstiges Laufzeitverhalten zeigen und deswegen alternativen Ansätzen vorzuziehen sind[9,10].

Im Bereich der Definition der Sprachsemantik existieren eine Reihe weiterer Arbeiten, wie [19,15] bei der Aspekte bzw. Termersetzung genutzt werden um die Sprachsemantik erweiterbar zu gestalten. Eine Auf Termersetzung basierende Implementierung einer Sprache zur Definition der Namensanalyse wird hingegen in [33] vorgestellt. Das Ziel typischer Muster war jedoch nicht die Verwendung von Termersetzung – dies erzeugt üblicherweise langsamere Übersetzer – sondern die Definition eines Formalismus, der mit geordneten Attributgrammatiken einher geht.

Darüber hinaus sind typische Muster auch nicht mit den kompositionellen Ansätzen die bspw. in [14,16] vorgestellt werden zu vergleichen. Bei typischen Mustern ist die abstrakte Syntax der Attributgrammatik bspw. „fixiert“ und wird nicht durch Anwendung eines Musters (bzw. durch Anwendung der Komposition) geändert.

In weiteren Arbeiten muss untersucht werden wie diese Muster aus Basismustern aufgebaut werden müssen.

Literatur

1. Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools (2Nd Edition). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2006)
2. Albreshne, A., Lahcen, A.A., Pasquier, J., Abdaladhem, A.: A framework and its associated process-oriented domain specific language for managing smart residential environments. International Journal of Smart Home 7(6), 377–392 (2013)
3. Albreshne, A., Lahcen, A.A.L., Pasquier, J.: Using a residential environment domain ontology for discovering and integrating smart objects in complex scenarios. Procedia Computer Science 32, 997 – 1002 (2014), http://www.sciencedirect.com/science/article/pii/S1877050914007248
4. Albreshne, A., Pasquier, J.: A domain specific language for high-level process control programming in smart buildings. Procedia Computer Science 63, 65 – 73 (2015), http://www.sciencedirect.com/science/article/pii/S1877050915024412
5. Backus, J.W., Bauer, F.L., Green, J., Katz, C., McCarthy, J., Naur, P., Perlis, A.J., Rutishauser, H., Samelson, K., Vauquois, B., Wegstein, J.H., van Wijngaarden, A., Woodger, M., van der Poel, W.L.: Revised report on the algorithmic language algol 60. Numerische Mathematik 4(1), 420–453 (1962), http://dx.doi.org/10.1007/BF01386340
6. Barendregt, H.P., et al.: The lambda calculus, vol. 2. North-Holland Amsterdam (1984)
7. Baum, B.: Another kind of modular attribute grammars. In: Kastens, U., Pfahler, P. (eds.) Compiler Construction, Lecture Notes in Computer Science, vol. 641, p. 44–50. Springer Berlin Heidelberg (1992), http://dx.doi.org/10.1007/3-540-55984-1_5
8. Bell, S., Gilbert, E.J.: Learning recursion with syntax diagrams. SIGCSE Bull. 6(3), 44–45 (Sep 1974), http://doi.acm.org/10.1145/988881.988890
9. Berg, C., Zimmermann, W.: Evaluierung von Möglichkeiten zur Implementierung von Semantischen Analysen für Domänenspezifische Sprachen (2014)
10. van Binsbergen, L.T., Bransen, J., Dijkstra, A.: Linearly ordered attribute grammars: With automatic augmenting dependency selection. In: Proceedings of the 2015 Workshop on Partial Evaluation and Program Manipulation. pp. 49–60. PEPM '15, ACM, New York, NY, USA (2015), http://doi.acm.org/10.1145/2678015.2682543
11. Boyland, J.T.: Remote attribute grammars. Journal of the ACM 52(4), 627–687 (Jul 2005), http://doi.acm.org/10.1145/1082036.1082042
12. Bransen, J., Middelkoop, A., Dijkstra, A., Swierstra, S.D.: The kennedy-warren algorithm revisited: Ordering attribute grammars. In: Russo, C., Zhou, N.F. (eds.) Practical Aspects of Declarative Languages: 14th International Symposium, PADL 2012, Philadelphia, PA, USA, January 23-24, 2012. Proceedings, pp. 183–197. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-27694-1_14
13. Campusano, M., Fabry, J.: Live robot programming: The language, its implementation, and robot {API} independence. Science of Computer Programming 133, Part 1, 1 – 19 (2017), http://www.sciencedirect.com/science/article/pii/S0167642316300697
14. Economopoulos, R., Fischer, B.: Higher-order transformations with nested concrete syntax. In: Proceedings of the Eleventh Workshop on Language Descriptions, Tools and Applications. p. 4. ACM (2011)

15. Ekman, T., Hedin, G.: Rewritable Reference Attributed Grammars. In: Odersky, M. (ed.) ECOOP 2004 – Object-Oriented Programming, Lecture Notes in Computer Science, vol. 3086, pp. 147–171. Springer Berlin Heidelberg (2004), http://dx.doi.org/10.1007/978-3-540-24851-4_7
16. Farrow, R., Marlowe, T.J., Yellin, D.M.: Composable attribute grammars: Support for modularity in translator design and implementation. In: Proceedings of the 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 223–234. POPL '92, ACM, New York, NY, USA (1992), http://doi.acm.org/10.1145/143165.143210
17. Hedin, G.: Compiler Construction: 5th International Conference, CC '94 Edinburgh, U.K., April 7–9, 1994 Proceedings, chap. An overview of door attribute grammars, pp. 31–51. Springer Berlin Heidelberg, Berlin, Heidelberg (1994), http://dx.doi.org/10.1007/3-540-57877-3_3
18. Hedin, G.: Reference attributed grammars. Informatica (Slovenia) 24(3), 301–317 (2000)
19. Hedin, G., Magnusson, E.: Jastadd—an aspect-oriented compiler construction system. Science of Computer Programming 47(1), 37 – 58 (2003), http://www.sciencedirect.com/science/article/pii/S0167642302001090
20. Hess, S., Gross, A., Maier, A., Orfgen, M., Meixner, G.: Standardizing model-based in-vehicle infotainment development in the german automotive industry. In: Proceedings of the 4th International Conference on Automotive User Interfaces and Interactive Vehicular Applications. pp. 59–66. AutomotiveUI '12, ACM, New York, NY, USA (2012), http://doi.acm.org/10.1145/2390256.2390265
21. Hindley, J.R., Seldin, J.P.: Lambda-calculus and combinators: an introduction, vol. 13. Cambridge University Press Cambridge (2008)
22. Jazayeri, M., Ogden, W.F., Rounds, W.C.: The intrinsically exponential complexity of the circularity problem for attribute grammars. Commun. ACM 18(12), 697–706 (Dec 1975), http://doi.acm.org/10.1145/361227.361231
23. Jazayeri, M., Walter, K.G.: Alternating semantic evaluator. In: Proceedings of the 1975 Annual Conference. pp. 230–234. ACM '75, ACM, New York, NY, USA (1975), http://doi.acm.org/10.1145/800181.810328
24. Kadhim, B., Waite, W.: Maptool — supporting modular syntax development. In: Gyimóthy, T. (ed.) Compiler Construction, Lecture Notes in Computer Science, vol. 1060, pp. 268–280. Springer Berlin Heidelberg (1996), http://dx.doi.org/10.1007/3-540-61053-7_67
25. Kastens, U., Waite, W.M.: Modularity and reusability in attribute grammars. Acta Informatica 31(7), 601–627 (1994), http://dx.doi.org/10.1007/BF01177548
26. Kastens, U., Waite, W.: An abstract data type for name analysis. Acta Informatica 28(6), 539–558 (1991), http://dx.doi.org/10.1007/BF01463944
27. Kastens, U.: Ordered attributed grammars. Acta Informatica 13(3), 229–256 (1980)
28. Kastens, U.: Liga: A language independent generator for attribute evaluators. Bericht der Reihe Informatik (63) (1989)
29. Kastens, U.: Attribute Grammars as a specification method. In: Alblas, H., Melichar, B. (eds.) Attribute Grammars, Applications and Systems, Lecture Notes in Computer Science, vol. 545, pp. 16–47. Springer Berlin Heidelberg (1991), http://dx.doi.org/10.1007/3-540-54572-7_2
30. Kastens, U., Hutt, B., Zimmermann, E.: GAG, a practical compiler generator, Lecture Notes in Computer Science, vol. 141. Springer-Verlag (1982)

31. Kennedy, K., Warren, S.K.: Automatic generation of efficient evaluators for attribute grammars. In: Proceedings of the 3rd ACM SIGACT-SIGPLAN Symposium on Principles on Programming Languages. pp. 32–49. POPL '76, ACM, New York, NY, USA (1976), http://doi.acm.org/10.1145/800168.811538
32. Knuth, D.E.: Semantics of context-free languages. Mathematical systems theory 2(2), 127–145 (1968), http://dx.doi.org/10.1007/BF01692511
33. Konat, G., Kats, L., Wachsmuth, G., Visser, E.: Software Language Engineering: 5th International Conference, SLE 2012, Dresden, Germany, September 26-28, 2012, Revised Selected Papers, chap. Declarative Name Binding and Scope Rules, pp. 311–331. Springer Berlin Heidelberg, Berlin, Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-36089-3_18
34. Koskimies, K.: Object-orientation in attribute grammars. In: Alblas, H., Melichar, B. (eds.) Attribute Grammars, Applications and Systems: International Summer School SAGA Prague, Czechoslovakia, June 4–13, 1991 Proceedings, pp. 297–329. Springer Berlin Heidelberg, Berlin, Heidelberg (1991), http://dx.doi.org/10.1007/3-540-54572-7_11
35. Lewis, P., Rosenkrantz, D., Stearns, R.: Attributed translations. Journal of Computer and System Sciences 9(3), 279 – 307 (1974), http://www.sciencedirect.com/science/article/pii/S0022000074800450
36. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. 37(4), 316–344 (Dec 2005), http://doi.acm.org/10.1145/1118890.1118892
37. Natori, S., Gondow, K., Imaizumi, T., Hagiwara, T., Katayama, T.: On eliminating type 3 circularities of ordered attribute grammars. In: Parigot, D., Mernik, M. (eds.) 2nd Int. Workshop on Attribute Grammars and their Applications (WAGA'99). pp. 93–112. INRIA Rocquencourt France (1999)
38. Paakki, J.: Attribute grammar paradigms – a high-level methodology in language implementation. ACM Comput. Surv. 27(2), 196–255 (Jun 1995), http://doi.acm.org/10.1145/210376.197409
39. Stewart, G., Gowda, M., Mainland, G., Radunovic, B., Vytiniotis, D., Agullo, C.L.: Ziria: A dsl for wireless systems programming. In: Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 415–428. ASPLOS '15, ACM, New York, NY, USA (2015), http://doi.acm.org/10.1145/2694344.2694368
40. Swierstra, D., Vogt, H.: Higher order attribute grammars. In: Alblas, H., Melichar, B. (eds.) Attribute Grammars, Applications and Systems: International Summer School SAGA Prague, Czechoslovakia, June 4–13, 1991 Proceedings, pp. 256–296. Springer Berlin Heidelberg, Berlin, Heidelberg (1991), http://dx.doi.org/10.1007/3-540-54572-7_10
41. Swierstra, S.D., Alcocer, P.R.A., Saraiva, J.: Designing and implementing combinator languages. In: International School on Advanced Functional Programming. pp. 150–206. Springer (1998)
42. Tienari, M.: On the definition of an attribute grammar. In: Jones, N.D. (ed.) Semantics-Directed Compiler Generation: Proceedings of a Workshop Aarhus, Denmark, January 1980, pp. 408–414. Springer Berlin Heidelberg, Berlin, Heidelberg (1980), http://dx.doi.org/10.1007/3-540-10250-7_31
43. Vogt, H.H., Swierstra, S.D., Kuiper, M.F.: Higher order attribute grammars. In: Proceedings of the ACM SIGPLAN 1989 Conference on Programming Language Design and Implementation. pp. 131–145. PLDI '89, ACM, New York, NY, USA (1989), http://doi.acm.org/10.1145/73141.74830

44. Waite, W.M., Goos, G.: Compiler construction. Springer-Verlag, New York (1984)
45. Wirth, N.: What can we do about the unnecessary diversity of notation for syntactic definitions? Commun. ACM 20(11), 822–823 (Nov 1977), http://doi.acm.org/10.1145/359863.359883