Erläuterungen zur attributierten Grammatik für PEARL

Dr. U. Kastens, Karlsruhe

1. Überblick

Die Definition der Programmiersprache PEARL und ihrer Teilsprache Basic PEARL ist im Normentwurf DIN 66253 Teil 2 standardisiert worden [1]. Er soll sowohl Implementierern als auch Anwendern als Referenzdokument dienen, das unterschiedliche Interpretationen soweit wie möglich ausschließt. Deshalb ist der größte Teil der Spracheigenschaften mit formalen Methoden beschrieben: die syntaktische Struktur und die Kontextabhängigkeiten durch eine attributierte Grammatik (AG) und die Bedeutung - soweit sie die asynchrone Ausführung von Programmen betrifft - durch Petri-Netze.

Anders als Lehrbücher oder Einführungen zu Programmiersprachen stellen formale Sprachdefinitionen naturgemäß höhere Anforderungen an die Vorkenntnisse der Leser. Klare Beschreibungskonzepte können und sollen zwar das Verständnis erleichtern, aber die Notwendigkeit zur Einarbeitung in das Dokument und seine Beschreibungsmethoden vermeiden sie nicht vollständig.

Wir wollen in diesem Papier Verständnishilfen geben, die den Zugang zu der formalen Definition der statischen Spracheigenschaften durch die AG erleichtern. Wir wenden uns dabei insbesondere an die Leser, die den Normentwurf heranziehen wollen, um Fragen zu Spracheigenschaften zu beantworten. Sprachimplementierer, die den Normentwurf als Teil des Pflichtenheftes oder der Übersetzerspezifikation benutzen, werden sich über den Rahmen dieser Erläuterungen hinaus Detailkenntnisse erarbeiten müssen.

Im zweiten Abschnitt werden die Grundprinzipien der Beschreibungsmethode "Attributierte Grammatiken" erläutert. Der dritte Abschnitt führt in das Arbeiten mit dem Normdokument ein und gibt am konkreten Beispiel Hinweise, wie man die Gliederung und verschiedene Arten von Querverweisen nutzt, um Fragen zu Spracheigenschaften zu beantworten. In Abschnitt 4 erläutern wir einige Beschreibungstechniken, die zur Definition zentraler Spracheigenschaften, wie Gültigkeit von Definitionen, Datentypen und die Teilspracheigenschaft angewandt werden. Ein Überblick über diese Techniken ist wichtig, weil sich diese Spracheigenschaften auf viele Elemente der Sprache auswirken. Die konkrete Notation der AG in der Beschreibungssprache ALADIN wird in Abschnitt 5 einführend erklärt.

2 Attributierte Grammatiken als Sprachdefinition

Eine AG definiert sowohl die syntaktische Struktur von Programmen einer Sprache als auch kontextabhängige Eigenschaften und Bedingungen. Anhand einer solchen Sprachdefinition kann deshalb entschieden werden, ob ein gegebenes Programm statisch korrekt ist, d.h. ob ein Sprachübersetzer es akzeptieren muß. Die Frage, ob ein Programm zusammen mit einem Eingabe-Datensatz auch ausführbar ist, wird im allgemeinen nicht durch die AG beantwortet, sondern durch die Definition der dynamischen Semantik - der Bedeutung der Sprachelemente.

Aus vielen informellen oder teilweise formalisierten Sprachdefinitionen ist der folgende Beschreibungsstil bekannt: Die syntaktische Struktur von Programmen wird durch eine kontext-freie Syntax, z.B. in

Backus-Naur-Form angegeben. Hinzu kommen verbal formulierte Aussagen, z.B. über die Gültigkeitsbereiche von Definitionen und die Typen von Ausdrücken, die nicht kontext-frei erfaßbar sind. (Die dem Normentwurf vorangehenden PEARL-Beschreibungen (z.B. [2], [3]) folgen diesem Stil.) Eine AG umfaßt auch diese kontextabhängigen Spracheigenschaften. Da das "Skelett" einer AG eine kontext-freie Grammatik in gewohnter Notation ist, kommt die Beschreibungsmethode dem gewohnten Denken entgegen, das zwischen syntaktischer Struktur und kontextabhängigen Regeln unterscheidet.

Ebenso wie verbale Definitionen Strukturelementen der Sprache Eigenschaften zuordnen (z.B. "der Typ eines Ausdrucks"), ordnet eine AG den Symbolen der kontext-freien Grammatik Attribute zu (z.B. Ausdruck.Typ). Die Analogie läßt sich fortsetzen: Eine verbale Aussage wie "Der Typ einer Vergleichsoperation ist BOOL" entspricht einer Attributregel

```
Vergleich.Typ := BOOL,
```

die der Produktion

```
Vergleich ::= Formel '=' Formel
```

zugeordnet ist. Restriktionen wie "Die linke und die rechte Seite einer Zuweisung müssen gleichen Typ haben" werden als Kontextbedingungen über Attributen formuliert, z.B. zur Produktion

```
Zuweisung ::= Variable ':=' Ausdruck
```

die Bedingung

```
CONDITION Variable.Typ = Ausdruck.Typ
```

Die Beschreibungsmethode "attributierte Grammatik" beruht also auf folgenden Prinzipien:

1. Eine AG basiert auf einer kontext-freien Grammatik.
2. Den Terminalen und Nichtterminalen der kontext-freien Grammatik werden Attribute zugeordnet, welche Eigenschaften dieser Sprachelemente beschreiben.
3. Den Produktionen der kontext-freien Grammatik werden Attributregeln zugeordnet, welche beschreiben, wie Attribute (Eigenschaften) zu bestimmen sind. Kontextabhängigkeiten werden durch Bezugnahme auf Attribute aus dem Kontext ausgedrückt.
4. Durch Kontextbedingungen zu Produktionen wird die Menge der syntaktisch korrekten Programme eingeschränkt auf die Menge der statisch korrekten (übersetzbaren) Programme.

Wir verzichten hier auf ein Beispiel und verweisen auf [4]. Eine exakte Definition attributierter Grammatiken und weitere Beispiele findet man z.B. in [6].

Will man anhand einer AG feststellen, ob ein gegebenes Programm korrekt ist, so geht man wie folgt vor:

1. Man bestimmt die syntaktische Struktur des Programms. Dazu leitet man das Programm unter Anwendung von Produktionen der kontext-freien Grammatik aus dem Startsymbol (hier: sx_PEARL_Programm) ab und stellt die Ableitungsschritte als Strukturbaum dar. Das Startsymbol ist dessen Wurzel, die Knoten sind Terminale und Nichtterminale der Grammatik, die Symbole des Programms sind Blätter des Baums. Zu jeder angewandten Produktion verbinden Kanten die Symbole der rechten Seite mit dem Nichtterminal der linken Seite der Produktion. Eine Reihe von Symbolen kann man aus dem Strukturbaum weglassen, da sie ausschließlich syntaktische Bedeutung haben (z.B. BEGIN, END, ;). Man spricht dann auch von einem abstrakten Strukturbaum.
2. Der Strukturbaum wird durch Attribute zum attributierten Strukturbaum erweitert. Jedem Knoten ordnet man die Attribute zu, die zum entsprechenden Symbol der Grammatik gehören.
3. Man bestimmt die Werte der Attribute von Baumknoten (und dadurch die kontextabhängigen Eigenschaften des Programms). Dazu wendet man diejenigen Attributregeln an, die den Produktionen

zugeordnet sind, welche zur Herstellung des Baums benutzt wurden. Häufig werden in einer Attributregel Attribute benutzt, die nicht durch Attributregeln zur gerade betrachteten Produktion bestimmt werden. Ihre Attributregeln findet man bei den in der unmittelbaren Umgebung angewandten Produktionen. Um den durch Attribute beschriebenen Informationsfluß nachzuvollziehen (z.B. von Objektdeklarationen zu Anwendungen) muß man im Strukturbaum entlang der Kanten auf- und absteigen.

4. Um die Korrektheit eines Programms zu ermitteln, wertet man wie in (3) die Kontextbedingungen zu den angewandten Produktionen aus.

Ist eine AG wohldefiniert [8], so können nach diesem Verfahren in jedem Strukturbaum die Werte aller Attribute eindeutig bestimmt werden. Die Wohldefiniertheit der PEARL-AG ist mit Hilfe des Übersetzer-erzeugenden Systems GAG [7] nachgewiesen worden.

Zur Beantwortung spezieller Fragen ist es meist nicht nötig, die Attributierung eines Programm vollständig nachzuvollziehen. Man kann sich auf Teile eines Strukturbaums (z.B. eine Zuweisung mit ihrer unmittelbaren Umgebung) und auf wenige Attribute (z.B. die Typ-Attribute) beschränken. Zusammenhänge über einen größeren Kontext lassen sich meist aus der Kenntnis einiger zentraler Beschreibungskonzepte erschließen (siehe Abschnitt 4).

3 Gliederung der attributierten Grammatik und Querverweise

In diesem Abschnitt stellen wir die Gliederung der AG im Normentwurf vor. Anhand eines Beispiels zeigen wir, wie man Antworten auf Fragen nach PEARL-Eigenschaften in der AG findet.

Die AG für PEARL ist in fünf Kapitel (Parts) gegliedert, welche jeweils die Definitionen eines Teilbereichs der Sprache zusammenfassen. Jedes Kapitel beginnt mit der Definition der dort angewandten Attributtypen. Es folgen Abschnitte, in denen jeweils zuerst die Attributzuordnungen für Symbole, dann die Regeln für diese Symbole und schließlich die darin angewandten Funktionen angegeben sind. Die Typ- und Symboldefinitionen sind alphabetisch, die Regeln und Funktionen sind im Sinne einer sukzessiven Verfeinerung angeordnet (in Part Six und Part Seven auch alphabetisch). Diese Gliederung ist hilfreich für den ersten Einstieg in die AG, bei dem man zwar sein Problem einem Themenkreis (z.B. "Tasking") zuordnen kann, aber noch keine Begriffe und Namen aus der AG kennt.

Die gesamte AG ist mit Zeilennummern versehen, deren führende Stellen den Abschnitt kennzeichnen, z.B. 15.0274. (Im folgenden verweisen wir auf die AG im Normentwurf mit diesen Zeilennummern: (15.0274).) Die Anhänge B und C stellen mit Hilfe der Zeilennummern Querverweise in die AG her: Anhang B enthält die kontext-freien Produktionen - alphabetisch sortiert nach dem Namen des Symbols auf der linken Seite - zusammen mit einem Verweis auf die Stelle der zugehörigen AG-Regel. Anhang C gibt zu jedem Namen aus der AG die Definitionsstelle(n) an.

Mit Hilfe der Anhänge beantwortet man leicht folgende Fragentypen:

(a) Wie wird das Symbol sx_exp kontext-frei abgeleitet? Im Anhang B die Produktionen anhand der alphabetischen Ordnung verfolgen!

(b) Welche Attributregeln gibt es zur Produktion sx_glob_id ::= sx_id? Aus Anhang B dem Verweis in die AG folgen!

(c) Welche Attribute hat sx_exp? Aus Anhang C dem Verweis auf die Symboldefinition folgen!

(d) Wie ist die Funktion f_coerce (der Typ tp_type,...) definiert? Mit Anhang C die Definition aufsuchen!

Als Beispiel wollen wir die Beantwortung folgender Frage vorführen: Welche Typen sind in Basic PEARL auf der linken Seite einer Zuweisung zulässig?

Den Einstieg finden wir z.B., indem wir im Anhang B die Ableitung von sx_statement nach sx_assignment verfolgen und in der AG die Regeln in (16.0959) und (16.0994) aufsuchen. (Die Zuweisung gehört zum Abschnitt "Expressions", da sie in PEARL auch an Ausdrucksposition stehen kann.)

Wegen der Teilsprachbedingung in (16.0998) brauchen wir die zweite Alternative nicht zu betrachten (siehe Abschnitt 4.1). In (16.0964) finden wir eine Bedingung zu einem Attribut der linken Seite der Zuweisung

```
CONDITION
f_assignable(sx_destination.at_attr.s_type)
AND
f_no_INV_component(sx_destination.at_attr)
```

Hinzu kommt eine Teilsprachbedingung über dasselbe Attribut zur zweiten Regel für sx_destination in (16.1061):

```
sx_destination.at_attr.s_type
   IS tp_basic_type
```

In (16.0185) ist der Typ von at_attr mit tp_attr angegeben. Mit Anhang C finden wir die Typdefinition dazu in (15.0147): tp_attr beschreibt Zugriffsfunktionen durch Angabe der Invariabilität und des Typs des Bezugsobjekts. In (15.0274) ist die Abstraktion der PEARL-Typen (tp_type) als Vereinigung von tp_basic_type mit einigen anderen Typen definiert. Aus der Definition für tp_basic_type in (15.0156) entnimmt man, daß die Teilsprachbedingung in (16.1061) nur für die PEARL-Typen FIXED, FLOAT, BIT, CHAR, CLOCK oder DUR erfüllt ist. Anhand der Definition von f_assignable in (16.3004) stellt man fest, daß sie für diese Typen das Ergebnis TRUE hat. Der erste Teil der Bedingung in (16.0964) ist für Basic PEARL also immer wahr. Der zweite Teil ist erfüllt, wenn es sich um eine Zugriffsfunktion für variable Objekte handelt. Diese Aussage erhält man durch Untersuchen der Funktion f_no_INV_component in (16.2960) für den Fall tp_attr. Die Antwort auf unsere Frage lautet also: **In Basic PEARL dürfen auf der linken Seite von Zuweisungen variable Objekte der Typen FIXED, FLOAT, BIT, CHAR, CLOCK oder DUR stehen.**

4 Beschreibungstechniken

Wie in allen höheren Programmiersprachen gibt es auch in PEARL einige Spracheigenschaften, die sich auf Sprachelemente in allen Teilbereichen der Sprache auswirken, z.B. die Gültigkeitsregeln für Definitionen, der Typbegriff für Objekte, die Typbestimmung in Ausdrücken. Im Normentwurf gehören auch die Teilsprachbedingungen für Basic PEARL zu solchen übergreifenden Eigenschaften. In diesem Abschnitt wollen wir dem Leser zumindest eine grobe Vorstellung von den Techniken vermitteln, mit denen diese Eigenschaften beschrieben werden, um die Beantwortung spezieller Fragen zu PEARL zu erleichtern.

4.1 Teilsprachbedingungen

Die Definition von Basic PEARL - eine echte Teilsprache von PEARL - ist in die PEARL-Definition integriert. Kontext-freien Produktionen, die in Basic PEARL nicht anwendbar sind, ist eine SUBSET-Bedingung der Form

```
SUBSET CONDITION c_non_Basic
```

zugeordnet, z.B. in (16.0997). Produktionen, die in Basic PEARL nur anwendbar sind, wenn eine einschränkende Bedingung B erfüllt ist, sind durch

```
SUBSET CONDITION c_non_Basic OR B
```

gekennzeichnet (z.B. in (16.1057)). Für die Definition von Basic PEARL nimmt man an, daß c_non_Basic der Wert FALSE zugeordnet ist. Dann ist von obigen Bedingungen nur die zweite erfüllbar. Für die PEARL-Definition sind alle SUBSET-Bedingungen erfüllt, da in diesem Fall c_non_Basic den Wert TRUE hat. Produktionen, die in Basic PEARL nicht ableitbare Symbole weiter ableiten, sind im allgemeinen keine SUBSET-Bedingungen zugeordnet:

```
sx_definition ::= sx_PRECEDENCE_def
```

wird durch SUBSET-Bedingung für Basic PEARL ausgeschlossen, die deshalb nicht erreichbare Produktion

```
sx_PRECEDENCE_def ::= ...
```

jedoch nicht.

4.2 Gültigkeitsbereiche - Bezeichneridentifikation

In höheren Programmiersprachen definieren die Gültigkeitsbereichsregeln, wie zu einem angewandten Auftreten eines Bezeichners die zugehörige Definition bestimmt wird. In der AG beschreiben Attribute und Attributregeln diesen "Informationsfluß" von den Definitionen zu den Anwendungsstellen. Wir wollen ihn hier entgegen seiner Richtung verfolgen: Das typische angewandte Auftreten eines Bezeichners repräsentiert die Produktion in (16.1110):

RULE sx_access_id ::= sx_id

Die zugehörige Definition wird mit Hilfe der rekursiven Funktion f_identify_id (12.0433) bestimmt. Sie liefert das erste Element aus einer Liste von Definitionen (2. Parameter), dessen Bezeichner-Komponenten mit dem gesuchten Bezeichner (1. Parameter) übereinstimmt. Falls es kein solches Element gibt, ist die Kontextbedingung in (12.0459) verletzt. Diese Liste von Definitionen ist der Wert des Attributs at_id_defs zum kleinsten die Anwendungsstelle umfassenden sx_PROBLEM_division, sx_block_Body, sx_detach_block_Body oder sx_loop_scope. (Dies ist die Bedeutung des INCLUDING-Konstrukts in (16.1117), vergl. Abschnitt 5.3.2.)

Die vier Symbole repräsentieren die Gültigkeitsbereiche in PEARL-Programmen. Als Beispiel für die Berechnung eines Attributs at_id_defs betrachten wir die Attributregel in (12.0257): Sie bestimmt den Wert von

sx_block_body.at_id_defs

durch Konkatenation von vier Teillisten: den lokalen Definitionen, den Markendefinitionen aus dem Anweisungsteil, den Parameterdefinitionen, falls der Block ein Prozedurrumpf ist, und den im kleinsten umfassenden Gültigkeitsbereich geltenden Definitionen. Da die zum Block globalen Definitionen in der Liste am Ende stehen, werden sie bei der Bezeichneridentifikation (f_identify_id) nur dann gefunden, wenn sie nicht durch eine gleichbezeichnete lokale Definition verdeckt sind.

Von dieser Identifikationstechnik wird in folgenden Fällen abgewichen: Globale Bezeichner in Spezifikationen werden anhand des Paares (Bezeichner, Modul-Qualifikation) in der Liste aller über Modulgrenzen hinweg gültigen Definitionen, die sx_PEARL_program zugeordnet ist, identifiziert (13.0050).

Operatorbezeichner können "überladen" sein, d.h. es können in einem Block mehrere gleich bezeichnete Operatordefinitionen gültig sein, die mit Hilfe der Operandentypen unterschieden werden (16.0592).

Die Selektoren von Verbund- oder DATION-Komponenten werden in den Komponentenlisten des Verbund- oder DATION-Typs identifiziert (16.1195).

Für Typbezeichner wird die oben beschriebene Identifikationstechnik nicht angewandt, da sie z.B. bei rekursiv definierten Typen wie in

TYPE t STRUCT [k REF t, ...]

zyklische Attributabhängigkeiten verursachen würde. Stattdessen wird jede Typdefinition "eindeutig umbenannt" (15.0481). Beim angewandten Auftreten eines Typbezeichners (sx_type_indicant, (15.0973)) wird zunächst nur diese neue eindeutige Benennung identifiziert. Dazu werden anstelle der Attribute at_id_defs die Attribute at_pre_defs benutzt. Erst bei "späterer" Analyse des Typs wird der neuen Benennug die Typdefinition zugeordnet.

4.3 Beschreibung von PEARL-Typen

Aussagen über Typen werden in PEARL zu zahlreichen Sprachelementen gemacht, z.B. zu Ausdrücken oder zu Typangaben in Definitionen. Attribute, welche solche Eigenschaften beschreiben, haben einen Wertebereich, der eine Abstraktion der statisch relevanten Eigenschaften von PEARL-Typen ist. Der Wertebereich wird in (15.0274) durch den Attributtyp tp_type als Vereinigung einiger Untertypen definiert. Der PEARL-Typ FIXED(5) wird z.B. durch tp_FIXED_type(5) beschrieben -

ein Verbundwert vom Typ tp_FIXED_type, dessen einzige Komponente den Wert 5 hat (vgl. Abschnitt 5.2). Untertypen, die keine weiteren unterscheidenden Merkmale haben, wie tp_SIGNAL_type in (15.0257), sind als einwertige Aufzählungstypen definiert. Zusammengesetzte Typen, z.B. tp_PROC_type in (15.0231), werden durch Verbunde definiert, deren Komponenten (z.B. s_params, s_RESULT) die Typeigenschaften beschreiben. Dazu wird häufig wieder auf den Wertebereich von tp_type bezug genommen, z.B. s_RESULT (siehe Abschnitt 5.2).

Der Begriff der "Zugriffsfunktion" ist in diesem Zusammenhang durch

```
TYPE tp_attr: STRUCT (s_INV : BOOL,
                      s_type: tp_type)
```

in (15.0147) abstrahiert. Der PEARL-Typ REF FIXED(5) wird also durch

```
tp_attr (FALSE, tp_FIXED_type(5))
```

abstrahiert und beschreibt "ganzzahlige Variable". In PEARL gehört zum Konzept "Zugriffsfunktion" der Begriff der "Variabilität": Mit Zugriffsfunktionen, deren Typ durch INV gekennzeichnet ist, darf der Wert des Bezugsobjekts nicht verändert werden, z.B. REF INV FIXED(5). Dies wird ausgedrückt durch die erste Komponente s_INV von tp_attr:

```
tp_attr (TRUE, tp_FIXED_type(5)).
```

PEARL-Deklarationen und -Spezifikationen, z.B.

```
DCL i FIXED(5)
```

führen Zugriffsfunktionen ein. Deshalb enthält die Beschreibung von Definitionen (tp_access_def in (11.0056)) eine Komponente vom Typ tp_attr, die in diesem Beispiel den Wert tp_attr(FALSE, tp_FIXED(5)) hat.

Der Wertebereich von tp_type umfaßt außer den Abstraktionen der PEARL-Typen auch solche, die in keinem syntaktisch korrekten Programm auftreten können, z.B.

```
tp_attr(FALSE,
        tp_attr(FALSE, tp_PROC_type(...))),
```

was REF REF PROC entspräche, und solche, die nur in bestimmtem Kontext zulässig sind (z.B. darf ein Interrupt nicht Prozedurergebnis sein). Die Einschränkung des Wertebereichs auf die im jeweiligen Kontext zulässigen PEARL-Typen definieren die Funktionen f_type_consistent und f_type_consistency_check in (15.1578). Den Kontext charakterisiert der Parameter p_restriction (z.B. sc_is_RESULT_tpye).

4.4 Typbestimmung in Ausdrücken

In PEARL werden wie in allen typisierten Programmiersprachen Aussagen über Typen von Ausdrücken gemacht. Da es in PEARL implizit angewandte Anpassungsoperationen (z.B. Typausweitung, Dereferenzieren, Aufruf ohne Parameter) gibt, unterscheidet man zwischen den Typen vor und nach Anwendung impliziter Anpassungen. Sie werden beschrieben durch die Attribute at_pre_type und at_post_type zu Symbolen der Ausdrucksgrammatik.

Das abgeleitete Attribut at_pre_type (siehe Abschnitt 5.1) wird z.B. für Bezeichner durch Bezeichneridentifikation (16.1110), für ein- und zweistellige Formeln durch Operatoridentifikation (16.0542), (16.0674) und für bedingte Ausdrücke durch Typabgleich (16.0172) bestimmt.

Das erworbene Attribut at_post_type (siehe Abschnitt 5.1) wird im Kontext bestimmt, in den der Ausdruck eingebettet ist. Wir unterscheiden hier zwei Situationen:

- Der Kontext bestimmt den Zieltyp vollständig und unabhängig von at_pre_type. Dies gilt z.B. für die rechte Seite von Zuweisungen (16.0968).
- Der Kontext bestimmt nur die "Typklasse" für at_post_type, in CONT sx_Basic_exp z.B. eine nicht näher spezifizierte Zugriffsfunktion (16.1032). Abhängig von at_pre_type wird daraus der vollständige Typ berechnet. Zur Beschreibung solcher Typklassen dient der Attributtyp tp_type_pattern (15.0348), dessen Wertebereich eine Vergröberung des Wertebereichs von tp_type ist.

Für jeden Ausdruck muß gelten: at_pre_type ist an at_post_type anpaßbar, wie es Attributbedingungen der Form

```
CONDITION
   f_coercible (sx_exp.at_pre_type,
                sx_exp.at_post_type)
```

z.B. in (16.0646) verlangen. Die Anpaßbarkeit ist in (15.2669) wie folgt erklärt: t1 = at_pre_type muß durch Anwenden einer (möglicherweise leeren) Sequenz von Anpassungsoperationen in einen Typ t2 transformierbar sein, so daß t2 mit t3 = at_post_type verträglich ist. Die Relation "verträglich" wird durch f_types_compatible (t3,t2) in (15.2001) definiert. Sie ist nicht symmetrisch: Der Typ t3 = at_post_type kann z.B. in Bezug auf die Zugriffsrechte (INV) restriktiver sein als t2: Für

```
   at_pre_type  = t1 = t2 = REF FIXED(5)
```

und

```
   at_post_type = t3 = REF INV FIXED(5)
```

gilt

```
   f_types_compatible (t3,t2) und
   f_coercible (t1,t3)
```

Die umgekehrte Richtung ist unzulässig, da die Zugriffsrechte erweitert würden. Die Verträglichkeitsbedingungen werden in Bezug auf die Zugriffsrechte mit "steigender Referenzstufe" verschärft (Parameter p_kind der Funktion f_type_compatibility_check).

5 Erläuterungen zu ALADIN

Die AG für PEARL ist in der Beschreibungssprache ALADIN (a Language for attributed definitions) formuliert. ALADIN verbindet Eigenschaften, die typisch sind für AG-Definitionen (Attributzuordnung, Attributregeln, Produktionen), mit Sprachelementen und Notationen aus höheren Programmiersprachen (Datentypen, Ausdrücke, Funktionen). In diesem Abschnitt geben wir eine kurze Charakterisierung der Sprache und erläutern anhand zahlreicher Beispiele aus dem Normentwurf die Bedeutung und Anwendung der Sprachelemente, von denen wir annehmen, daß sie nicht aus Programmiersprachen bekannt sind. Der Normentwurf selbst enthält eine Beschreibung aller in der PEARL-AG verwendeten Sprachelemente ([1], S. 13 ff); eine vollständige ALADIN-Definition ist in [5] enthalten.

ALADIN ist (wie PEARL, ALGOL68 oder PASCAL) eine streng typisierte Sprache.

Der Wertebereich jedes Attributs ist durch seinen Typ exakt definiert. Daraus und aus seinem Namen kann man häufig schon ohne Kenntnis der Attributregeln ersehen, welche Eigenschaft das Attribut beschreibt. Formeln und Ausdrücke müssen den Typregeln von ALADIN genügen. Der Zwang zur Einhaltung dieser Regeln vermeidet Beschreibungsfehler.

ALADIN ist (wie LISP) eine streng applikative, variablenfreie Sprache. Da Attribute statische Eigenschaften von Sprachelementen beschreiben, sind ihre Werte unveränderlich. Es widerspräche dem Grundprinzip attributierter Grammatiken, Attribute als Variable anzusehen. Als Konsequenz gibt es in ALADIN keine Elemente zur Ablaufsteuerung (z.B. Schleifen oder bedingte Anweisungen), sondern nur Ausdrücke, die z.B. durch Fallunterscheidungen strukturiert sind. Die Reihenfolge, in der Attributregeln angegeben sind, ist bedeutungslos und nicht als Ausführungsreihenfolge zu verstehen.

ALADIN sieht vor, daß die Objekte der AG (Attribute, Symbole, Typen, Funktionen, usw.) einen Namen haben, der in einer Definition explizit eingeführt wird. Durch suggestive Wahl der Namen wird die Lesbarkeit der AG wesentlich erhöht. Einige Konventionen zur Namensgebung werden in [1], S. 14 gegeben.

Der Kern einer AG in ALADIN ist eine Menge von Regeln, die jeweils eine kontext-freie Produktion mit den zugeordneten Attributregeln und Attributbedingungen zusammenfassen. Hinzu kommen Symboldefinitionen, die den Symbolen Attribute zuordnen, Typdefinitionen zur Beschreibung der Attribut-Wertebereiche, Definitionen von Funktionen und von Werten, die in den Attributregeln angewandt werden.

5.1 Attributzuordnung

Jedes Nichtterminal der kontext-freien Grammatik wird in einer Symboldefinition eingeführt, welche ihm Attribute bestimmter Typen zuordnet, z.B. in (16.0201).

```
NONTERM sx_exp:
        at_pre_type    : tp_type,
        at_post_type   : tp_type INH,
        at_value       : tp_value,
        at_PRECEDENCE  : tp_precedence;
```

In Attributregeln werden die Attribute in der Form sx_exp.at_pre_type notiert. Der Zusatz INH zum Attributtyp gibt an, daß es sich um ein erworbenes (engl.: inherited) Attribut handelt, mit dem ein Informationsfluß im Strukturbaum von oben (Wurzel) nach unten beschrieben wird - in diesem Beispiel der Typ des Ausdrucks, der vom umfassenden Kontext (z.B. rechte Seite einer Zuweisung) verlangt wird. Die nicht so gekennzeichneten Attribute sind abgeleitet (engl.: synthesized) und beschreiben einen Informationsfluß von unten nach oben im Strukturbaum.

Auch einige Terminale werden durch Symboldefinitionen eingeführt, um ihnen Attribute zuzuordnen, z.B. in (16.0300)

```
TERM sx_FIXED_const_denot :
        at_integer_value : INT
```

Die Attribute dieser Terminale nehmen eine Sonderstellung ein: Sie beschreiben eine Eigenschaft des Symbols, die sich unmittelbar aus seiner Aufschreibung ergibt, z.B. den Zahlwert 134 der Zahlkonstanten "0134". Es gibt keine Attributregeln für diese Attribute, ihre Werte werden als bekannt vorausgesetzt. (In einem PEARL-Übersetzer werden sie durch die lexikalische Analyse der Symbole bestimmt.)

5.2 Attributtypen

Durch den Typ eines Attributs wird sein Wertebereich festgelegt. Er kann als Abstraktion der Eigenschaft verstanden werden, die das Attribut beschreibt. ALADIN hat neben einigen elementaren Typen mächtige Typkonstruktoren, die es erlauben, auch komplexe Eigenschaften (z.B. die Definitionen eines Blocks) strukturiert zu beschreiben. Mit Hilfe von Typkonstruktoren werden in Typdefinitionen neue, für die Anwendung passende Typen eingeführt (vergleichbar mit den MODE-Deklarationen in ALGOL68 oder den Typdefinitionen in PASCAL). Im folgenden erläutern wir die elementaren Typen und die Typkonstruktoren zusammen mit den wichtigsten der jeweils anwendbaren Operationen. (Wertvergleiche sind für alle Typen definiert und werden hier nicht mehr erwähnt.)

Im Normentwurf werden die elementaren Typen INT, BOOL und SYMB benutzt. Der Wertebereich des Typs SYMB umfaßt die terminalen Symbole der Sprache. Typische Anwendungen von SYMB sind Attribute von Terminalen, welche die Identität des Symbols beschreiben, z.B. sx_id.at_id für Bezeichner (13.0108) oder sx_op_token.at_op_token für Operatoren (16.0305). Für Werte vom Typ SYMB können Fallunterscheidungen angewandt werden, z.B. zur Identifikation von Standard-Operatoren in (16.2275).

Neben den Werten der Grundtypen werden neue elementare Werte (wie in PASCAL) durch Aufzählung ihrer Namen eingeführt, z.B. in (11.0068):

```
TYPE tp_definition_kind:
        (sc_length_or_prec_def,
           ...
         sc_TASK_del)
```

(Werte dieses Typs klassifizieren PEARL-Definitionen, um für Basic PEARL die Reihenfolgebedingung zu prüfen.) Außer Fallunterscheidungen sind Ordnungsrelationen möglich, z.B. in (12.0369). Die Ordnung der Werte wird durch ihre Reihenfolge in der Typdefinition festgelegt.

Aufzählungstypen können zur Mengenbildung verwendet werden, z.B. in (17.0116) mit (17.0136):

```
TYPE tp_from_to : (sc_from, sc_to)
TYPE tp_dir_set : SETOF tp_from_to .
```

(Werte dieses Mengentyps geben an, in welchen Richtungen eine Datenstation betrieben werden kann.) Es sind die

Mengenoperationen (+, -, *, <=, >=, IN) definiert. Mengenwerte werden durch Aufzählung der Elemente mit vorangestelltem Typnamen angegeben, z.B. tp_dir_set (sc_to); tp_dir_set() gibt die leere Menge an.

Die Wertebereiche von INT und von Aufzählungstypen können auf Ausschnitte eingeschränkt werden, z.B. in (15.0228)

```
TYPE tp_posint: [c_zero : MAXINT]
```

Der Typkonstruktor STRUCT bildet (wie in ALGOL68) zusammengesetzte Wertebereiche für Verbunde, z.B. in (11.0056).

```
TYPE tp_access_def:
     STRUCT (s_id    : SYMB,
             s_global: SYMB,
             s_value : tp_value,
             s_attr  : tp_attr),
```

mit dem PEARL-Definitionen (einer bestimmten Klasse) abstrakt beschrieben werden. Die Komponentennamen (s_id, s_global, usw.) werden in Selektionsoperationen angewandt, z.B. in (12.0338) p_access.s_id. Verbundwerte werden durch die Werte der Komponenten in der Reihenfolge, welche die Typdefinition vorschreibt, mit vorangestelltem Typnamen angegeben, z.B. in (13.1215).

```
tp_access_def (HEAD (p_ids),
               p_global,
               HEAD (p_value),
               p_attr)
```

Der Typkonstruktor UNION vereinigt (wie in ALGOL68) mehrere Wertebereiche zu einem neuen, z.B. in (11.0103).

```
TYPE tp_id_def:
     UNION (tp_access_def,
            tp_OPERATOR_def,
            tp_TYPE_def)
```

(Hier werden Beschreibungen verschiedener, bezeichneter PEARL-Definitionen zusammengefaßt.) Werte dieses Typs werden durch implizite Anpassung aus Werten der Untertypen gebildet. Mit Fallunterscheidungen kann festgestellt werden, ob ein Wert des vereinigten Wertebereichs zu einem der Teilbereiche gehört, z.B. in (12.0532).

```
CASE HEAD (p_l) OF
IS tp_access_def:
    IF THIS.s_global /= c_not_global
    ...
OUT ...
ESAC
```

Der Standardname THIS ist eine Abkürzung für den untersuchten Ausdruck zwischen CASE und OF. Häufig wird stattdessen auch ein frei gewählter Name explizit eingeführt. Das obige Beispiel würde dann lauten

```
CASE ce_acc: HEAD (p_l) OF
IS tp_access_def:
   IF ce_acc.s_global /= c_not_global
    ...
OUT ...
ESAC
```

Die Prüfung der Zugehörigkeit zu einem Untertyp kann ebenso durch einen Typtest (IS-Operator) formuliert werden:

```
IF HEAD (p_l) IS tp_access_def
THEN IF HEAD (p_l) QUA tp_access_def
        .s_global /= c_not_global
             ...
ELSE ...
```

Der QUA-Operator gibt an, daß der davor stehende Ausdruck als Wert des Untertyps tp_access_def benutzt wird. In den oben angegebenen Fallunterscheidungen wird er an entsprechender Stelle impliziert.

Der Typkonstruktor LISTOF definiert als Wertebereich Sequenzen von Werten eines Grundtyps (lineare Listen), z.B. in (11.0108).

```
TYPE tp_id_def_list : LISTOF tp_id_def
```

Diesen Typ hat z.B. das Attribut sx_block.at_id_defs, das alle gültigen benannten Definitionen beschreibt. Auf Listenwerte sind vordefinierte Funktionen, wie HEAD, TAIL, FRONT, LAST, LENGTH und EMPTY (siehe [1], S. 23) und die Konkatenation (+) anwendbar, z.B. in (12.0257).

```
sx_block_body.at_id_defs :=
    sx_definitions_ety.at_id_defs
  + sx_statements_ety.at_label_defs
    ...
```

Das Durchlaufen der Elemente einer Liste (z.B. zur Suche eines Elements in [12.0433]) wird durch rekursive Funktionen beschrieben.

Listenwerte gibt man wie Verbundwerte durch Aufzählung ihrer Elemente mit vorangestelltem Typnamen an, z.B.

```
tp_id_def_list(a,b,c).
```

Die leere Liste wird entsprechend notiert: tp_id_def_list(). Einige Listentypdefinitionen enthalten eine KEY-Angabe, die nur im Zusammenhang mit der vordefinierten ALADIN-Funktion SELECT_BY_KEY Bedeutung hat (siehe [1], S. 22/23).

Die Typbildungsregeln für ALADIN sind im wesentlichen aus ALGOL68, PASCAL und LISP übernommen. Ein prinzipieller Unterschied ergibt sich jedoch aus dem streng applikativen Charakter von ALADIN: In höheren Programmiersprachen sind rekursiv definierte Typen nur unter Verwendung von Referenzen zulässig, z.B. ein Binärbaum in ALGOL68:

```
MODE Knoten=UNION (Blatt,Zweig);
MODE Zweig =STRUCT (REF Knoten links,
                                rechts,
                    INT opr);
MODE Blatt =STRUCT (INT wert);
```

Einerseits enthält ALADIN kein Referenzkonzept - es würde nicht zur Variablenfreiheit passen. Andererseits besteht keine Veranlassung, in einer Beschreibungssprache rekursiv definierte Typen auszuschließen, solange ihr Wertebereich endliche Werte enthält. Das obige Beispiel lautet deshalb in ALADIN:

```
TYPE Knoten: UNION (Blatt,Zweig);
TYPE Zweig : STRUCT (links, rechts: Knoten,
                     opr: INT);
TYPE Blatt : STRUCT (Wert: INT);
```

(Für die Implementierung solcher Strukturen wendet man natürlich das Referenzkonzept an.)

Ein Typ wie

```
TYPE seq: STRUCT (Wert:INT, nachf:seq)
```

ist auch in ALADIN unzulässig.

Typische Anwendungen rekursiver Typdefinitionen sind die Wertebereiche, welche die PEARL-Typen beschreiben (tp_type in [15.0274]).

5.3 Regeln

Der Kern einer AG besteht aus den Regeln, die eine kontext-freie Produktion mit den zugehörigen Attributregeln und -bedingungen zusammenfaßt.

Die Gliederung der Regeln sei am Beispiel in [16.0994] erläutert:

```
RULE sx_assignment ::=          % RULE 2
     sx_destination ':=' sx_assignment
SUBSET CONDITION c_non_Basic
STATIC
   CONDITION ...
   sx_assignment[2].at_post_type := ...
   sx_assignment[1].at_pre_type  := ...
   CONDITION ...
   sx_assignment[1].at_value := ...
DYNAMIC
   % ...
END
```

Im ersten Abschnitt steht die kontext-freie Produktion in erweiterter Backus-Naur-Form notiert. (Die Erweiterungen erläutern wir unten.) Der Kommentar % RULE 2 gibt an, daß es sich um die zweite alternative Produktion mit gleicher linker Seite (sx_assignment) handelt. Der zweite Abschnitt (SUBSET) enthält eine Bedingung, welche in diesem Fall die Anwendung der Produktion für Basic PEARL verbietet (siehe Abschnitt 4.1).

Der dritte Abschnitt (STATIC) enthält Attributbedingungen (eingeleitet durch CONDITION) und Attributregeln, welche den Wert des links vom Zuweisungszeichen angegebenen Attributs bestimmen. Die rechte Seite einer Attributregel ist ein Ausdruck, der von anderen Attributen abhängen kann. Alle in der Regel vorkommenden Attribute beziehen sich auf Symbole der kontext-freien Produktion (Ausnahme siehe unten). Ein Symbol, das mehrfach in der Produktion vorkommt, wird in Attributbenennungen indiziert angegeben. Die Menge der Attributregeln ist im folgenden Sinne immer vollständig und eindeutig: Es gibt

genau eine Attributregel zu jedem abgeleiteten Attribut des Symbols auf der linken Seite der Produktion (sx_assignment[1].at_pre_type und sx_assignment[1].at_value) und zu jedem erworbenen Attribut jedes Symbols auf der rechten Seite der Produktion (sx_assignment[2].at_post_type).

Der DYNAMIC-Abschnitt enthält Kommentare mit Angaben zur dynamischen Semantik (meist "Laufzeitbedingungen" oder Verweise auf Abbildungen mit Netzen).

Außer der kontext-freien Produktion kann jeder der Abschnitte fehlen.

Im folgenden erläutern wir zunächst die Erweiterungen der BNF zusammen mit den Konsequenzen für die Attributregeln und geben dann die Bedeutung einiger abkürzender Schreibweisen in Attributregeln an.

5.3.1 Schreibweise der Produktionen – Auswirkungen auf Attributregeln

Symbolfolgen, die in der Anwendung einer Produktion optional sind, werden durch [,] geklammert, z.B. in (14.0449).

```
RULE sx_cond_statement ::=
     'IF' sx_BIT1_exp
     'THEN' sx_statements_ety
    ['ELSE' sx_statements_ety]
     'FIN'
     ...
```

Wird in Ausdrücken auf ein Attribut eines solchen Symbols Bezug genommen, so muß sichergestellt sein, daß der Wert des Ausdrucks auch dann definiert ist, wenn das Symbol fehlt. Dies geschieht durch einen bedingten Ausdruck, in dem das optionale Symbol anstelle der Bedingung steht, z.B. in (14.0463):

```
IF sx_statements_ety[2]
THEN at_label_defs
ELSE tp_id_def_list()
FI
```

Im THEN-Teil wird der Attributname ohne das optionale Symbol angegeben. Der ELSE-Teil gibt den Wert des bedingten Ausdrucks an für den Fall, daß das Symbol fehlt.

Logische Ausdrücke der Form Symbolname IS THERE geben an, ob in einer Anwendung der Produktion das optionale Symbol tatsächlich vorhanden ist. In (15.0873)

```
RULE sx_REF_type ::=
     'REF' [sx_INV] sx_type
           ...
STATIC
     sx_REF_type.at_type ::=
        tp_attr (sx_INV IS THERE,
                 sx_type.at_type)
END
```

entscheidet das Vorhandensein des INV-Symbols über die Variabilitätseigenschaft des Referenztyps.

Die übrigen Erweiterungen der BNF beschreiben verschiedene Formen der syntaktischen Wiederholung: Der * in (14.0093)

```
RULE sx_statement ::=
     sx_label_dcl* sx_unlabelled_statement
```

bedeutet, daß eine Anweisung mit beliebig vielen (auch keiner) Markendefinition beginnen kann. Ein + anstelle des * würde mindestens eine Markendefinition verlangen. Soll eine Sequenz von mehreren Symbolen wiederholt werden, so sind diese in (...)+ bzw. (...)* eingeschlossen, z.B. in (13.0678)

```
RULE sx_PROC_dcl ::=
     (sx_id ':')+ 'PROCEDURE' ...
```

Eine korrekte Ableitung wäre z.B.

```
P1 : P2 : P3 : PROCEDURE ...
```

In (13.1038)

```
RULE sx_ids ::=
     '(' (sx_id // ',' ) ')'
```

wird eine nicht leere Sequenz von Bezeichnern beschrieben, die durch ',' getrennt sind. Korrekte Ableitungen wären z.B. (A) oder (A,B) oder (A,B,C) ...

Nimmt man auf Attribute von Symbolen Bezug, die in einer der beschriebenen Weisen als "wiederholbar" gekennzeichnet sind, so meint man nicht den Wert eines einzelnen Attributs, sondern die Liste der

Attributwerte aller Symbole, die aus der Wiederholung abgeleitet wurden. Wird z.B. sx_ids nach obiger Produktion zu (A,B,C) abgeleitet, so hat der Ausdruck sx_id.at_id in (13.1042) den Wert

tp_symb_list('A', 'B', 'C').

5.3.2 Abkürzende Schreibweisen in Attributregeln

Attributregeln, die einen "Wert-Transport" mit gleich benannten Attributen beschreiben, werden durch TRANSFER abgekürzt. Z.B. in der "Kettenproduktion" (16.0927)

```
RULE sx_prim_exp ::= sx_basic_exp
STATIC TRANSFER
END
```

steht TRANSFER für die drei Attributregeln

```
sx_prim_exp.at_pre_type :=
   sx_basic_exp.at_pre_type;
sx_prim_ex.at_value :=
   sx_basic_exp.at_value;
sx_basic_exp.at_post_type :=
   sx_prim_exp.at_post_type;
```

Jeweils eines der beiden Attribute gehört zum Symbol auf der linken Seite der Produktion. Die "Transportrichtung" wird durch die Attributklasse bestimmt: at_pre_type und at_value sind abgeleitet - "Transport nach oben im Baum", at_post_type ist erworben - "Transport nach unten im Baum".

In der Form TRANSFER at_value, z.B. in (16.0947), wird die Abkürzung auch für einzelne Attributregeln neben ausgeschriebenen Attributregeln benutzt.

Um einen "Attributwert-Transport" über viele syntaktische Zwischenstufen zu beschreiben, wird mit dem INCLUDING-Konstrukt auf ein Attribut eines Symbols Bezug genommen, das nicht in der Produktion vorkommt. Es ist ein Symbol, aus dem die linke Seite der Produktion (direkt oder indirekt) abgeleitet ist, z.B. in (13.0538) zur Regel in (13.0527):

```
RULE sx_GLOBAL_attr ::=
 'GLOBAL' ['(' sx_GLOBAL_qual ')']
          ...
STATIC
 sx_GLOBAL_attr.at_GLOBAL_qual :=
  IF sx_GLOBAL_qual
  THEN at_GLOBAL_qual
  ELSE INCLUDING sx_MODULE.at_GLOBAL_qual
  FI
END
```

Im folgenden Beispiel werden zwei Alternativen für die Herkunft des Attributwerts angegeben (13.1123)

```
RULE sx_FIXED_precision_def ::=
     'LENGTH' 'FIXED'
     '('sx_FIXED_const_denot')'
SUBSET
  CONDITION c_non_basic OR
  ((INCLUDING
      (sx_PROBLEM_division.at_context,
       sx_block_body.at_context))
    = sc_PROBLEM_division)
   ...
```

Es wird dann das im Strukturbaum auf dem Wege nach oben nächste der beiden Symbole gewählt. Ist das z.B. ein sx_block_body, dessen Attribut at_context immer den Wert sc_block_body hat, so ist diese SUBSET-Bedingung verletzt (d.h. LENGTH-Definitionen dürfen in Basic PEARL nur auf der Ebene der PROBLEM-Division stehen).

Auf Attribute aus Teilbäumen, die aus der rechten Seite der Produktion abgeleitet sind, wird mit dem CONSTITUENT-Konstrukt Bezug genommen. Da es in der PEARL-AG nur an wenigen Stellen vorkommt, sei hier auf die Beschreibung in [1], S.21 verwiesen.

In den Beispielen haben wir bisher nur solche Attributbedingungen vorgestellt, die innerhalb von Regeln gleichrangig neben Attributregeln stehen. In einigen Attributregeln und Funktionsrümpfen stehen Ausdrücke der Form (a CONDITION b), welche den Wert a haben und gleichzeitig die Kontextbedingung b spezifizieren, z.B. in (15.2944).

```
FUNCTION f_deproceduring
  (p_PROC_type : tp_PROC_type) tp_type :
   f_expand_type
      ((p_PROC_type.s_RESULT
        CONDITION
        EMPTY(p_PROC_type.s_params)))
```

Die Anpassungsoperation "Deprozedurieren" liefert den Ergebnistyp der Prozedur unter der Voraussetzung, daß keine formalen Parameter existieren. Im ELSE-Teil der Attributregel in (12.0266)

```
sx_block_body.at_FIXED_precision :=
  IF ...
  THEN ...
  ELSE HEAD
     ((f_select_FIXED_precision
       (sx_definitions_ety.at_pre_defs)
        CONDITION LENGTH (IT) = 1))
  FI
```

wird die FIXED-Precision eines Blocks gemäß der ersten FIXED-Precision-Definition im Definitionsteil bestimmt und gleichzeitig gefordert, daß es keine weitere gibt. IT ist eine Abkürzung des Ausdrucks, der vor CONDITION steht. In einigen Fallunterscheidungen, z.B. in (13.0670) findet man als Wert eines Falls Konstrukte der Form (a CONDITION FALSE). Sie drücken aus, daß in PEARL-Programmen dieser Fall unzulässig ist.

Literatur

[1] DIN 66253 Teil 2, Informationsverarbeitung, Programmiersprache PEARL, Full PEARL, Beuth Verlag Berlin 1980

[2] DIN 66253 Teil 1, Informationsverarbeitung, Programmiersprache PEARL, Basic PEARL, Beuth Verlag Berlin 1978

[3] Full PEARL Language Description, Gesellschaft für Kernforschung mbH, Karlsruhe, PDV-Bericht KfK-PDV 130, 1977

[4] S. Heilbrunner, M. Schmitz: Zur attributierten Grammatik von PEARL (in dieser PEARL-Rundschau)

[5] U. Kastens: ALADIN - Eine Definitionssprache für attributierte Grammatiken. Fak. f. Informatik, Universität Karlsruhe, Bericht 7/79

[6] U. Kastens: Ordered Attributed Grammars. Acta Informatica 13, 1980, 229-256

[7] U. Kastens, E. Zimmermann: GAG - A Generator Based on Attributed Grammars. Fak. f. Informatik, Universität Karlsruhe, Bericht 16/81

[8] D.E. Knuth: Semantics of context-free languages. Math. Syst. Th. 2 (1968), 127-145. Korrekturen in: Math. Syst. Th. 5 (1971), 95 ff