Fix point internal hierarchy specification for context free grammars

VASILE CRĂCIUNEAN, RALF FABIAN, DANIEL HUNYADI, EMIL MARIN POPA
Department for Computer Sciences and Economic Informatics
"Lucian Blaga" University of Sibiu
Ion Rațiu Street, no. 5-7, Sibiu
ROMANIA

Abstract: - In the particular case when a context free grammar is used as a model for a computation system, each nonterminal will be naturally associated to a meaning, i.e. a type, and every generation rule represents an operation in the computations system. Thus, the derivation tree (syntax tree) of this grammar yields a hierarchic structure of types as well as a hierarchic structure of system operations. Consequently, on each hierarchic level there exists a set of types and a set of operations, namely a language. Recursive specifications are a useful tool for representing and studying this kind of language hierarchies. As we show in this paper, the recursive specifications are a kind of constructor for these languages.

Key-Words: - computation system, fix point, recursive specification, internal hierarchy, type hierarchies

1 Introduction

With this paper we address the issue of generation and specification methods for system design based on recursive specifications, considering context free grammars as the basics milestone in current modern programming languages.

Therefore we link the properties of Kleene's fix point theory to internal hierarchy specifications of context free grammars. This leads to the possibility of specify abstraction levels for application generation that relay on previous (older) levels. In other words, there are unlimited possibilities to extend the existing specifications.

We assume familiarity with basic definitions and results of formal language theory. First we stress out some important results from set theory, needed in the formal modeling of recursive systems. Further we present an algorithm that determines the types and an algorithm for the proper hierarchy of types. At the end we finish with a simple example showing the immediate results of the presented theory.

2 Preliminary concepts and notions

In this paragraph we recall some basic concepts and notions related to our further discussions.

From the set theory we now that posets (partial ordered set: $\le$ is reflexive, antisymmetric and transitive) have the properties.

- y is an *upper bound* of a subset Z of a poset $(P,\le)$ iff $y \in P$ and, for all $z \in Z$, $z \le y$;
- y is the *least element* ($\perp$) of Z iff $y \in Z$ and, for all $z \in Z$, $y \le z$;
- y is a *maximal element* of Z iff $y \in Z$ and there is no $z \in Z$ such that $z \ne y$ and $y \le z$.

The notion of lower bound, greatest element, and minimal element receive dual definitions (i.e. definitions obtained by replacing "$\le$" by "$\ge$").

- y is the *supremum*, $\vee Z$, of Z iff y is an upper bound of Z and y is the least of the upper bounds of Z;
- y is the *infimum*, $\wedge Z$, of Z iff y is a lower bound of Z and y is the greatest of the lower bounds of Z.

Definition 2.1. A partial ordered set $(P,\le)$ is called ***domain*** if it has one least element and if any ascending sequence over P has an upper bound in P.

Definition 2.2. Let $(D_1,\le_1),...,(D_n,\le_n)$, $n>0$ be domains. Then the ***product domain*** of n domains is a domain $(D,\le)$, where:

- $D = D_1 \times ... \times D_n$ and
- $(x_1,x_2,...,x_n) \le (y_1,y_2,...,y_n)$ iff $x_i \le_i y_i, i=\overline{1,n}$, $(x_1,x_2,...,x_n),(y_1,y_2,...,y_n) \in D_1 \times ... \times D_n$.

Definition 2.3. Let D be a domain. A ***recursive specification*** over D is a total function $\psi : D \to D$ such that $\psi(\perp) \le \psi^2(\perp) \le \psi^3(\perp) \le ...$. In this conditions the sequence $\psi(\perp) \le \psi^2(\perp) \le \psi^3(\perp) \le ...$ is called ***Kleene sequence*** for ψ .[1],[10]

Definition 2.4. An element $f_\psi \in D$, defined by

$$f_\psi = \bigvee_{k=1}^{\infty} \psi^k(\perp), \tag{1}$$

is called ***Kleene semantic*** of ψ .[1]

Remark: From the domain definition, this supremum exists.

Lemma 2.1. If $(D,\leq)$ is a domain and $\psi:(D,\leq)\to(D,\leq)$ is monotone then ψ is a ***recursive specification***. (Proof: see [1].)

Definition 2.5. Let $(P,\leq)$ be a partial ordered set and $\psi:P\to P$ a total function. A ***fixed point*** of ψ is an element $f\in P$ that verifies $\psi(f)=f$. The ***least fix point*** of ψ (if it exists) is the least element from the set of fix points. [1],[8]

Theorem 2.1. Let $(P,\leq)$ be a partial ordered set and $\psi:(P,\leq)\to(P,\leq)$ a monotone function. If there exists $f=\vee\{h \mid h\leq\psi(h)\}$, then it is a ***fix point of*** ψ. (Proof: see [1].)

Definition 2.6. Let $(D,\leq)$ and $(E,\leq')$ be domains. A monotone mapping $\psi:(D,\leq)\to(E,\leq')$ is continuous if it preserves the leases upper bounds of the increasing sequences, namely $\psi(\vee f_n)=\vee(\psi(f_n))$.[1]

An important result, on which this paper relies on, is the following theorem, known as Kleene's fix point theorem.

Theorem 2.2. (***Kleene's fix point theorem***) Let $(D,\leq)$ be a domain and $\psi:(D,\leq)\to(D,\leq)$ a continuous function. Then Kleen's semantic $f_\psi=\bigvee_{k=1}^{\infty}\psi^k(\perp)$ is the least fix point of ψ .([1],[8])

Example 2.1. We denote $Pfn(X,Y)$ the set of all partial functions $f:X\to Y$. Then $(Pfn(X,Y),\leq)$ is a domain, where the relation $\leq$ is defined as follows:

$f\leq g \Leftrightarrow DD(f)\subseteq DD(g)$ and

$g(x)=f(x)$, $(\forall)x\in DD(f)$.

Thus, if $f_1\leq f_2\leq\ldots\leq f_j\leq\ldots$ we define $\vee f_i$ as

$$DD(\vee f_i)=\bigcup_{i=0}^{\infty}DD(f_i)$$

$(\vee f_i)(x)=f_k(x)$, $(\forall)$ k such that $x\in DD(f_k)$.

The function $\vee f_i$ is well defined because if $x\in DD(f_j)\cap DD(f_k)$, then $f_j(x)=f_k(x)$.

In this circumstances it can easily been seen that the Kleene sequence $\psi(\perp)\leq\psi^2(\perp)\leq\psi^3(\perp)\leq\ldots$ is indeed an increasing sequence in $Pfn(X,Y)$, and Kleene's semantic

- $DD(\vee f_\psi)=\bigcup_{k=1}^{\infty}DD(\psi^k(\perp))$,
- $f_\psi(x)=\psi^k(\perp)(x)$, $(\forall)$ k, $x\in DD(\psi^k(\perp))$

satisfies the relation $f_\psi=\vee\psi^i(\perp)$.

Remark: The relation $f\leq g$ shows us that g offers at least as much information than *f* dose.

3 Fix point and formal languages

In this part we focus on context free grammars (type 2 grammars in Chomsky's classification). Fist, we show on an example, that the language generated by a context free grammar (CFG) can be obtained from the smallest fix point of a well chosen recursive specification.

Example 3.1. Suppose the following simple CFG: $G=(V_N,V_T,S,P)$ where

P= { *S*→*A*
S→*B*
A→a*A*b
A→ab
B→b*B*a
B→ba
}

$V_N=\{S,A,B\}$, $V_T=\{a,b\}$ and S is the grammar axiom (starting symbol).

First we rewrite the production rules in the following way

S→*A*+*B*
A→*aAb*+*ab*
B→*bBa*+*ba*

where "+" denotes the union.

Then we define the recursive specification:

$\psi:(2^{V_T^*})^3\to(2^{V_T^*})^3$

$\psi(S, A, B)=(A+B$, aAb+ab, bBa+ba)

Applying Kleene's theorem to determine the leases fix point for the above chosen ψ (the fact that ψ is continues will be shown later) and computing the Kleene sequence we have:

$\psi^0(\perp,\perp,\perp)=(\varnothing,\varnothing,\varnothing)$

$\psi^1(\perp,\perp,\perp)=\psi(\varnothing,\varnothing,\varnothing)=\{\varnothing,ab,ba)\}$

$\psi^2(\perp,\perp,\perp)=\psi(\varnothing,ab,ba)=\{ab+ba,a^2b^2+ab,$
$b^2a^2+ba\}$.

It can be easily seen that an induction over m proofs that $\psi^m(\perp)=(\{a^jb^j \mid 1\leq j\leq m\}\cup\{b^ja^j \mid 1\leq j\leq m\},$
$\{a^jb^j \mid 1\leq j\leq m\},\{b^ja^j \mid 1\leq j\leq m\})$

Thus, the sequence $\psi^m(\perp)$ is truly an increasing sequence and we have

$$\bigvee_{m\geq 0}\psi^m(\perp)=(L(G),\{a^jb^j \mid j\geq 1\},\{b^ja^j \mid j\geq 1\})$$

(here $L(G)$ means the language generated by the grammar G).

However, $L(G)$ is the first component of the least fix point of ψ. More generally speaking, for

$k = 1,2,3,...$ the k-th component of $\underset{m\geq 0}{\vee} \psi^m(\perp)$ is $\{w \mid w \in X^* \text{ and } v_k \overset{*}{\Rightarrow} w\}$, where $v_k \in \{S, A, B\}$.

A closer look on the above relations will reveal some other information to. Let $\overset{j}{\Rightarrow}$ denote the power j of the relation $\Rightarrow$, i.e. $w \overset{j}{\Rightarrow} w'$ means that w derives (generates) w' in j steps if there exists a sequence $w_1, w_2, ..., w_j$ such that $w = w_1 \Rightarrow w_2 \Rightarrow ... \Rightarrow w_{j-1} \Rightarrow w_j = w'$; $w \overset{0}{\Rightarrow} w'$ means that $w = w'$.

Hence, the k-th element of $\psi^m(\perp)$ is

$$\{w \mid w \in X^* \text{ and } v_k \overset{j}{\Rightarrow} w', \text{for } j \leq m\}. \qquad (2)$$

The representation $\{w \mid w \in X^* \text{ and } v_k \overset{j}{\Rightarrow} w, j \leq m\}$ is tricky, as the following example shows.

Example 3.2. Considering a grammar with a single variable and the productions *S→a+aSa+SbbS*, then, this implies $\psi : 2^{X^*} \to 2^{X^*}$ and $\psi(\perp)=a+aSa+SbbS$. Thus,

$\psi^0(\perp) = \varnothing$

$\psi^1(\perp) = \{a\}$

$\psi^2(\perp) = \{aaa, abba\} \cup \{a\}$

But $\psi^2(\perp)$ is not acceptable since the shortest derivation for the word *abba* is $v_1 \Rightarrow v_1 bbv_1 \Rightarrow abbv_1 \Rightarrow abba$, which needs 3 steps (in stead of les then 2). Anyway, looking at the derivation tree, depict in figure 1, for the word *abba*, we see that he is of height 2.

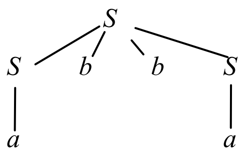

Figure 1. Derivation tree for example 3.2

In other words, if we admit parallel substitutions (all the variables of a sequence can be replaced during a single step), then our word is indeed derivable in two steps: $S \Rightarrow SbbS \Rightarrow abba$. This is a model of the semantic of "total call" by ".", due the fact that all variables are replaced by each step; however this model is nondeterministic because any production from the set of productions can be chosen to replace every occurrence of a variable.

4 Internal hierarchy of context free languages

The generating grammar for a context free language offers a internal language hierarchy. By this means, we consider the language semantics equal to the calculus system obtained in the following way:

- every nonterminal symbol of the grammar represents the name of a set of elements called ***type***;
- every production represents a ***heterogeneous operation***, namely, if $A \to \alpha \in P$ and $\alpha = s_0 A_1 s_1 A_2 ... s_n A_n s_{n+1}$, then the heterogeneous operation associated to this production is denoted by $s_0 s_1 ... s_n s_{n+1}$ and operates on the sets $A_1, A_2, ..., A_n$ and produces a result of type A, i.e. $(s_0 s_1 ... s_n s_{n+1}) : A_1 \times A_2 \times ... \times A_n \to A$.

If $p = A \to \alpha \in P$ then $t(p)$ denotes the type of p, $t(p) = A$; $d(p)$ denotes the domain of p, $d(p) = \{A_1, A_2, ..., A_n\}$; $s(p)$ denotes the state word (or symbol) of p, $s(p) = s_0 s_1 ... s_n s_{n+1}$, and $m(p)$ the arity of p, $m(p) = n$.

Given a context free language $G = (V_N, V_T, S, P)$, we consider a hierarchy of nonterminal symbols build as follows:

$$V_N^0 = \{A \in V_N \mid A \to \alpha \in P, \alpha \in V_T^*\} \qquad (3)$$

$$V_N^{i+1} = V_N^i \cup \{A \in V_N \mid A \to s_0 A_1 s_1 A_2 ... s_n A_n s_{n+1} \in P \text{ and } A_1, A_2, ..., A_n \in V_N^i\} \qquad (4)$$

The next algorithm determines the types hierarchy for a context free grammar $G = (V_N, V_T, S, P)$.

Algorithm 4.1.

Input: A CFG $G = (V_N, V_T, S, P)$.

Output: The set of types V_N^j.

Method: consists in building a sequence of sets $\{V_N^i \mid i \in \mathbb{N}\}$ that comply with the property $V_N^0 \subset V_N^1 \subset ... \subset V_N^{k-1} = V_N^k = ...$. Thus, V_N^k will contain all the nonterminal symbols that can generate words over V_T^*. Obviously, if $S \notin V_N^k$ the language is equal to the empty set.

```
00 START
01 $i := 0$
02 $V_N^0 = \{A \in V_N \mid A \to \alpha \in P, \alpha \in V_T^*\}$
03     DO UNTIL ($V_N^i = V_N^{i-1}$)
           $V_N^{i+1} = V_N^i \cup \{A \in V_N \mid$
04                 $A \to s_0 A_1 s_1 A_2 ... s_n A_n s_{n+1} \in P$
                   $\text{and } A_1, A_2, ..., A_n \in V_N^i\}$
```

```
05        $i := i+1$
06    ENDDO
07 STOP
```

Property 4.1. For any given context free grammar $G=(V_N,V_T,S,P)$, there exists a finite number k of hierarchic type levels. (The complete proof can be found in [3])

Example 4.1. We consider now as example, a classic CFG, namely the grammar that generates arithmetic expressions:

$G=(V_N,V_T,S,P)$, where

$V_N=\{E, T, F\}$;

$V_T=\{a,*,+,(,)\}$;

$S=E$;

$P=\{F\to a,\ T\to F, T\to T*F,\ E\to T,\ E\to E+T,\ F\to (E)\}$

We obtain the following inner hierarchy of types:

$V_N^0=\{F\}$

$V_N^1=\{F, T\}$

$V_N^2=\{F, T, E\}$

$V_N^3=\{F, T, E\}$

Building an inner hierarchy of productions leads us to:

$P^1=\{p\in P \mid d(p)=\varnothing \text{ and } t(p)\subseteq V_N^0\}$,

$P^{i+1}=P^i \cup \{p\in P \mid d(p)\subseteq V_N^{i-1} \text{ and } t(p)\subseteq V_N^i\}$

For a given context free grammar $G=(V_N,V_T,S,P)$, the following algorithm determines the corresponding hierarchy of types.

Algorithm 4.2.

Input: A context free grammar $G=(V_N,V_T,S,P)$.

Output: The set of productions P^i.

Method: consists of building a sequence of sets $\{P^i \mid i\in\mathbb{N}\}$ with the property $P^1\subset P^2\subset\ldots\subset P^{k-1}=P^k=\ldots$ The set P^k will contain all useful productions of the grammar G.

```
00 START
01 $i := 0$
02 $P^1=\{p\in P \mid d(p)=\varnothing \text{ and } t(p)\subseteq V_N^0\}$
03    DO UNTIL ($P^i = P^{i-1}$)
04        $P^{i+1}=P^i \cup \{p\in P \mid d(p)\subseteq V_N^{i-1}$
                 $\text{and } t(p)\subseteq V_N^i\}$
05        $i := i+1$
06    ENDDO
07 STOP
```

Property 4.2. For any given context free grammar $G=(V_N,V_T,S,P)$ there exist a finite number, k, of production levels. (For the complete proof see [3].)

In building the type hierarchy we use an appropriate recursive specification for every hierarchic level. Every $V_N^i=\{v_{i1},v_{i2},\ldots,v_{ir}\}$ has the recursive specification:

$\psi_i:(2^{V_T^*})^r\to(2^{V_T^*})^r$,

$\psi_i(v_{i1},v_{i2},\ldots,v_{ir})=([v_{i1}],[v_{i2}],\ldots,[v_{ir}])$, where

$[v_{ij}]=\sum\{\alpha=s_0A_1s_1A_2\ldots s_nA_ns_{n+1} \mid p=A\to\alpha\in P \text{ and } d(p)=v_{ij}\}$.

Example 4.2. For the context free grammar of arithmetic expressions, form example 4.1, we have the following production hierarchy:

$P^1=\{F\to a\}$;

$P^2=\{F\to a,\ T\to F\}$;

$P^3=\{F\to a,\ T\to F,\ T\to T*F,\ E\to T\}$;

$P^4=P$.

and the corresponding recursive specifications are:

$\psi_1:(2^{V_T^*})\to(2^{V_T^*})$, $\psi_1(F)=(a)$;

$\psi_2:(2^{V_T^*})^2\to(2^{V_T^*})^2$, $\psi_2(F, T)=(a, F)$;

$\psi_3:(2^{V_T^*})^3\to(2^{V_T^*})^3$, $\psi_3(F, T, E)=(a, F\oplus T*F, T)$;

$\psi_4:(2^{V_T^*})^3\to(2^{V_T^*})^3$, $\psi_4(F, T, E)=(a\oplus (E), F\oplus T*F, T\oplus E+T)$.

The fix points of this specifications yield the appropriate types:

$\psi_1(\perp)=(a)$;

$\psi_1^2(\perp)=(a)$;

...

$\psi_1^n(\perp)=(a)$;

Thus $f_{\psi_1}=(a)$, i.e. $v_{11}=\{a\}$.

$\psi_2(\perp,\perp)=(a,\perp)$;

$\psi_2^2(\perp)=\psi_2(a,\perp)=(a,a)$;

...

$\psi_2^n(\perp)=(a,a)$;

...

Thus $f_{\psi_2}=(a,a)$, i.e. $v_{11}=\{a\}$, $v_{12}=\{a\}$.

$\psi_3(\perp,\perp,\perp)=(a,\perp,\perp)$;

$\psi_3^2(\perp,\perp,\perp)=\psi_3^1(a,\perp,\perp)=(a,a,\perp)$;

$\psi_3^3(\perp,\perp,\perp)=\psi_3^1(a,a,\perp)=(a,\{a,a*a\},a)$;

$\psi_3^4(\perp,\perp,\perp)=\psi_3^1(a,\{a,a*a\},a)=(a,\{a,a*a,a*a*a\},\{a,a*a\})$;

...

$\psi_3^n(\perp)=(a,\underbrace{\{a,a*a,a*a*a,\ldots,a*a*a*\ldots*a\}}_{n-1\,times},$

$\underbrace{\{a,a*a,a*a*a,\ldots,a*a*a*\ldots*a\}}_{n-2\,times})$

Therefore $f_{\psi_3} = (a, \{a^i \mid i \in \mathbb{N}\}, \{a^i \mid i \in \mathbb{N}\})$ and $v_{11} = \{a\}$, $v_{12} = \{a^i \mid i \in \mathbb{N}\}$, $v_{13} = \{a^i \mid i \in \mathbb{N}\}$, where $a^i = \underbrace{a * a * ... * a}_{i\ times}$.

$\psi_4(\perp, \perp, \perp) = (a, \perp, \perp)$;

$\psi_4^2(\perp, \perp, \perp) = \psi_4^1(a, \perp, \perp) = (a, a, \perp)$;

$\psi_4^3(\perp, \perp, \perp) = \psi_4^1(a, a, \perp) = (a, \{a, a * a\}, a)$;

$$\begin{aligned}\psi_4^4(\perp, \perp, \perp) = \psi_4^1(a, \{a, a * a\}, a) = (&\{a, (a)\},\\ &\{a, a * a, a * a * a\},\\ &\{a, a * a, a + a, a * a + a\})\end{aligned}$$

$$\begin{aligned}\psi_4^5(\perp, \perp, \perp) = \psi_4^1(&\{a, (a)\}, \{a, a*a, a*a*a\}, \{a, a*a,\\ &a+a, a*a+a\}) = (\{a, (a), (a*a), (a+a),\\ &(a*a+a)\}, \{a, (a), a*a, a*a*a,\\ &a*a*a*a, a*(a), a*a(a), a*a*a*(a)\},\\ &\{a, a*a, a*a*a, a+a, a*a+a, a+a+a,\\ &a*a+a+a, a+a*a, a*a+a*a,\\ &a+a+a*a, a*a+a+a*a, a+a*a*a,\\ &a*a+a*a*a, a+a+a*a*a,\\ &a*a+a+a*a*a\});\end{aligned}$$

...

Generally, f_{ψ_1} builds the set of base primary types of language $L(G)$ and represents the generating level or the lexicon of language $L(G)$, in other words, the lexical level of language $L(G)$.

5 Conclusions and future works

In general, concrete computational structures are composed of a family of object sets, a number of (partial determined operations) on these objects and a series of properties of these operations. Hence, a computational structure is an algebraic structure.

Since complex applications will require the possibility to reason about relational algebras build from other relational algebras via certain construction principles, it becomes necessarily to allow reasoning not only within single relation algebra but also about several structures and the connections between them.

Recursive specifications emphasize language construction from simple levels to complex ones. Even if we pointed out the internal hierarchy of a given grammar, this in turn can be extended unlimitedly, by adding, to new types and operations build upon the already defined levels.

The here presented ideas are part of our research on formal methods for programming language and on of the main research directions of our local Research Centre. Various connotations and practical aspects will be part of further scientific papers. As main purpose, we are currently intending to apply this method in systems for application generation based on multi layer specification. We will monitor with great interest the evolution of the system, over the time, to see if the unlimited extensibility is a reliable long run solution.

References:

[1] Ernest G. Manes, Michael A. Arbib, *Algebraic Approaches to program semantics* – Springer Verlag, New York, Berlin, Heidelberg, London, Paris, Tokyo, 1986.

[2] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, *Compilers*: *Principles, Techniques, and Tools*, Addison Wesley, 2001.

[3] Teodor Rus, *Mecanisme formale pentru specificarea limbajelor*, Ed. Academie Române, Bucureşti, 1983.

[4] Crăciunean V., *Translatoare şi compilatoare*, Ed. Alma Mater, Sibiu, 2002.

[5] Emil M. Popa, *Modele formale computationale, Editura* „Alma Mater", Sibiu, 2000

[6] Emil M. Popa, *Limbaje formale. Fundamentele limbajelor de programare*, Editura „Alma Mater", Sibiu, 2003

[7] Emil M. Popa, *Programare genetica si evolutiva*, Editura „Alma Mater", Sibiu, 2003

[8] Emil M. Popa, *Formal Syntax and Semantics of Programming Language*, Editura „Alma Mater", Sibiu, 2004

[9] Ralf Fabian, *Limbaje formale, Teorie, Exemple, Probleme*, Editura Univeristăţii „Lucian Blaga" Sibiu, 2006.

[10] Creangă, C. Reischer, D. Simovici, *Introducere algebrică în informatică – Limbaje formale*, Ed. Junimea, Iaşi, 1974.

[11] Creangă, C. Reischer, D. Simovici, *Introducere algebrică în informatică – Teorie automatelor*, Ed. Junimea, Iaşi, 1973.

[12] Toader Jucan, *Limbaje formale şi automate*, Ed. MatrixRom, 1999.

[13] Gabriel V. Orman, *Limbaje formale şi acceptori*, Ed. Albastră, Cluj-Napoca, 2002.

[14] Luca Dan Şerbănaţi, *Limbaje de programare şi compilatoare*, Ed. Academiei Române, Bucureşti, 1987.

[15] Jürgen Dassow, Gheorghe Păun, Regulated *Rewriting in Formal Language Theory*, Akademie Verlag Berlin, 1989

[16] Gh. Păun, *Gramatici contextuale*, Bucureşti, 1982

[17] Gh. Păun, *Mecanisme generative ale proceselor economice*, Ed Tehnică, Bucureşti, 1988.