ICOT Technical Memorandum: TM-0403

TM-0403

Inferring Parsers of Context-Free Languages from Examples of their Structural Descripitions

by
Y. Sakakibara (Fujitsu)

October, 1987

©1987, ICOT

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

Inferring Parsers of Context-Free Languages from Examples of their Structural Descriptions

Yasubumi SAKAKIBARA

IIAS-SIS, FUJITSU LIMITED

Abstract We consider a grammatical inference of context-free languages from their structural descriptions. In the context of inferring *parsers*, the *structure* of the grammar inferred is significant. The structure of context-free languages is described by the *shapes* of derivation trees. We will present an efficient inductive inference algorithm for parsers such that a grammar (or parser) inferred by the algorithm is not only a *correct* grammar which correctly generates the language but also assign a *correct* structure on the sentences of the language.

1. Introduction

In this paper, we will study inductive inference of parsers (or grammars) of context-free languages from examples of their structural descriptions. The problem of identifying a "correct" grammar for a language from finite examples of the language is known as grammatical inference. In the context of grammatical inference, a "correct" grammar only means a grammar which correctly generates the language. However when we consider the problem of identifying a parser for a language, the *structure* of the grammar identified is more significant. Consider the following example. The grammar G_1 below describes the set of all valid arithmetic expressions involving a variable "v" and the operations of multiplication "×" and addition "+".

(G_1)	(G_2)
$S \to v$	$S \to E$
$S \to Av$	$E \to F$
$A \to v+$	$E \to F+E$
$A \to v\times$	$F \to v$
$A \to v+A$	$F \to v\times F$
$A \to v\times A$	

However the structure assigned by grammar G_1 to sentences is semantically meaningless. The same language can be described by grammar G_2 below in a meaningful manner. Here the phrases are all significant in terms of the rules of arithmetic. Although G_1 and G_2 are weakly equivalent, this fact is not very relevant from a practical point of view since it would be unusual to consider a grammar such as G_1 which structures sentences in a nonsignificant manner.

Thus in the context of inferring a parser, since a grammar is intended for use in a practical situation entailing the translation or interpretation of sentences as in a compiler, it is necessary that a grammar inferred must not only generates the unknown language, but also assign a meaningful structure on the sentences of the language. To do so, it is necessary for us to assume that information on the structure of the language is available to the inference algorithm. In the case of context-free languages, the structure of the languages is usually described by the *shapes* of the derivation trees. Such structural descriptions are called *skeletons*. A skeleton is a kind of tree whose interior nodes have no label.

On the other hand, the set of derivation trees of a context-free language is rational, where a *rational* set of trees is a set of trees which can be recognized by some tree automaton. Furthermore, the set of skeletons of a context-free grammar is also rational. Based on this fact, the problem of inductive inference of parsers of context-free languages from the sentences and structures is reduced to the problem of inductive inference of tree automata. Then we can get an efficient inductive inference algorithm for parsers of context-free languages which is extended from one for automata [1].

2. Basic definitions of tree

Definition Let N denotes the set of positive integers. Dom is a *tree domain* iff it satisfies that (a) $Dom \subseteq N^*$ and Dom is finite, (b) Dom is prefix-closed, i.e. if $m, n \in N^*$ and $mn \in Dom$ then $m \in Dom$, (c) $ni \in Dom$ implies $nj \in Dom$ for $1 \leq j \leq i$, $j \in N$. A *direct successor* (*direct predecessor*) of a node x is a node y, where $y = xi$ ($yi = x$) for $i \in N$. A *terminal node* in Dom is one which has no direct successor. The *frontier* of Dom, denoted *frontier(Dom)*, is the set of all terminal nodes in Dom. The *interior* of Dom, denoted *interior(Dom)*, is Dom − frontier(Dom).

A *ranked alphabet* Γ is a finite set of symbols associated with a relation $r_\Gamma \subset \Gamma \times \{0,1,2,...,m\}$. For each $n \geq 0$, the subset $\{a \in \Gamma : (a, n) \in r_\Gamma\}$ is denoted by Γ_n. A *tree* over Γ is a mapping $t : \mathrm{Dom} \to \Gamma$, which labels the nodes of the tree domain Dom. We require the following condition : if $t(m) = f \in \Gamma_n$, then for $i \in N$, $mi \in \mathrm{Dom}(t)$ iff $1 \leq i \leq n$. Γ^T denote the set of all trees over Γ. Let $t = f(t_1,...,t_n)$ be a tree over Γ. The *replacement of terminal nodes* labeled $c \in \Gamma$ with a tree u is defined as $t(c \leftarrow u) = \{(m, x) : t(m) = x \text{ and } x \neq c\} \cup \{(ni, x) : t(n) = c, u(i) = x \text{ and } i \in \mathrm{Dom}(u)\}$. Let \$ be a new symbol of arity 0. $\Gamma_\$^T$ denotes the subset of $(\Gamma \cup \{\$\})^T$ which is the set of all trees $t \in (\Gamma \cup \{\$\})^T$ such that t exactly contains one \$-symbol. For trees $t \in \Gamma^T$ and $s \in \Gamma_\$^T$, we define an operation "#" to replace the node labeled \$ of s with t by $s\#t = s(\$ \leftarrow t)$.

Definition [4] A *skeletal alphabet* Sk is a ranked alphabet consisting of the singleton $\{\sigma\}$ of the special symbol σ associated with a relation $r_{Sk} \subseteq \{\sigma\} \times \{1,2,...,m\}$. A *skeleton* over an alphabet A is a mapping $s : \mathrm{Dom} \to A \cup Sk$ where σ is not in A, mapping frontier(Dom) to A and interior(Dom) to Sk. Let t be a tree over Γ. The *skeletal* (or *structural*) *description* of t, denoted *s(t)*, is a skeleton over Γ_0 such that

$$s(x) = t(x) \quad \text{for } x \in \mathrm{frontier(Dom)}$$
$$= \sigma \quad \text{for } x \in \mathrm{interior(Dom)}.$$

Let T be a set of trees. The *corresponding skeletal set*, denoted *S(T)*, is $S(T) = \{s(t) : t \text{ is in } T\}$.

3. Tree automaton and context-free grammar

Definition A *deterministic (frontier to root) tree automaton* over Γ is a 4-tuple $T_A = (Q, \Gamma, \delta, F)$, where (a) Q is a nonempty finite set of states,
(b) Γ is a nonempty finite ranked alphabet,
(c) $\delta = (\delta_0, \delta_1, ..., \delta_m)$ is a state transition function such that $\delta_k : \Gamma_k \times (Q \cup \Gamma_0)^k \to Q$ $(k = 1,2,...,m)$,
and $\delta_0(a) = a$ for $a \in \Gamma_0$,
(d) $F \subseteq Q$ is the set of final states.

If δ is a state transition function from $\Gamma_k \times (Q \cup \Gamma_0)^k$ to 2^Q, then T_A is *nondeterministic*.

δ can be extended to Γ^T by letting : $\delta(f(t_1,...,t_k)) = \delta_k(f, \delta(t_1),...,\delta(t_k))$. The tree t is *accepted* by T_A iff $\delta(t) \in F$. The set of trees accepted by T_A is the subset $L(T_A)$ of Γ^T defined as : $L(T_A) = \{t : \delta(t) \in F\}$.

In this definition, the labels on the frontier are taken as "initial" states.

Definition A *context-free grammar* is denoted $G = (N, \Sigma, P, S)$, where N and Σ are alphabets of *nonterminals* and *terminals* respectively such that $N \cap \Sigma = \emptyset$. P is a finite set of productions; each production is of the form $A \to \alpha$, where A is a nonterminal and α is a string of symbols from $(N \cup \Sigma)^*$. Finally, S is a special nonterminal called the *start symbol*. We define two relations $\Rightarrow$ and $\Rightarrow^*$ between strings in $(N \cup \Sigma)^*$. If $A \to \beta$ is a production of P and α and γ are any strings in $(N \cup \Sigma)^*$, then $\alpha A \gamma \Rightarrow \alpha\beta\gamma$. $\Rightarrow^*$ is the reflexive and transitive closure of $\Rightarrow$. The *language generated* by G, denoted *L(G)*, is $\{w : w \text{ is in } \Sigma^* \text{ and } S \Rightarrow^* w\}$.

$G = (N, \Sigma, P, S)$ is called a *wide-sense context-free grammar* if G is an usual context-free grammar but may have more than one starting symbol.

For A in $N \cup \Sigma$, the set $D_A(G)$ of trees over $N \cup \Sigma$, called a *derivation tree* of G from A, is recursively defined as :

$$D_A(G) = \{a\} \quad \text{for } A = a \in \Sigma,$$
$$= \{A(t_1,...,t_k) : A \Rightarrow B_1 \cdots B_k,\ t_i \in D_{B_i}(G)\ (1 \leq i \leq k)\} \quad \text{for } A \in N.$$

For the set $D_S(G)$ of derivation trees of G from the start symbol S, the S-subscript will be deleted.

Definition Two context-free grammars G_1 and G_2 are said to be *equivalent* if $L(G_1) = L(G_2)$. G_1 and G_2 are said to be *structurally equivalent* if $S(D(G_1)) = S(D(G_2))$.

For each wide-sense context-free grammar G, there is a context-free grammar G' with a unique start symbol such that G' is structurally equivalent to G.

Definition-A Let $G = (N, \Sigma, P, S)$ be a wide-sense context-free grammar. The *corresponding (nondeterministic) tree automaton* $T_A(G) = (Q, Sk \cup \Sigma, \delta, F)$ over $Sk \cup \Sigma$ is defined with (1) state set $Q = N$, (2) for each production of the form $A \to B_1 \cdots B_n$, the transition $\delta_n : Sk_n \times (Q \cup \Sigma)^n \to Q$ as $\delta_n(\sigma, B_1,...,B_n) = A$, and (3) final states $F = S$.

Proposition 1 Let $G = (N, \Sigma, P, S)$ be a wide-sense context-free grammar and $T_A(G)$ be the corresponding tree automaton in the sense of definition-A. Then $S(D(G)) = L(T_A(G))$.

Definition-B Let $T_A=(Q, Sk\cup\Sigma, \delta, F)$ be a tree automaton for a skeletal set over Σ. The *corresponding wide-sense context-free grammar* $G(T_A)=(N, \Sigma, P, S)$ is defined with (1) nonterminal alphabet $N=Q$, (2) for each σ of arity n and n-tuple $(x_1,...,x_n)$ of $Q\cup\Sigma$, the production $P_{x_1,...,x_n}\in P$ as $\delta_n(\sigma, x_1,...,x_n)\rightarrow x_1\cdots x_n$, and (3) start symbols $S=F$.

Proposition 2 Let $T_A=(Q, \Sigma\cup Sk, \delta, F)$ be a tree automaton and $G(T_A)$ be the corresponding context-free grammar in the sense of definition-B. Then $L(T_A)=S(D(G(T_A)))$.

4. State characterization matrix

Definition Let S be a finite set of trees over $\Sigma\cup Sk$ which includes the set $\{\sigma(\bar{a}) : \sigma\in Sk_i$ and $\bar{a}\in\Sigma^i$ for $i\geq 1\}$, $X(S)=\{\sigma(\bar{s}) : \sigma\in Sk_i, \bar{s}\in(S\cup\Sigma)^i$, and $\sigma(\bar{s})\notin S$ for $i\geq 1\}$, and E be a finite subset of $(\Sigma\cup Sk)_\$^T$. S is called *subtree-closed* if $s\in S$ implies all subtrees with depth at least 1 of s are elements of S. E is called *\$-prefix-closed with respect to* S if $e\in E$ except \$ implies there exists an e' in E such that $e=e'\#\sigma(s_1,...,s_{i-1},\$,s_i,...,s_{n-1})$ for some $s_1,...,s_{n-1}\in S\cup\Sigma$.

A *state characterization matrix* is a triple (S, E, M) where M is a matrix with labeled rows and columns such that (1) The rows are labeled with the elements of $S\cup X(S)$, (2) The columns are labeled with the elements of E, (3) Each entry of M is either 0 or 1, (4) If $s_i, s_j\in S\cup X(S)$ and $e_i, e_j\in E$ and $e_i\#s_i=e_j\#s_j$, then the (s_i, e_i) and (s_j, e_j) positions in M must have the same entry. The *data contained in* M is $D(M)=\{(e\#s, y) : s\in S\cup X(S), e\in E$, and the entry of M is $y\in\{0, 1\}\}$. For s in $(S\cup X(S))$, *row(s)* denotes the finite function f from E to {0, 1} defined by $f(e)=D(M)(e\#s)$.

A state characterization matrix is called *closed* if every row(x) of $x\in X(S)$ is identical to some row(s) of $s\in S$. A state characterization matrix is called *consistent* if whenever s_1 and s_2 are in S such that $row(s_1)$ is equal to $row(s_2)$, for all $u_1,...,u_{n-1}\in S\cup\Sigma$, $row(\sigma(u_1,...,u_{i-1},s_1,u_i,...,u_{n-1}))$ is equal to $row(\sigma(u_1,...,u_{i-1},s_2,u_i,...,u_{n-1}))$ for $1\leq i\leq n$.

The idea of the closed, consistent state characterization matrix is essentially the extensions of Angluin's one [1].

Definition Let (S, E, M) be a closed, consistent state characterization matrix such that E contains \$. The *constructed tree automaton* $T_A(M)$ over $\Sigma\cup Sk$ from (S, E, M) is defined with state set Q, final states F, and state transition function δ as follows.

$Q = \{row(s) : s\in S\}$,
$F = \{row(s) : s\in S \text{ and } D(M)(s)=1\}$,
$\delta_n(\sigma, row(s_1),...,row(s_n)) = row(\sigma(s_1,...,s_n))$,
$\delta_0(a) = a \quad \text{for } a\in\Sigma$,

where the function row is augmented to be $row(a)=a$ for $a\in\Sigma$.

Theorem 3 Supppose that (S, E, M) is a closed, consistent state characterization matrix such that S is subtree-closed and E is \$-prefix-closed with respect to S. Then the constructed tree automaton $T_A(M)$ agrees with the data in M. That is, for every tree s in $(S\cup X(S))$ and e in E, $\delta(e\#s)$ is in F iff $D(M)(e\#s)=1$.

5. Inductive inference algorithm for context-free grammar

We assume that a finite alphabet Σ which the unknown context-free grammar G is defined over and a skeletal alphabet Sk for the G are given.

Definition (construction of a context-free grammar) Let (S, E, M) be a closed, consistent state characterization matrix such that E contains \$. The *constructed wide-sense context-free grammar* $G(M)=(N, \Sigma, P, S)$ from (S, E, M) is defined with nonterminal alphabet N, start symbols $F\subseteq N$, and a finite set of productions P as follows.

$N = \{row(s) : s\in S\}$,
$S = \{row(s) : s\in S \text{ and } D(M)(s)=1\}$,
$P = \{row(\sigma(s_1,...,s_n))\rightarrow row(s_1)\cdots row(s_n)\}$,

where the function row is augmented to be $row(a)=a$ for $a\in\Sigma$.

(Algorithm 1)

Input : An oracle EX() for a sufficient set of examples of the skeletal descriptions of the unknown context-free grammar G, i.e. of S(D(G)),

An oracle MEMBER(s) on a skeleton s as input for a membership query to output 1 or 0 according to whether s is a skeletal description of a derivation tree of G from S, i.e. $s\in S(D(G))$,

Output : A sequence of conjectures of context-free grammar.

Procedure :

$S := \{\sigma(\bar{a}) : \sigma\in Sk_i \text{ and } \bar{a}\in\Sigma^i \text{ for } i\geq 1\}$;

E := {\$};

TA := ∅; CFG := ∅; Examples := ∅;

construct the initial matrix (S, E, M) using MEMBER;
TA := T_A(M);
CFG := G(M);
do forever
add an example EX() to Examples;
while there is a negative example $-s \in$ Examples which TA accepts s or there is a positive example $+s \in$ Examples which TA does not accept s;
add s and all its subtrees except constants to S;
extend (S, E, M) to $E\#(S \cup X(S))$ using MEMBER;
repeat
if (S, E, M) is not consistent
then find s_1 and s_2 in S, $u_1, \ldots, u_{n-1} \in S \cup \Sigma$, $e \in E$, and i ($1 \leqq i \leqq n$) such that row(s_1) is equal to row(s_2) and
$D(e\#\sigma(u_1, \ldots, u_{i-1}, s_1, u_i, \ldots, u_{n-1})) \neq$
$D(e\#\sigma(u_1, \ldots, u_{i-1}, s_2, u_i, \ldots, u_{n-1}))$;
add $e\#\sigma(u_1, \ldots, u_{i-1}, \$, u_i, \ldots, u_{n-1})$ to E;
extend (S, E, M) to $E\#(S \cup X(S))$ using MEMBER;
if (S, E, M) is not closed;
then find $\sigma(\hat{s}) \in X(S)$ for $\hat{s} \in (S \cup \Sigma)^n$ such that row($\sigma(\hat{s})$) is different from row(s) for all $s \in S$;
add $\sigma(\hat{s})$ to S;
extend (S, E, M) to $E\#(S \cup X(S))$ using MEMBER;
until (S, E, M) is closed and consistent;
TA := T_A(M);
CFG := G(M);
end;
output CFG;
end.

(Correctness) Let G be the unknown context-free grammar. *Given the oracles* EX *and* MEMBER *for* G, *the algorithm 1 identifies in the limit a minimal nonterminals wide-sense context-free grammar* CFG *such that* L(CFG) = L(G), CFG *is structurally equivalent to* G *and no two productions in* P *have the same right side.*

In [2], this type of identification is called *structural identification in the limit.*

(Time complexity) *The algorithm 1 infers a conjecture of context-free grammar and requests a new example in time polynomial in* l, m *and* n *after the last example has been added*, where l is the number of examples known at the time of the request, m is the maximum size of them and n is the number of states in the minimum tree automaton for S(D(G)).

Definition (construction of a parser) Let (S, E, M) be a closed, consistent state characterization matrix such that E contains \$. The *constructed parsing Prolog program* PARSER(M) using difference-lists from (S, E, M) is defined with the predicate set Predicate, the finite set of function symbols Function, the calling predicate st(T,X,X'), and the finite set of clauses PARSER(M) as follows

Predicate = $\{phr_{row(s)}(T,X,X') : s \in S\} \cup \{ter_a(a,[a|X],X) : a \in \Sigma\}$,
Function = $\{f_{row(s)} : s \in S\}$,
PARSER(M) =
$\{st(T,X_0,X_1) :- phr_{row(s)}(T,X_0,X_1).\ : s \in S \text{ and } D(M)(s) = 1\}$
$\cup \{phr_{row(\sigma(s_1,\ldots,s_n))}(f_{row(\sigma(s_1,\ldots,s_n))}(T_1,\ldots,T_n),X_0,X_n)$
$:- R_1(T_1,X_0,X_1),\ldots,R_n(T_n,X_{n-1},X_n).$
$: R_i = phr_{row(s_i)}$ if $s_i \in S$ and $R_i = ter_a$ if $s_i = a \in \Sigma$ $(1 \leqq i \leqq n)\}$
$\cup \{ter_a(a,[a|X],X).\ : a \in \Sigma\}$.

6. Discussions

As Crespi-Reghizzi et al. [3] suggest, grammatical inference may be useful in specifying programming languages. Then an application of our algorithm is designing programming languages or synthesis of compiler, because the structure or syntax of programming languages is usually defined by means of a context-free grammar. As in [3], the definition of structure and the definition of meaning should be interconnected since structural orderings are an aid to interpreting a sentence. Thus in inferring a programming language, a grammar inferred for the language should be constructed such that it not only generates correctly sentences but also assigns to each sentence a structure required by the designer. Then our approach will provide an effective method for the process of programming language design.

This is part of the work in the major R&D of FGCP, conducted under program set up by MITI.

References

[1] Angluin,D., Learning regular sets from queries and counter-examples, *Yale DCS TR-464*, 1986.
[2] Crespi-Reghizzi,S., An effective model for grammar inference, in *Information Processing 71*, Elsevier North-Holland (1972), 524-529.
[3] Crespi-Reghizzi et al., The use of grammatical inference for designing programming languages, *Comm. ACM 16* (1973), 83-90.
[4] Levy,L.S., Joshi,A.K., Skeletal structural descriptions, *Information and Control 39* (1978), 192-211.