A Lattice Framework for Analyzing Context-Free Languages with Applications in Parser Simplification and Data-Flow Analysis

WUU YANG
Department of Computer and Information Science
National Chiao Tung University
Hsinchu, Taiwan 300, R.O.C.

We propose a lattice framework for analyzing context-free grammars and context-free languages. This framework is motivated by a technique for simplifying parsers with information derived from the associated scanners. We define the lattice framework and demonstrate it using additional applications, including data-flow analysis. Soundness and other properties of the lattice framework are also discussed.

Keywords: compiler, context-free grammar, finite-state machine, lattice, Mealy machine, parser, regular expression, scanner

1. INTRODUCTION

In this paper, we first propose a technique for simplifying parsers using information derived from scanners (Sections 2 through 4). We then describe a lattice framework underlying the technique (Sections 5 and 6). The lattice framework can be used to analyze many properties of context-free grammars and context-free languages. We demonstrate the lattice framework using several additional examples, including a data-flow analysis problem. Soundness and other properties of the lattice framework are also proved.

The front-end of a traditional compiler consists of a scanner and a parser. The scanner and the parser form a producer/consumer relationship: the scanner produces a stream of tokens from an input character stream while the parser consumes the tokens. Since the parser has no control over what the scanner will produce, the parser usually assumes that its input is an arbitrary sequence of tokens. However, in a previous paper [1], we proposed that the scanner should be described by a Mealy machine [2] (rather than a Moore machine), in which tokens produced by a scanner are associated with state transitions, rather than states. A Mealy-machine description implies that what the scanner produces is *not* an arbitrary sequence of tokens; rather, the sequence of tokens produced by the scanner is defined by a regular expression. When the input to the parser is defined by a regular expression, it is possible to simplify the parser (by means of removing states, transitions, and production rules) using information derived from the regular expression.

Received April 29, 1997; accepted September 15, 1997.
Communicated by Jang-Ping Sheu.
This work was supported in part by the National Science Council, Taiwan, R.O.C., under grant NSC 85-2213-E-009-051.

We first describe a new, algebraic technique to extend the transition function of the *characteristic output machine* of the scanner (to be defined in Section 2). The extension is encoded in a $\oplus$ operator. The $\oplus$ operator is used to set up equations on sets of states. These equations are solved by an iteration algorithm. Based on the solution of the equations, we can identify useless transitions and unreachable states of the finite-state machine underlying an LR parser. It is also possible to eliminate production rules from the grammar in certain cases. In this paper, we demonstrate the technique on an LR(0) machine. However, it is straightforward to apply the technique to other LR(k) machines.

The simplification technique is actually a special case of a general lattice framework underlying context-free languages. A context-free language is defined by a context-free grammar, which contains terminals, nonterminals, and production rules. We may adopt a different view of a context-free grammar: The terminals and nonterminals denote sets of strings; the production rules are equations specifying relationships among these sets. The equations can be solved for the sets denoted by the nonterminals. The context-free language is the set denoted by the start symbol of the grammar. For instance, let $A \rightarrow UVW \mid XYZ$ be the set of all the A-productions (where the A-productions are the productions whose left-hand-sides are the nonterminal A) in a grammar. These A-productions can be viewed as the equation $A = (UVW) \cup (XYZ)$. From this viewpoint, a context-free language is defined, not in isolation, but by means of relationships among closely related sets (i.e., those sets denoted by terminals and nonterminals).

We might be interested in some properties of a context-free language. For instance, we might want to know whether all the sentences of a context-free language have even lengths. Since a context-free language is a set of sentences and a sentence is the concatenation of symbols, a property of a context-free language is the aggregate of the properties of individual sentences, and the properties of individual sentences are the aggregates of the properties of individual symbols. Note that the aggregation corresponds closely to the production rules. Hence, the production rules may be viewed as equations defining relationships among properties of terminals and nonterminals. For instance, the above set of A-productions can be viewed as the equation $f_A=(f_U \otimes f_V \otimes f_W) \blacklozenge (f_X \otimes f_Y \otimes f_Z)$, where f_A denotes a property of (the set represented by) the nonterminal A, and $\otimes$ and $\blacklozenge$ are the aggregation operations corresponding to the concatenation and the | operators in the production rules, respectively. Thus, a property of a context-free language is defined, not in isolation, but by means of relationships among properties of closely related languages. (Note that each terminal or nonterminal denotes a context-free language.)

We can solve the equations if the domain-of-interest is a finite lattice in which the aggregation operator $\blacklozenge$ is the $\sqcup$ (or $\sqcap$) operation. Note that $\sqcup$ (or $\sqcap$, respectively) acts "accumulatively"; that is, if $x \leq y$, then $x \sqcup y = y$ (or $x \sqcap y = x$, respectively). If $\otimes$ satisfies certain desirable properties, we can solve the equations by gradually accumulating the results from the least (or greatest, respectively) elements of the lattice. Since the lattice is finite, the accumulation process is guaranteed to terminate. It is found in Sections 2 through 4 that the simplification technique makes use of two such lattice frameworks. We also find that many other interesting problems can be solved using a lattice framework. We characterize the lattice framework and prove several fundamental properties of the framework.

The remainder of this paper is organized as follows. Section 2 reviews the Mealy-machine description of a scanner and defines the *characteristic output machine* of the scanner. Section 3 presents an algebraic technique to extend the characteristic output machine. We will use the results to simplify parsers, which are described in Section 4. In Section 5, we propose a lattice framework for analyzing context-free grammars and context-free languages. Several additional examples are used to demonstrate the framework. In Section 6, we show that when a lattice framework is used to analyze context-free *languages*, the results of analysis are independent of the particular context-free *grammars* used to describe the context-free *languages*. Properties of the language-analysis schemes are also discussed. In Section 7, we briefly discuss the language-analysis schemes based on an infinite lattice. The last section concludes this paper and discusses related work.

2. COMPUTING THE CHARACTERISTIC OUTPUT MACHINE OF A SCANNER

A scanner is formally specified by a set of regular expressions defining various kinds of tokens. Ambiguities encountered during scanning are resolved by the *longest match rule*. For instance, the string “123456” is considered an integer of six digits rather than six integers of one digit each. The longest match rule forces the scanner to look ahead a few characters in deciding the end of a token. If the scanner looks ahead only a finite number of characters, its look-ahead behavior can be integrated into the finite-state machine by associating a scanner’s output with state transitions. That is, a Mealy machine can accommodate the finite look-ahead behavior quite well and, hence, is a better model of scanners [1]. The process of creating the Mealy machine of a scanner is described in detail in [1].

The output of a scanner, which is a Mealy machine, can also be described by a finite-state machine. The *characteristic output machine* (COM) of a Mealy machine is obtained by ignoring the input components on transitions in the Mealy machine. If there is no output token associated with a transition, that transition will be an ε-transition. The resulting COM is usually non-deterministic. A deterministic COM can be obtained from the non-deterministic COM by means of the subset construction technique[3].

Example: Consider the scanner specification in Fig. 1(a), which defines four kinds of tokens. The *$* is the end-of-file character, which is also the end-of-file token. Fig. 1(b) is the finite-state machine of the scanner. The double-circles denote accepting states. State 1 is the initial state. Fig. 1(c) is the Mealy machine of the scanner, in which output is associated with transitions rather than states. The bold-face edges are the augmented edges. The notation c/μ on an edge means that for input character *c*, the Mealy machine will produce an output token μ. Note that Fig. 1(c) is actually a generalized Mealy machine in the sense that a sequence of zero or more tokens may be associated with a transition. When the input components on the edges of Fig. 1(c) are ignored, a nondeterministic COM is obtained. This nondeterministic COM can be transformed into a deterministic one using standard techniques. The resulting characteristic output machine is shown in Fig. 1(d). Note that the characteristic output machine does not accept a trivial regular expression; in particular, it cannot accept two consecutive μ’s. Hence, the scanner is *singular* (defined below).

Definition: A scanner is *non-singular* if it can produce an arbitrary stream of tokens ending with a unique *end-of-file* token. A scanner is *singular* if it is not non-singular.

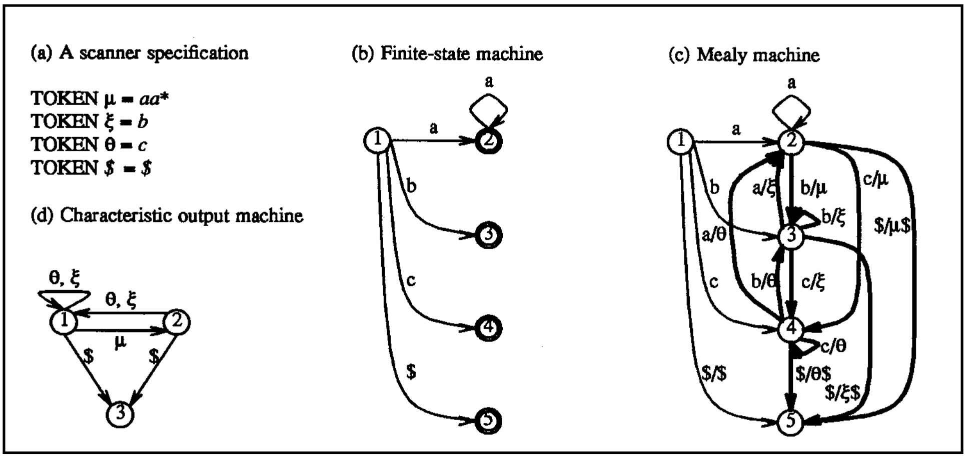


Fig. 1. A scanner specification.

The stream of tokens produced by a *non-singular* scanner is the trivial regular expression (μ | ξ | θ | ...)*$, where μ, ξ, θ,.... are the tokens, and *$* is the end-of-file mark. On the other hand, the stream of tokens produced by a *singular* scanner is described by a non-trivial regular expression. Intuitively, a singular scanner provides additional information about its output. Most scanners are singular. (Consider that most scanners cannot produce two consecutive integers without an intervening space.) Though practical programming languages avoid singularity by allowing blanks and comments to be inserted between tokens in the source programs, some programming languages do not allow such arbitrary white spaces. For instance, some VLSI description languages are translated into intermediate languages for further processing. Since the intermediate languages are not intended for humans to read or write, they can forbid blanks and comments. The technique proposed in the paper will be useful in these intermediate languages.

A parser usually *assumes* that it takes input from a non-singular scanner. If it is known that the parser takes input from a singular scanner, additional information can be obtained from the characteristic output machine of the scanner. The information can be used to simplify parsers. We will discuss this technique in the following two sections.

3. EXTENDING THE CHARACTERISTIC OUTPUT MACHINES

The interface between a scanner and a parser is the stream of tokens. The stream of tokens must be accepted by the COM of the scanner and must orm to the context-free grammar of the parser. Therefore, we seek a relationship between a finite-state machine and a context-free grammar.

Let *M* be a finite-state machine and *G* be a context-free grammar. We define an operator ⊕ that takes two arguments: a set of states (of *M*) and a string of terminals and nonterminals (of *G*). The result of ⊕ is a set of states (of *M*). The notation $Q \oplus \alpha = Q'$ means that, starting from a state of *Q*, *M* will reach a state of Q' on an input string that is derivable from the string α (according to the production rules of G). ⊕ may be viewed as an extension of the transition function of the COM of a scanner to the nonterminals of the context-free grammar of a parser. The operator ⊕ is defined by the following four axioms:

(1) $\{q\} \oplus a = \{q' \mid M$ moves from state q to state q' on input a, where a is a terminal of $G\}$.
(2) $\{q\} \oplus A = \{q' \mid M$ moves from state q to state q' on a string of terminals α, where A is a nonterminal of G and $A \rightarrow {}^*\alpha\}$.
(3) $Q \oplus \alpha\beta = (Q \oplus \alpha) \oplus \beta$, where α and β are strings of terminals and nonterminals.
(4) $(Q_1 \cup Q_2) \oplus \alpha = (Q_1 \oplus \alpha) \cup (Q_2 \oplus \alpha)$, where α is a string of terminals and nonterminals.

By convention, $\varnothing \oplus \alpha = \varnothing$.

To find $\{q\} \oplus a$ for a state q of M and a terminal a of G, we can simply examine the transition table of M. To find $\{q\} \oplus A$, where A is a nonterminal of G, we can establish a set of equations and solve the equations iteratively. Let $A \rightarrow \alpha \mid \beta \mid$....be the set of all the A-productions in G. Then, $\{q\} \oplus A = (\{q\} \oplus \alpha) \cup (\{q\} \oplus \beta) \cup$.....

There is one such equation for each state q of M and each nonterminal A of G. The above set of equations can be solved using an iteration algorithm. Initially, assume $\{q\} \oplus A = \varnothing$ for each q and each A. Then, we can repeatedly evaluate the set of equations until a stable solution is reached. The iteration algorithm is shown in Fig. 2, where an expression, such as $\{q\} \oplus A$, is treated as a variable.

```
Algorithm: Iteration
Given a set of equations $x_i = f_i$ (...), for $i$ = 1, 2, ..., $k$
**for** $i$: = 1 **to** $k$ **do** $x_i$: = $\varnothing$
**repeat**
    **for** $i$: = 1 **to** $k$ **do**
        $x_i$: = $f_i$ (...)
    $stable$: = $true$
    **for** $i$: = 1 **to** $k$ **do**
        **if** $x_i$= $x_i$ **then begin**
                    $x_i$: = $\overline{x_i}$
                    $stable$: = $false$
                    **end**
**until** $stable$
```

Fig. 2. The iteration algorithm.

Example: Fig. 3(b) is the ⊕ operator applied to the characteristic output machine shown in Fig. 1(d) and the context-free grammar in Fig. 3(a). For the sake of brevity, we have omitted the set symbols {……}. From the definition of the ⊕ operator, we obtain the following six equations:

$$
\begin{aligned}
\{1\}\oplus T &= (\{1\}\oplus \mu T\mu)\cup(\{1\}\oplus \xi T\xi)\cup(\{1\}\oplus\theta)\\
&= (\{1\}\oplus\mu\oplus T\oplus\mu)\cup(\{1\}\oplus\xi\oplus T\oplus\xi)\cup(\{1\}\oplus\theta)\\
&= (\{2\}\oplus T\oplus\mu)\cup(\{1\}\oplus T\oplus\xi)\cup\{1\}\\
\{2\}\oplus T &= (\{2\}\oplus \mu T\mu)\cup(\{2\}\oplus \xi T\xi)\cup(\{2\}\oplus\theta)\\
&= (\{2\}\oplus\mu\oplus T\oplus\mu)\cup(\{2\}\oplus\xi\oplus T\oplus\xi)\cup(\{2\}\oplus\theta)\\
&= (\emptyset\oplus T\oplus\mu)\cup(\{1\}\oplus T\oplus\xi)\cup\{1\}\\
\{3\}\oplus T &= (\{3\}\oplus \mu T\mu)\cup(\{3\}\oplus \xi T\xi)\cup(\{3\}\oplus\theta)\\
&= (\{3\}\oplus\mu\oplus T\oplus\mu)\cup(\{3\}\oplus\xi\oplus T\oplus\xi)\cup(\{3\}\oplus\theta)\\
&= (\emptyset\oplus T\oplus\mu)\cup(\emptyset\oplus T\oplus\xi)\cup\emptyset\\
\{1\}\oplus S &= \{1\}\oplus T\$\\
&= \{1\}\oplus T\oplus\$\\
\{2\}\oplus S &= \{2\}\oplus T\$\\
&= \{2\}\oplus T\oplus\$\\
\{3\}\oplus S &= \{3\}\oplus T\$\\
&= \{3\}\oplus T\oplus\$.
\end{aligned}
$$

(a) A context-free grammar G

P1: $S \to T\,\$$
P2: $T \to \mu T\,\mu$
P3: $T \to \xi T\,\xi$
P4: $T \to \theta$

(b) the ⊕ operator

⊕	μ	ξ	θ	$	*T*	*S*
1	2	1	1	3	1,2	3
2	∅	1	1	3	1	3
3	∅	∅	∅	∅	∅	∅

Fig. 3. A context-free grammar and the ⊕ operator.

Let x_1,x_2,x_3,x_4,x_5, and x_6 denote the six terms: $\{1\} \oplus T$, $\{2\} \oplus T$, $\{3\} \oplus T$, $\{1\} \oplus S$, $\{2\} \oplus S$, and $\{3\} \oplus S$, respectively. After some simplification, we get the following six equations:

$$
\begin{aligned}
x_1 &= (x_2\oplus\mu)\cup(x_1\oplus\xi)\cup\{1\}\\
x_2 &= (x_1\oplus\xi)\cup\{1\}\\
x_3 &= \emptyset\\
x_4 &= x_1\oplus\$\\
x_5 &= x_2\oplus\$\\
x_6 &= x_3\oplus\$.
\end{aligned}
$$

Initially, assume that $x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = \emptyset$. We can repeatedly evaluate the six equations. After three iterations, we reach a stable solution: $\{1\}\oplus S = \{3\}$, $\{2\} \oplus S = \{3\}$, $\{3\} \oplus S = \emptyset$, $\{1\} \oplus T = \{1,2\}$, $\{2\} \oplus T = \{1\}$, and $\{3\} \oplus T = \emptyset$.

The algorithm in Fig. 2 always halts due to its accumulative nature. That the solution is correct can be proved by inductive reasoning as follows: the addition of a state q' into the solution of $\{q\} \oplus A$ can be traced backward, eventually to a transition $q_1 \rightarrow^a q_2$, where a is a token, in the finite-state machine. Thus, we can construct a string of terminals that is derivable from A, and that moves the finite-state machine from state q to state q'.

An application of the $\oplus$ operator is to decide whether a regular language and a context-free language intersect. A classical method for this problem is to integrate the finite-state machine of the regular language and the pushdown automaton of the context-free language into a new pushdown automaton. A new context-free grammar can then be derived from the integrated pushdown automaton. By contrast, with the $\oplus$ operator, the regular language and the context-free language intersect if and only if the set $\{q_0\} \oplus S$, where q_0 is the initial state of the finite-state machine of the regular language, and S is the start symbol of the context-free grammar, contains an accepting state of the finite-state machine. Though the $\oplus$ operator does not compute the exact intersection, it provides additional information relating the states of a finite-state machine to the nonterminals of a context-free grammar. This information is useful in simplifying the LR parser of the context-free grammar as well as the grammar itself.

4. SIMPLIFYING THE GRAMMARS AND THE PARSERS

The information provided by the $\oplus$ operator may be useful in simplifying parsers as well as grammars. Given a context-free grammar G, let $CFSM(G)$ be the characteristic finite-state machine of G [4]. Given a state s of $CFSM(G)$, let $L(s)$ denote the language accepted by state s, that is, the set of terminal strings α such that the parser of G halts at state s on input α. Formally, $L(s)$ is defined by the following context-free grammar $G_s : S \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots.$, where S is a new nonterminal not occurring in G, and $\alpha_1, \alpha_2, \ldots.$are the labels on all the distinct paths from the initial state to state s in $CFSM(G)$. G_s also contains all the A-productions of G if the nonterminal A occurs in any of $\alpha_1, \alpha_2, \ldots..$ Similarly, G_s contains all the B-productions of G if the nonterminal B occurs in any of the A-productions, etc. Note that $L(s)$ is also a context-free language.

Given the regular expression defining a scanner, let M be the characteristic output machine of the scanner. Let Q be a set of states of M and s be a state of $CFSM(G)$. Define $\tau(Q,s) = \cup \{\{q\} \oplus \alpha \mid q \in Q, \alpha \in L(s)\}$. Intuitively, $\tau(Q, s)$ denotes the set of possible final states of M when M, starting from a state in Q, scans a string of $L(s)$. Set union $\cup$ may be distributed over the function $\tau : \tau(Q_1 \cup Q_2, s) = \tau(Q_1, s) \cup \tau(Q_2, s)$. Note that $\tau(\emptyset, s) = \emptyset$. We can compute the τ function as follows: Let $t_1 \rightarrow^{x1} s, t_2 \rightarrow^{x2} s, \ldots., t_k \rightarrow^{xk} s$ be all the incoming edges of state s in $CFSM(G)$, where $x1, x2, \ldots, xk$ are either terminals or nonterminals. Note that $L(s) = L(t_1) \bullet \{x_1\} \cup L(t_2) \bullet \{x_2\} \cup \ldots \cup L(t_k) \bullet \{x_k\}$,where $\bullet$ is defined as follows: $A \bullet B = \{\alpha\beta \mid \alpha \in A, \beta \in B\}$ and $\alpha\beta$ is the concatenation of the two strings α and β. Then, $\tau(Q, s) = (\tau(Q, t_1) \oplus x1) \cup (\tau(Q, t_2) \oplus x2) \ldots. \cup (\tau(Q, t_k) \oplus x_k)$.

In order to compute the values of $\tau(\{q\}, s)$ for each state q of M and each state s of $CFSM(G)$, we can set up an equation for $\tau(\{q\}, s)$. Then, an iteration algorithm similar to the one in the previous section can be applied. Initially, let $\tau(\{q\}, s_0) = \{q\}$, where s_0 is the initial state of $CFSM(G)$. (The reason is that the empty string $\varepsilon \in L(s_0)$ and $\{q\} \subseteq \{q\} \oplus \varepsilon$.) Let $\tau(\{q\}, s) = \emptyset$ for all other states s. We can then evaluate the set of equations defining τ repeatedly until a stable solution is obtained. Note that in computing $\tau(\{q\},s)$, we can make use of the $\oplus$ operator obtained in the previous section.

The τ function can be used to remove redundant states and transitions from *CFSM* (*G*). Let q_0 be the initial state of *M*. First, let *s* be a state of *CFSM*(*G*). If $\tau(\{q_0\}, s) = \emptyset$, the state *s* is not reachable during parsing and, hence, can be removed. Secondly, let $t \to^{x} s$ be an edge in *CFSM*(*G*). If $\tau(\{q_0\}, t) \oplus x = \emptyset$, the transition $t \to^{x} s$ can never be followed during parsing and hence can be removed as well. The removal of redundant states and transitions may render certain productions of the context-free grammar useless. These useless productions can be deleted. The removal of states, transitions, or productions may also reduce the size of the parse table. The technique presented here can be applied not only to *CFSM*(*G*), but also to LR(*k*) machines for any *k* and to LL parsers.

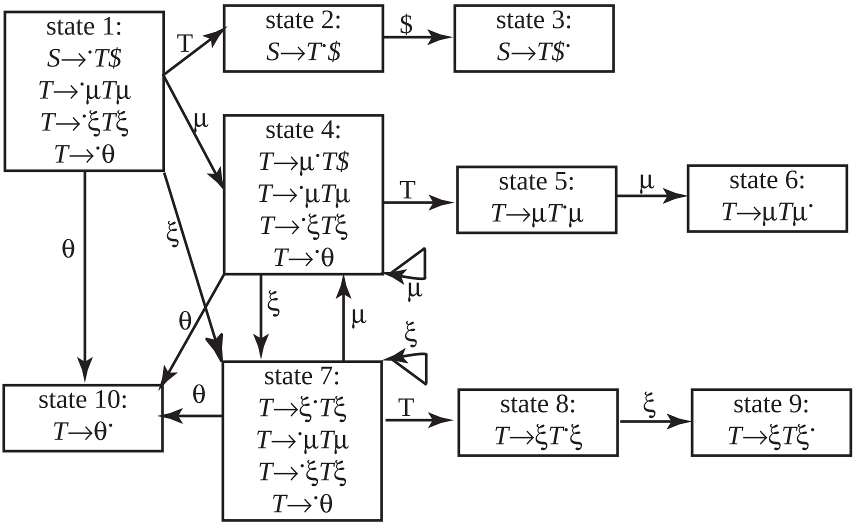
Note that the removal of states and transitions from *CFSM*(*G*) depends on the constraints placed on the input to the parser, which is the output from the scanner. The output from the scanner is characterized by the characteristic output machine of the scanner. On the other hand, if the input to the parser is an arbitrary stream of tokens, no states and transitions can be removed.

Example: Fig. 4(a) is the characteristic finite-state machine of the grammar in Fig. 3(a). Figs. 4(b) and (c) are the action and goto tables. Fig. 4(d) is the τ function on the COM in Fig. 1(d) and the characteristic finite-state machine in Fig. 4(a). We have omitted the set symbol {…} in Fig. 4 for the sake of brevity. The equations defining the τ function on state 1 are as follows:

$$\begin{aligned}
&\tau(\{1\},\ 1) = \{1\}\\
&\tau(\{1\},\ 2) = \tau(\{1\},\ 1) \oplus T\\
&\tau(\{1\},\ 3) = \tau(\{1\},\ 2) \oplus \$\\
&\tau(\{1\},\ 4) = (\tau(\{1\},\ 1) \oplus \mu) \cup (\tau(\{1\},\ 4) \oplus \mu) \cup (\tau(\{1\},\ 7) \oplus \mu)\\
&\tau(\{1\},\ 5) = \tau(\{1\},\ 4) \oplus T\\
&\tau(\{1\},\ 6) = \tau(\{1\},\ 5) \oplus \mu\\
&\tau(\{1\},\ 7) = (\tau(\{1\},\ 1) \oplus \xi) \cup (\tau(\{1\},\ 4) \oplus \xi) \cup (\tau(\{1\},\ 7) \oplus \xi)\\
&\tau(\{1\},\ 8) = \tau(\{1\},\ 7) \oplus T\\
&\tau(\{1\},\ 9) = \tau(\{1\},\ 8) \oplus \xi\\
&\tau(\{1\},\ 10) = (\tau(\{1\},\ 1) \oplus \theta) \cup (\tau(\{1\},\ 4) \oplus \theta) \cup (\tau(\{1\},\ 7) \oplus \theta).
\end{aligned}$$

The equations are solved using an iteration algorithm similar to that in Fig. 2. The τ function for other states of *M* is computed similarly. Consider the edge *e* from state 4 to itself labeled μ in Fig. 4(a). Intuitively, the edge *e* is traversed only when the scanner produces two consecutive μ's. Since the scanner in Fig. 1 can never produce two consecutive μ's, that edge can be eliminated from the parser. Formally, the edge *e* may be removed because $\tau(\{1\}, 4) \oplus \mu = \emptyset$.

Example: Suppose the definition of the token ξ is changed to *bb**. In this case, the scanner would not be able to produce two consecutive ξ's. Hence, the edge from state 7 to itself in *CFSM*(*G*) will not be traversed during parsing.



(a) CFSM(G)

(b) the Action table of *G*

state	1	2	3	4	5	6	7	8	9	10
action	S	S	R1	S	S	R2	S	S	R3	R4

(c) the Goto table of *G*

	1	2	3	4	5	6	7	8	9	10
μ	4			4	6		4			
ξ	7			7			7	9		
θ	10			10			10			
$		3								
T	2			5			8			

(d) The τ function

τ	1	2	3	4	5	6	7	8	9	10
1	1	1,2	3	2	1	2	1	1,2	1	1
2	2	1	3	2	1	2	1	1,2	1	1
3	3	∅	∅	∅	∅	∅	∅	∅	∅	∅

Fig. 4. The τ function.

5. A LATTICE FRAMEWORK FOR ANALYZING CONTEXT-FREE LANGUAGES

Let $G = (N, T, P, S)$ be a context-free grammar. Let L_G be the language defined by G. We may wish to compute certain collective properties of L_G based on a mapping $M : T^* \rightarrow F'$, where F' is the domain-of-interest (M maps individual strings of terminals into the domain-of-interest). For instance, M may be taken as the *length* function (i.e., by computing the length of a string), and we want to know whether all sentences of L_G are of even length. Since L_G is, in general, infinite, it is not possible to compute the collective properties by enumerating the sentences of L_G one by one. The lattice framework proposed in this section provides us with a feasible way to compute the collective properties.

Assume that the domain-of-interest F' can be embedded into (or is closely related to) a finite (see problem 6 below) lattice $(F, \leq)$. Let $\sqcup$ and $\sqcap$ be the join and meet operations, respectively. The least element and the greatest element of F are denoted by $f_{\min}$ and $f_{\max}$, respectively. M is modified and extended accordingly so that M maps strings of terminals and nonterminals to s of F, and M satisfies the following property: $M(\alpha\beta) = M(\alpha) \otimes M(\beta)$, where $\otimes$ is an associative and monotone operation on F with an identity element e (that is, if $a \leq b$, then $a \otimes c \leq b \otimes c$, $c \otimes a \leq c \otimes b$, and $c \otimes e = e \otimes c = c$, for all $c \in F$). (Under the above conditions, $M(\varepsilon) = e$, here ε isthe empty string.) As a notational convenience, $M(\alpha)$ will be written as f_α. f_α represents the collective properties of $\{\beta \mid \beta \in T^*, \alpha \rightarrow {}^*\beta\}$.

For each $a \in T$, we may compute its image f_a in F directly. In order to compute f_α, where $\alpha \in (N \cup T)^*$, we need to find f_A for each $A \in N$. Since A may be viewed as the set of terminal strings L_A that are derivable from the nonterminal A through the production rules of G, the production rules of G define the relationships among these context-free languages L_A. These relationships (or production rules) are translated into equations in F as follows: For each $A \in N$, the set of all the A-productions $A \rightarrow \alpha \mid \beta \mid \ldots$ is translated into the equation $f_A = h_\alpha \blacklozenge h_\beta \blacklozenge \ldots.$, where $h_\alpha = H(\alpha, f_\alpha)$ for a function H that is monotone in its second argument. H is monotone in the sense that if $f \leq f'$, then $H(\alpha, f) \leq H(\alpha, f')$ for all strings $\alpha \in (N \cup T)^*$. The $\blacklozenge$ operator is called the luence operator, which may be either $\sqcup$ or $\sqcap$. We also require that $\blacklozenge$ be distributive over $\otimes$, that is, $f \otimes (h_1 \blacklozenge h_2) = (f \otimes h_1) \blacklozenge (f \otimes h_2)$ and $(h_1 \blacklozenge h_2) \otimes f = (h_1 \otimes f) \blacklozenge (h_2 \otimes f)$. Let $f_\blacklozenge$ be the identity element of $\blacklozenge$, namely, $f_\blacklozenge = f_{\min}$ if $\sqcup$ is the confluence operator; $f_\blacklozenge = f_{\max}$ if $\sqcap$ is the confluence operator. It is also required that $f \blacklozenge f_\blacklozenge = f_\blacklozenge \blacklozenge f = f$ for all $f \in F$. Then we can solve this set of equations using the iteration algorithm in Fig. 5, where, for each $A \in N$, the initial value of f_A is $f_\blacklozenge$.

```
Algorithm: Iteration
for each nonterminal $A$ do $f_A := f_\blacklozenge$
repeat
  for each nonterminal $A$ do
    $\overline{f_A} := h_\alpha \blacklozenge h_\beta \blacklozenge \ldots$, where $f_A = h_\alpha \blacklozenge h_\beta \blacklozenge \ldots$ is the equation defining $f_A$
stable: = true
for each nonterminal $A$ do
    if $f_A \neq \overline{f_A}$ then begin
            $f_A := \overline{f_A}$
            stable: = false
            end
until stable
```

Fig. 5. The iteration algorithm.

Let $B = \{f_a \mid a \in T\}$. B, the collection of the images of terminals in the lattice, represents the basic building blocks of the framework. The five components F, $\otimes$, B, H, and ♦ consist of an analysis scheme of a context-free grammar. The detailed definition is delayed until the next section. In this section, we will examine several examples to show the many applications of the lattice frameworks.

That the iteration algorithm always halts with a stable solution can be demonstrated as follows. For each $A \in N$, let $f_{A,j}$ be the value for A at the end of the j^{th} iteration. Let $f_{A,j+1}$ be the value for A at the end of the $j+1^{st}$ iteration. Since H, $\otimes$, and $\sqcup$ (or $\sqcap$) are monotone, and the iteration algorithm starts from $f_{\min}$ (or $f_{\max}$, respectively), $f_{A,j} \le f_{A,j+1}$ (or $f_{A,j+1} \le f_{A,j}$, respectively) for every $A \in N$. Since F is a finite lattice, the iteration algorithm always halts with a stable solution. (We will show that the stable solution is correct in the next section.)

Let S be the start symbol of the grammar G. In general, f_s is the intended result. Sometimes, we might be interested in f_α for some string α. It is straightforward to compute f_α once f_A, for each $A \in N$, is known. We can gain insight into properties of the string α through its image, f_α, in F.

The essence of the framework is to compute some values, such as f_s or f_α, that cannot be computed directly. Note that there are subtle relationships (induced by the production rules of the grammar) between the values that interest us and certain closely related values. If the domain-of-interest can be embedded in a finite lattice with appropriate $\otimes$, ♦, and H functions, the values that interest us can be computed *together with* the closely related values using the iteration algorithm. This technique is similar to some inductive proofs in that, sometimes, we need to strengthen the induction hypothesis (that is, in order to prove a stronger result than is actually needed). In what follows, we will list several problems that can be solved in a lattice framework.

Problem 1: In Section 3 of this paper, we try to decide whether a regular language and a context-free language have common elements. Our strategy is to feed each sentence of the context-free grammar into the finite-state machine R of the regular language and examine the final states of R, that is, the set $\{q \mid R$ moves from its initial state to state q on input β, where β is a sentence of the context-free language$\}$. The regular language is transformed into a finite-state machine R. Let Q be the set of states of R. Let $F=\{f{:}2^Q \rightarrow 2^Q \mid f(\emptyset)=\emptyset, f(Q_1 \cup Q_2)=f(Q_1) \cup f(Q_2), Q_1 \subseteq Q, Q_2 \subseteq Q\}$. Elements of F are ordered as follows: Let f_1 and f_2 be two elements of F. $f_1 \le f_2$ if and only if, for all $Q' \subseteq Q$, $f_1(Q') \subseteq f_2(Q')$. The least element of F is $\lambda e.\ \emptyset$, and the greatest element is $\lambda e.$ *if* $e = \emptyset$, *then* $\emptyset$, *else* Q. Hence, $(F, \le)$ is a lattice. Note that function composition $\circ$ on F is associative and monotone and has an identity element $\lambda e.\ e$. For each $\alpha \in (N \cup T)^*$, let f_α be the "extended" state transition function of R as defined below: For any terminal symbol a, f_a is the element of F that is uniquely determined from the row (corresponding to a) of the state transition table of R. For , $\alpha, \beta \in (N \cup T)^*$, $f_{\alpha\beta} = f_\beta \circ f_\alpha$. For each nonterminal A of the context-free grammar, the set of all the A-productions $A \rightarrow \alpha \mid \beta \mid \ldots$is translated into the equation $f_A = f_\alpha \sqcup f_\beta \sqcup \ldots$. Now, the problem is cast in a lattice framework. We can use the iteration algorithm to find solutions for f_A. Finally, we are interested in $f_s(\{q_0\})$, where q_0 is the initial state of R, and S is the start symbol of the context-free grammar. Note that, in this framework, $f_\alpha = \lambda e.\ (e \oplus \alpha)$ in Section 3.

Problem 2: In Section 4 of this paper, we attempted to simplify a context-free grammar G and its parser with information provided by the COM of the scanner. The lattice framework is applicable to this problem as well. Let $G = (N, T, P, S)$. The characteristic finite-state machine $CFSM(G)$ of grammar G is transformed into a regular grammar $\overline{G}$ as follows. Each state s of $CFSM(G)$ corresponds to a nonterminal $\overline{S}$ in $\overline{G}$. Each transition $t \rightarrow {}^{x}s$ in $CFSM(G)$ corresponds to a production rule $\overline{S} \rightarrow \overline{T}\ x$ in $\overline{G}$. An additional production rule $\overline{U} \rightarrow u$ is added to $\overline{G}$, where $\overline{U}$ corresponds to the initial state of $CFSM(G)$, and u is a new terminal symbol in $\overline{G}$. The start symbol $\overline{X}$ of $\overline{G}$ corresponds to the accepting state of $CFSM(G)$. The lattice F is the same one as in Problem 1. Note that the set of terminal symbols of $\overline{G}$ is $N \cup T \cup \{u\}$. For $x \in (N \cup T)$, let f_x be the one computed in Problem 1. Let $f_u = \lambda e.e$. Define $f_{\alpha\beta} = f_\beta \circ f_\alpha$, where $\circ$ is defined in Problem 1. For each nonterminal $\overline{S}$ in $\overline{G}$, the set of all the $\overline{S}$-productions, $\overline{S} \rightarrow \alpha \mid \beta \mid ....$, is translated into the equation $f_{\overline{S}} = f_\alpha \sqcup f_\beta \sqcup .....$ Now, the problem is cast in a lattice framework. We can use the iteration algorithm to find solutions for $f_{\overline{S}}$ for each nonterminal $\overline{S}$ of $\overline{G}$. Finally, we are interested in $f_{\overline{X}}$, where $\overline{X}$ is the start symbol of $\overline{G}$. Note that, in this framework, $f_{\overline{S}}(Q) = \tau(Q, s)$, where s is a state of $CFSM(G)$.

Problem 3: Suppose we want to decide whether the lengths of all the sentences of a context-free language are multiples of 3. We can use the lattice framework to solve this problem. Let F be the powerset of the set $\{0,1,2\}$. $(F, \subseteq)$ is a lattice. Define $f_{\alpha\beta} = f_\alpha \bullet f_\beta$, where $s \bullet t = \{(a+b) \bmod 3 | a \in s, b \in t\}$. Note that $\bullet$ is an associative and monotone operator on F with an identity element $\{0\}$. Note that $f_\varepsilon = \{0\}$.For each terminal a, whose length is 1, $f_a = \{1\}$. For each nonterminal A, the set of all the A-productions $A \rightarrow \alpha \mid \beta \mid \ldots$ is translated into the equation $f_A = f_\alpha \cup f_\beta \cup .....$ We can use the iteration algorithm to solve the set of equations. The initial value of f_A, for each nonterminal A, is $\emptyset$. Finally, we are interested in whether $f_S = \{0\}$, where S is the start symbol of the context-free grammar.

Problem 4: (erasable nonterminals) Suppose we want to know whether a nonterminal A can derive the empty string. Let $F = \{\bot, \top\}$, with $\bot \leq \top$. Now define M as follows. For each terminal symbol a, $f_a = \bot$. Let $f_{\alpha\beta} = f_\alpha \sqcap f_\beta$. Note that $f_\varepsilon = \top$. The set of all the A-productions $A \rightarrow \alpha \mid \beta \mid \ldots.$ is transformed into the equation $f_A = f_\alpha \sqcup f_\beta \sqcup .....$ Then, we can solve the equations using the iteration algorithm. The initial value of f_A, for each nonterminal A, is $\bot$. Finally, $f_A = \top$ if and only if $A \rightarrow^{*} \varepsilon$.

Problem 5: (non-empty languages) A context-free language is *non-empty* if it can derive at least one sentence. This problem can also be solved in the lattice framework. Let $F = \{\bot, \top\}$, with $\bot \leq \top$. Define M as follows. For all terminal symbols a, $f_a = \top$. Let $f_{\alpha\beta} = f_\alpha \sqcap f_\beta$. Note that $f_\varepsilon = \bot$. The set of all the A-productions $A \rightarrow \alpha \mid \beta \mid \ldots.$ is transformed into $f_A = f_\alpha \sqcup f_\beta \sqcup ....$ Then, we an solve the equations using the iteration algorithm. The initial value for f_A, for each nonterminal A, is $\bot$. Finally, $f_S = \top$ if and only if the language is non-empty, where S is the start symbol of the grammar.

Problem 6: (context-free languages) Let $G = (N, T, P, S)$ be a context-free grammar. Let T^* be the set of all the strings of symbols from a vocabulary T. Let F be the powerset of T^*. The elements of F are ordered by means of subset containment. Note that $(F, \subseteq)$ is an (infinite) lattice. Let $f_a = \{a\}$ for each $a \in T$. Let $f_{\alpha\beta} = f_\alpha \nabla f_\beta$, where $s_1 \nabla s_2 = \{t_1 t_2 | t_1 \in s_1, t_2$

$\in s_2\}$. Note that ∇ is associative and monotone and has an identify element $\{\varepsilon\}$. Hence, $f_\varepsilon = \{\varepsilon\}$. For each $A \in N$, the set of all the A-productions $A \rightarrow \alpha \mid \beta \mid \ldots$ is translated into the equation $f_A = f_\alpha \cup f_\beta \cup \ldots$. For each $A \in N$, the initial value for f_A is $\emptyset$. In this example, f_A is the set of terminal strings derivable from the nonterminal A by the production rules of G. Since the lattice is infinite, the iteration algorithm can not terminate. However, any element of f_A, for any nonterminal A, can be generated in a finite number of iterations. Let S be the start symbol of the grammar. $f_S = L_G$. (This example is essentially the same as the one in [5].)

Problem 7: The *essential nonterminals* of a context-free grammar G are those that are used in the derivation of every sentence of L_G. Assume that G contains no useless symbols. Let F be the powerset of N (where N is the set of nonterminals of G). Elements of F are ordered by means of subset containment. Now, define M as follows. Let $f_a = \emptyset$ for all terminals a. Let $f_{\alpha\beta} = f_\alpha \cup f_\beta$. Note that $f_\varepsilon = \emptyset$. For each nonterminal A, the set of all the A-productions $A \rightarrow \alpha \mid \beta \mid \ldots\mid$ is transformed into the equation $f_A = (NONTERM\,(\alpha) \cup f_\alpha) \cap (NONTERM\,(\beta) \cup f_\beta) \cap \ldots$, where $NONTERM(\alpha)$ is the set of all nonterminals in α. For each nonterminal A, the initial value for f_A is N. We can solve for f_A using the iteration algorithm. Finally, $\{S\} \cup f_S$, where S is the start symbol of G, is the set of essential nonterminals.

Problem 8: (shortest-path problem) This problem is adapted from [6]. Given a directed graph in which each edge carries a non-negative distance, we wish to compute the shortest distance from a starting node s to all other nodes. The graph is transformed into a context-free grammar as follows. Each node a corresponds to a nonterminal A. Each edge $a \rightarrow^{e} b$ (where e is the identity of the edge in the graph) corresponds to a production rule $B \rightarrow Ae$, where e is a new terminal symbol in the grammar. The starting node s induces an additional rule $S \rightarrow \varepsilon$. Let F be $R^+ = R \cup \{\infty\}$(where ∞ is a positive infinite large number). Let f_e be the distance carried by the edge e. Let $f_{\alpha\beta} = f_\alpha + f_\beta$. Let the confluence operator be the minimum function. Then, f_A, for each nonterminal A, is the shortest distance from node s to node a in the given graph.

Problem 9: (data-flow analysis) Many data-flow analysis problems can be cast in lattice frameworks. We will demonstrate this application using an interprocedural data-flow analysis problem, taken from [4]. Assume that there are several procedures that call one another in a program. We wish to compute Use(A), the set of variables that may be used directly or indirectly during a call to procedure A. Let F be the powerset of the set of variables in the program, with $\cup$ and $\cap$ as the join and meet operators.For each procedure A, there is a nonterminal A and a terminal a in the associated grammar. For each nonterminal A, the set of all the A-productions is $A \rightarrow a \mid \alpha \ldots$, where $\alpha = B_1 B_2 \ldots$ and $B_1 B_2 \ldots$ are all the procedures that can be called directly by procedure A. The set of all the A-productions is translated into the equation $f_A = f_a \cup f_\alpha$ (that is, $\cup$ serves as the ♦ operator). Let $f_{\alpha\beta} = f_\alpha \cup f_\beta$ (that is, also serves as the $\otimes$ operator). Finally, for each terminal a, let $f_a =$ LocalUse(A), the set of variables that can be used locally in procedure A. Then we can solve the equations with the iteration algorithm. Note that the equation defining Use(A) in [4] is Use(A)= LocalUse (A) $\cup$ ($\cup_{\text{B is called by A}}$ Use(B)). A similar lattice can be used to solve the Def(A), the set of variables that can be assigned values during a call to procedure A.

6. ANALYSIS SCHEMES

We will now formalize the notion of a lattice framework and discuss the fundamental properties of a framework.

Defintion: An *analysis scheme* Ω is a 5-tuple $(F, \otimes, B, H, \blacklozenge)$, where F is a lattice with a join $\sqcup$ and a meet $\sqcap$ operation, $\otimes$ is an associative and monotone operation on F with an identity element, $B = \{f_a \mid a$ is a (terminal) symbol of the context-free language to which the scheme is applied$\}$, H is a monotone (in its second argument) function that maps a string (of terminals and nonterminals) and an element of F to an element of F (H is monotone in its second argument in the sense that if $f \leq f'$, then $H(\alpha, f) \leq H(\alpha, f')$, for all strings α), and $\blacklozenge$ is either $\sqcup$ or $\sqcap$. Let $f_{\blacklozenge}$ be the identity element of $\blacklozenge$. $\blacklozenge$ and $\otimes$ satisfy the following three axioms: (1) $f \otimes (h_1 \blacklozenge h_2) = (f \otimes h_1) \blacklozenge (f \otimes h_2)$, (2) $(h_1 \blacklozenge h_2) \otimes f = (h_1 \otimes f) \blacklozenge (h_2 \otimes f)$, and (3) $f \otimes f_{\blacklozenge} = f_{\blacklozenge} \otimes f = f_{\blacklozenge}$.

To apply an analysis scheme $\Omega = (F, \otimes, B, H, \blacklozenge)$ to a context-free grammar $G = (N, T, P, S)$, we first translate, for each $A \in N$, the set of all the A-production rules $A \rightarrow \alpha \mid \beta \mid \ldots$ into the equation $f_A = H(\alpha, f_\alpha) \blacklozenge H(\beta, f_\beta) \blacklozenge \ldots$, where $f_{\mu\xi}$ is an abbreviation of $f_\mu \otimes f_\xi$, for μ, $\xi \in (N \cup T)^*$. The iteration algorithm in Fig. 5 is used to compute the value of f_A. The initial value for f_A, for each $A \in N$, is $f_{\blacklozenge}$, the identity element of $\blacklozenge$. The result of the analysis is f_S, where S is the start symbol of grammar G.

An analysis scheme is applied to context-free grammars. We can check that all of the above problems make use of analysis schemes. There might be other constraints as to the kinds of context-free grammars and languages to which an analysis scheme is applicable. For instance, the analysis scheme in Problem 7 is applicable only to grammars having no useless symbols. These constraints are considered to be outside the schemes.

An analysis scheme can be used to compute the properties of both context-free *languages* and context-free *grammars*. For instance, Problems 1 through 6 are concerned with languages whereas Problem 7 is concerned with grammars. A common characteristic of Problems 1 through 6 lies in the H function, where $H(\alpha, f) = f$. By contrast, in Problem 7, $H(\alpha, f)$ depends not only on f, but also on α, (Note that α is the right-hand-side of a production rule.) This distinction is reasonable in that the properties of a context-free language should not depend on any particular choice of grammar. Hence, we can derive the following definition.

Definition: An analysis scheme is called a *language-analysis scheme* if $H(\alpha, f) = f$. It is called a *grammar-analysis scheme* otherwise.

This definition raises an interesting question: Since a context-free language may be defined by more than one context-free grammar, does a language-analysis scheme always compute the same answer no matter which context-free grammar is used in the computation? We will use a lemma to make this a claim. (In what follows, the notation $\blacklozenge S$ denotes the value of $s_1 \blacklozenge s_2 \blacklozenge \ldots$, where $s_1, s_2 \ldots$ are all the elements of S. Since $\blacklozenge$ is commutative and associative, the order of elements is not significant.)

Lemma 1: Suppose that a language-analysis scheme is applied to a context-free grammar G. For any nonterminal A in G, $f_A = \blacklozenge \{ f_\alpha \mid A \rightarrow^* \alpha, \alpha \in T^* \}$.

Proof: Let $f_{A,k}$ be the value for the nonterminal A after the k^{th} iteration of the loop in the iteration algorithm in Fig. 5. Note that $f_A = f_{A,\infty} = f_{A,k}$, for some finite k, since the iteration algorithm reaches a stable solution after a finite number of iterations. We can prove by induction on k that $f_{A,k} = \blacklozenge\ \{f_\alpha \mid \alpha \in T^*$, the height of the derivation tree of $A \rightarrow^* \alpha$ is at most $k\}$. For the sake of clarity, we assume that the confluence operator $\blacklozenge$ is $\sqcup$ in the proof. A parallel argument applies if $\sqcap$ is the confluence operator.

Base case. Let $k = 1$. Consider any nonterminal A, which is defined by the equation $f_A = f_{\alpha 1} \sqcup f_{\alpha 2} \sqcup$. In the computation of $f_{A,1}$, if $\alpha\, i$ contains any nonterminals, $f_{\alpha i} = f_{\min}$ (due to the properties of $\otimes$ and $f_{\min}$ in the analysis scheme). This term is dropped from the above equation since $f_{\min} \sqcup x = x \sqcup f_{\min} = x$ for any x. Therefore, $f_{A,l} = \sqcup \{f_\alpha | A \rightarrow \alpha$ is a production and $\alpha \in T^*\}$.

Induction hypothesis. Assume that, for $k < m$ and for any nonterminal A, $f_{A,k} = \sqcup \{f_\alpha | \alpha \in T^*$, the height of the derivation tree of $A \rightarrow^* \alpha$ is at most k $\}$.

Induction step. We need to prove that, for any nonterminal A, $f_{A,m} = \sqcup \{f_\alpha \mid \alpha \in T^*$, the height of the derivation tree of $A \rightarrow^* \alpha$ is at most $m\}$.

Consider any nonterminal A, which is defined by the equation $f_A = f_{\alpha 1} \sqcup f_{\alpha 2} \sqcup \ldots$ Note that, in the computation, $f_{A,m} = f_{\alpha 1,m\text{-}1} \sqcup f_{\alpha 2,m\text{-}1} \sqcup \ldots$. (The notation $f_{\alpha i,m\text{-}1}$ means that, for any nonterminal B in αi, $f_{B,m\text{-}1}$ will be used for f_B.) If $\alpha\, i$ contains a nonterminal, say B, then αi may be written as $\gamma B \delta$. Because $\otimes$ is associative, $f_{\alpha i,m\text{-}1} = f_{\gamma,m\text{-}1} \otimes f_{B,m\text{-}1} \otimes f_{\delta,m\text{-}1}$. By the induction hypothesis, $f_{B,m\text{-}1} = \sqcup \{f_\beta \mid \beta \in T^*$, the height of the derivation tree of $B \rightarrow^* \beta$ is at most m-1$\}$. Thanks to the distributive law of $\sqcup$ over $\otimes$, $f_{\alpha i,m\text{-}1} = \sqcup\{f_{\gamma,m\text{-}1} \otimes f_\beta \otimes f_{\delta,m\text{-}1} \mid \beta \in T^*$, the height of the derivation tree of $B \rightarrow^* \beta$ is at most m-1$\}$. By applying the same argument to all nonterminals in γ and δ, we know that $f_{\alpha i,m\text{-}1} = \sqcup \{f_\mu \mid \mu \in T^*$, the height of the derivation tree of $A \rightarrow \alpha i \rightarrow^* \mu$ is at most $m\}$. Since the equation $f_A = f_{\alpha 1} \sqcup f_{\alpha 2} \sqcup \ldots$ is translated from the set of *all* the A-productions, any string $\mu \in T^*$ such that the height of the derivation tree $A \rightarrow^* \mu$ is at most m is considered in one of $f_{\alpha i,m\text{-}1}$. Therefore, $f_{A,m} = \sqcup\{f_\mu \mid \mu \in T^*$, the height of the derivation tree of $A \rightarrow^* \mu$ is at most $m\}$. This completes the induction proof. □

The following theorem asserts that, if a language-analysis scheme is used to compute the properties of a context-free language, the result computed by the scheme does not depend on the particular choice of context-free grammar. The theorem is a corollary to the above lemma.

Theorem 1: (Soundness of language-analysis schemes) Assume that the context-free grammars G and $\overline{G}$ define the same context-free language. Let S and $\overline{S}$ be the start symbols of G and $\overline{G}$, respectively. Then, the results obtained by applying a language-analysis scheme to G and $\overline{G}$ are the same. That is, $f_S = f_{\overline{S}}$.

Definition: Let Ω be a language-analysis scheme. Let L be a context-free language. The notation $\Omega(L)$ means f_S, where S is the start symbol of a context-free grammar defining L.

Below, we list several fundamental properties of language-analysis schemes. The proofs of these theorems are based on Lemma 1 and are easy enough to be omitted.

Theorem 2: Let Ω be a language-analysis scheme. Let L and M be context-free languages. $\Omega(L \cup M) = \Omega(L) \blacklozenge \Omega(M)$.

Theorem 3: Let Ω be a language-analysis scheme. Let L and M be context-free languages. Assume that $L \subseteq M$. If $\sqcup$ is the confluence operator, $\Omega(L) \leq \Omega(M)$. If $\sqcap$ is the confluence operator, $\Omega(L) \geq \Omega(M)$.

Theorem 4: Let Ω be a language-analysis scheme. Let L and M be context-free languages. If $\sqcup$ is the confluence operator, $\Omega(L \cap M) \leq \Omega(L) \sqcap \Omega(M)$. If $\sqcap$ is the confluence operator, $\Omega(L \cap M) \geq \Omega(L) \sqcup \Omega(M)$.

Theorem 5: Let Ω be a language-analysis scheme. Let L and M be context-free languages. $\Omega(LM) = \Omega(L) \otimes \Omega(M)$. ($LM$ is a language consisting of the concatenation of a sentence of L and a sentence of M.)

Theorem 6: Let Ω be a language-analysis scheme. Let L be a context-free language. $\Omega(L^*) = \blacklozenge\{\ \Omega(L)^k \mid k \geq 0\ \}$, where $\Omega(L)^0$ is the identity element of $\otimes$ and $\Omega(L)^{k+1} = \Omega(L) \otimes \Omega(L)^k$. ($L^*$ is a language consisting of the concatenation of zero or more sentences of L.)

Theorem 7: Let Ω be a language-analysis scheme. $\Omega(\{\varepsilon\})$ = the identity element of $\otimes$.

Theorem 8: Let Ω be a language-analysis scheme. $\Omega(\varnothing) = f_{\blacklozenge}$, the identity element of $\blacklozenge$.

We will now turn to another important characterization of language-analysis schemes: any difference between two context-free languages is witnessed by a language-analysis scheme. The essence of this characterization lies in the condition that the lattice F in a language-analysis scheme is finite. If we can use infinite lattices, the following theorem is trivial. The scheme in Problem 6 in the previous section is the witness. It is not so obvious if F must be finite.

Theorem 9: Let L and M be context-free languages. Assume that $L \neq M$. Then, there is a language-analysis scheme Ω such that $\Omega(L) \neq \Omega(M)$.

Proof: Since $L \neq M$, without loss of generality, we may assume that there is a sentence $\alpha \in L$ but $\alpha \notin M$. Let j be the length of α. If $j = 0$, the scheme in Problem 4 is the witness. Assume $j > 0$. Let $k = j+1$.

We will now construct a language-analysis scheme Ω, as follows: Let $T^{\leq k} = \{\alpha \mid \alpha \in T^*$, the length of α is at most $k\}$. Let F be the powerset of $T^{\leq k}$. Elements of F are ordered by means of subset containment. Let $f_{\alpha\beta} = f_\alpha \Delta f_\beta$, where $f_1 \Delta f_2 = \{\text{first}k(t_1t_2) \mid t_1 \in f_1, t_2 \in f_2\}$ and first$k(t)$ computes the first k symbols of the string t. It can be shown that Δ is associative and monotone and has an identity element $\{\varepsilon\}$. For $a \in T$, let $f_a = \{a\}$. Let $\cup$ be the confluence operator. Hence, $f_{\blacklozenge} = \varnothing$. It can be verified that $\cup$ and Δ satisfy the following three axioms: (1) $f \Delta (h_1 \cup h_2) = (f \Delta h_1) \cup (f \Delta h_2)$, (2) $(h_1 \cup h_2) \Delta f = (h_1 \Delta f) \cup (h_2 \Delta f)$, and (3) $f \Delta \emptyset = \emptyset \Delta f = \emptyset$. In essence, $\Omega(L) = \{\text{first}k(t) \mid t \in L\}$. We can see that $\alpha \in \Omega(L)$ but $\alpha \notin \Omega(M)$.□

Corollary 1: Let L and M be context-free languages. If $\Omega(L) = \Omega(M)$ for all language-analysis schemes Ω, then $L = M$.

7. LANGUAGE-ANALYSIS SCHEMES BASED ON INFINITE LATTICES

The requirement that F be a *finite* lattice in an analysis scheme can be relaxed. Finiteness is used to guarantee termination of the iteration algorithm. The iteration algorithm also terminates for the class of *finite-height* lattices, in which the longest path from $f_{\max}$ to $f_{\min}$ is of finite length.

It is interesting to briefly investigate the case where F in a language-analysis scheme is infinite. The main problem with infinite lattices is that the iteration algorithm can not terminate. However, we can show that termination is an inherent property of context-free languages. That is, termination is independent of the particular grammar used to describe the language.

Theorem 10: Let Ω be a language-analysis scheme based on an infinite lattice. Suppose that grammars G_1 and G_2 define the same language. The application of Ω to G_1 terminates if and only if the application of Ω to G_2 terminates. Furthermore, when the applications terminate, the applications yield the same values.

Proof: Let L be the language defined by G_1 (or, equivalently, G_2). Suppose that the application of Ω to G_1 terminates after k_1 steps. Let $L_1 = \{\alpha \mid \alpha \in L$ and the height of the derivation tree of α based on G_1 is at most $k_1\}$. Since the application of Ω to G_1 terminates after k_1 steps, $\blacklozenge\{f_\alpha \mid \alpha \in L_1\} = \blacklozenge\{f_\alpha \mid \alpha \in L\}$. Furthermore, L_1 is finite. Let $k_2 = \max\{h \mid h$ is the height of the derivation tree of α based on G_2, where $\alpha \in L_1\}$. Let $L_2 = \{\beta \mid \beta \in L$ and the height of the derivation tree of β based on G_2 is at most $k_2\}$. Note that $L_1 \subseteq L_2 \subseteq L$. Because $\blacklozenge$ is monotone, we have $\blacklozenge\{f_\alpha \mid \alpha \in L_1\} = \blacklozenge\{f_\alpha \mid \alpha \in L_2\} = \blacklozenge\{f_\alpha \mid \alpha \in L\}$. This implies that the application of Ω to G_2 will terminate by at most k_2 steps.□

Once we know that the application of terminates, the properties listed in the previous section all hold.

8. CONCLUSIONS AND RELATED WORK

We have characterized a lattice framework for analyzing context-free grammars and context-free languages. This lattice framework is motivated by a technique for simplifying parsers using information derived from the associated scanners and can be used in many other applications, including data-flow analysis. The crux of the lattice framework is to view the → symbols in the production rules of grammars as defining equations of closely related entities. When the domain-of-interest is a finite lattice, the equations can be solved using an iteration algorithm. An introduction to lattice theory can be found in [7].

The lattice framework can be viewed as a form of induction [8]. (This is no surprise at all since context-free grammars are an inductive definition of context-free languages.) Imagine that we want to prove a property of a context-free language $\mathfrak{P}(L)$. We may not be able to prove it directly if $\mathfrak{P}(L)$ *alone* is the induction hypothesis. Sometimes, the proof proceeds smoothly when we strengthen the induction hypothesis to $\mathfrak{P}^A$ (L_A) **and** $\mathfrak{P}^B$ (L_B) **and**…., where *A*, *B*, … are all the nonterminals of a grammar defining *L* and $\mathfrak{P}^A$,$\mathfrak{P}^B$,… are properties closely related to $\mathfrak{P}$. Remember that L_A are the strings of terminals that are derivable from the nonterminal *A*. This induction hypothesis means that, in order to prove that *L* has the property $\mathfrak{P}$, we actually need to prove that L_A has the property $\mathfrak{P}^A$, that L_B has the property $\mathfrak{P}^B$, *etc*. For instance, consider the language defined by the grammar consisting of two production rules: $S \rightarrow aX$ and $X \rightarrow aS \mid a$. Now, we want to prove that all sentences of L_S have even length. In an inductive proof, the induction hypothesis could be that **all the sentences of L_S have even length and that all sentences of L_X have odd length**. This is exactly what a scheme similar to that in Problem 3 (in Section 5) will compute. The challenge for an inductive proof is to find an appropriate induction hypothesis and to prove it. By contrast, in a language-analysis scheme, the difficulty lies in constructing an appropriate lattice. When such a lattice is constructed, the iteration algorithm automatically computes the “induction hypothesis” (which is f_A, f_B etc.).

Knuth’s work [6] was the first on computing properties of grammars. Our lattice framework differs from Knuth’s work in four aspects: (1) Knuth uses real numbers ($R \cup \{\infty\}$), which is a total order, whereas we allow general lattices; (2) Knuth’s algorithm transforms production rules into independent *superior functions* whereas we use a single operator $\otimes$ to transform all the productions; (3) Knuth’s superior functions need to satisfy an additional constraint *viz*, the value of a function application must be at least as large as any of its arguments; and (4) Knuth uses only the minimum function, which corresponds to the $\sqcap$ operator, as the confluence operator. By contrast, our lattice framework uses $\sqcup$ as well as $\sqcap$ as the confluence operator. On the other hand, because Knuth assumed certain properties that allow him to adopt a generalization of Dijkstra’s shortest-path algorithm, Knuth’s work was particularly efficient. Moencke and Wilhelm [9]developed a theory similar to the work reported for flow analysis in attribute grammars. Their work extends Knuth’s in that they employ partial orders, rather than total orders. Ramalingam in his thesis [10] pointed out that interprocedural dataflow analysis frameworks, such as [11], can be viewed as instances of grammar-analysis problems. In these problems, the grammar describes valid execution paths. Ramalingam’s work can be viewed as a generalization of Knuth’s and Moencke and Wilhelm’s results. All these researchers did not distinguish between language-analysis and grammar-analysis schemes. We also address various properties of language-analysis schemes, in particular, the independence of the computation results with respect to the particular grammars used in the computation.

The lattice framework bears some similarity with denotational semantics [7]. In denotational semantics, each terminal symbol corresponds to a function, and $\otimes$ corresponds to a function application. One important difference lies in the fact that function application is *not* associative. By contrast, $\otimes$ is an associative operator in our framework. Another difference is that denotational semantics compute a property (that is, the *meaning*) of a sentence in the language whereas our framework computes a property of the (context-free) language.

This lattice framework is similar to some circular attribute grammars [12]: We may view f_A, f_B,.... as attributes of the nonterminals and $f_A = h_\alpha \blacklozenge h_\beta \blacklozenge$... as attribution equations. Clearly, this "attribute grammar" was circular in general. However, a finite-lattice framework guarantees that the iteration algorithm reaches a stable solution in a finite number of iterations.

The Mealy-machine description of a scanner is motivated by a study of the look-ahead problem in lexical analysis [13]. The technique for handling the look-ahead problem is similar, in spirit, to the pattern-matching algorithm of [14] and the string matching algorithm of [15].

ACKNOWLEDGMENT

The author wishes to thank Professor Thomas Reps of the University of Wisconsin, Madison, for many helpful discussions and for pointing out many related works.

REFERENCES

1. W. Yang, "Mealy machines are a better model of lexical analyzers," *Computer Languages*, Vol. 22, No. 1, 1996, pp. 27-38.
2. J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.
3. A.V. Aho, R. Sethi and J.D. Ullman, *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, Reading, MA, 1986.
4. C.N. Fischer and R.J. LeBlanc, Jr., *Crafting a Compiler with C*, Benjamin/Cummings, Reading, MA, 1991.
5. D. Mandrioli and C. Ghezzi, *Theoretical Foundations of Computer Science*, Addison-Wesley, Reading, MA, 1987.
6. D.E. Knuth, "A generalization of Dijkstra's algorithm," *Information Processing Letters*, Vol. 6, No. 1, 1977, pp. 1-5.
7. J.E. Stoy, *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*, MIT Press, Cambridge, MA, 1977.
8. R.M. Burstall, "Proving properties of programs by structural induction," *The Computer Journal*, Vol. 12, No. 1, 1969, pp. 41-48.
9. U. Moencke and R. Wilhelm, "Grammar flow analysis," in *Proceedings of the International Summer School SAGA*, *Lecture Notes in Computer Science*, Vol. 545, 1991, pp. 151-186.
10. G. Ramalingam, "Bounded incremental computation," Ph.D. dissertation TR-1172, Computer Science Deptartment, University of Wisconsin, 1993.
11. M. Sharir and A. Pnueli, "Two approaches to interprocedural data flow analysis," in S. S. Muchnick and N.D. Jones (eds.), *Program Flow Analysis: Theory and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1981, pp. 189-223.
12. D.E. Knuth, "Semantics of context-free languages," *Mathematical System Theory*, Vol. 2, No. 2, 1968, pp. 127-145. Correction, *ibid.*, Vol. 5, No. 1, 1971, pp. 95-96.
13. W. Yang, "On the look-ahead problem in lexical analysis," *ACTA Informatica*, Vol. 32, 1995, pp. 459-476.
14. D.E. Knuth, J.H. Morris, Jr. and V.R. Pratt, "Fast pattern matching in strings," *SIAM*

Journal on Computing, Vol. 6, No. 2, 1977, pp. 323-350.
15. A.V. Aho and M.J. Corasick, “Efficient string matching: An aid to bibliographic search, ” *Communication ACM*, Vol. 18, No. 6, 1975, pp. 333-340.

Wuu Yang (**楊武**) received his B.S. degree in computer science from National Taiwan University in 1982 and the M.S. and Ph.D. degrees in computer science from the University of Wisconsin, Madison, in 1987 and 1990, respectively. Currently, he is an associate professor at National Chiao-Tung University, Taiwan, Republic of China. Dr. Yang’s research interests include programming languages and compilers, attribute grammars, and parallel and distributed computing. He is also very interested in the study of human languages and human intelligence.