Bidirectional Ascendant Parsing for Natural Language Processing

Ştefan ANDREI
Faculty of Computer Science, "Al.I.Cuza" University
Str. Berthelot, nr. 16, 6600, Iaşi, România.
E-mail: stefan@infoiasi.ro

Abstract

The paper emphasizes some subclasses of context free grammars for which there exists a parallel approach useful for solving the membership problem. We combine the classical style of LR parsers attached to a grammar G with a "mirror" process for G. The input word will be analysed from both sides using two processors. We present the *general bidirectional parser* (for any context free language) using a nondeterministic device. A general MIMD model for describing the bidirectional parsing is presented. Our general bidirectional parser can be also used as a deterministic model for the known $LR(k)$ and $RL(k)$ parsers. Accordingly, the membership problem may be solved in linear time complexity with a parallel algorithm. Finally, we present an example of a context free grammar useful for describing syntactic dependencies for English language.

1 Introduction

In recent years, recognition algorithms, both sequential and parallel, for context free grammars (with applications to parsing programming languages) or for non context free languages (with applications to computational linguistic) have been the subject of study by many computer scientists ([7, 11]). In ([11]) was described a formal framework for bidirectional tabular parsing of general context free languages, and some applications to natural language processing were studied. Moreover, an algorithm for head-driven parsing and a general algorithm for island-driven parsing were studied.

The main purpose of this paper is to provide parallel parsing for (subclasses of) context free grammars using two processors. Similar ideas concerning parsing of the input word from both sides were presented in [9, 8]. However, the mention papers do not construct effectively any parsers and do not mention any parallel model of computers for such computation.

In this paper, we define a new parser for the class of context free languages. The input word is analysed from both sides, but the parse strategy is in up-to-up sense (LR and RL styles are combined for this parallel strategy). Some of the theoretical results used in this paper were reported previously in [3, 4].

The associated derivation tree corresponding to the input word w is down to up traversed by both processors, i.e. from the leaves to its root. Only one processor will be strongly active after the parallel algorithm "meets" the "middle" of the input word:

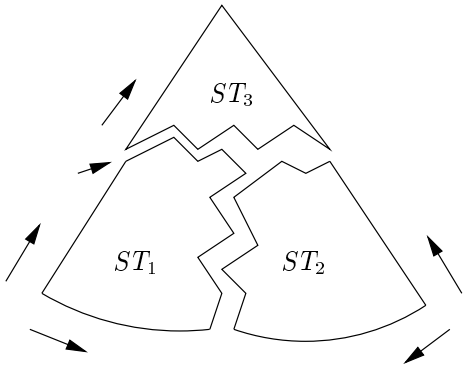

Figure 1. Up-to-up bidirectional parsing

The "derivation forests" ST_1 and ST_2 will be parsed in parallel, and, finally, the subtree ST_3 will be parsed in a sequential way.

The second section contains the definition of the general up-to-up bidirectional parser. The correctness of this parser is proven as the main result of this section (Theorem 2.1). The third section presents a very convenient parallel approach for describing the general up-to-up bidirectional parsing strategy. The chosen model is a MIMD computer. Two processors `P1` and `P2` analyse the input word from both sides. Theorems 3.1 and 3.2 ensure the finiteness and correctness of our parallel algorithm. The next section points out the definition of some new subclasses of context free grammars. We called them $LR - RL$ grammars. These grammars can be viewed as some combinations between classical LR grammars and the mirror ones. The fifth section presents an example of a simple context free grammar useful for describing syntactic dependencies for English language. Some open problems are emphasized in Conclusions.

2 The General Bidirectional Parser

We suppose the reader familiar with the notions as context free grammars, words ([5, 10]). Let $\alpha = \alpha_1\,\alpha_2 \ldots \alpha_k$ be a word over V. Then

$$ {}^{(m)}\alpha = \begin{cases} \alpha_1\,\alpha_2 \ldots \alpha_m & \text{if } m \leq k \\ \alpha & \text{otherwise} \end{cases} \text{ and } \alpha^{(n)} = \begin{cases} \alpha_{k-n+1}\,\alpha_{k-n+2} \ldots \alpha_k & \text{if } n \leq k \\ \alpha & \text{otherwise.} \end{cases} $$

Notation 2.1 *Let $G = (V_N, V_T, S, P)$ be a context free grammar, $V = V_N \cup V_T$ and V' be the set $V_N \times \mathbf{N} \times \mathbf{N}$. Let $h : V \cup V' \to V$ be a function given by $h(X) = X$, $\forall\ X \in V$ and $h(X_{b,e}) = X$, $\forall\ X_{b,e} \in V'$. This function can be easily extended to an homomorphism $h : (V \cup V')^* \to V^*$ such as $h(\lambda) = \lambda$ and $h(X_1X_2...X_n) = h(X_1) \cdot h(X_2) \cdot \ldots \cdot h(X_n)$, $\forall\ X_1,\ X_2,\ \ldots,\ X_n \in V \cup V'$, $\forall\ n \geq 2$ ($\cdot$ being the catenation operation).*

The occurrence of $A_{b,e}$ signifies that A is the label of the derivation subtree of root v in the forests ST_1 or ST_2 (Figure 1), and the frontier has the corresponding right most derivation $[r_b, r_{b+1}, ..., r_e]$. In the following definition, we shall describe our up-to-up bidirectional parser which contains 14 transitions. Some of the transitions can be "compacted" (e.g. 1^0, 5^0 and 7^0 or 2^0, 6^0 and 7^0, a.s.o.). We defined explicitely all the possible cases (similar to "cartesian product" of the classical transitions)

because it is easier to prove the correctness of our model.

Definition 2.1 *Let $G = (V_N, V_T, S, P)$ be a context free grammar. Let $\mathcal{C} \subseteq \{s_1, s_2\} \times \{1, 2, ..., |P|\}^* \times \#(V \cup V')^* \times \#V_T^* \# \times (V \cup V')^* \# \times \{1, 2, ..., |P|\}^* \times \{1, 2, ..., |P|\}^* \cup \{ACC, REJ\}$ be the set of all possible configurations, where # is a special character (a new terminal symbol). The* **general up-to-up bidirectional parser** *(denoted by $G_u BP(G)$) is the pair $(\mathcal{C}_0, \vdash)$, where $\mathcal{C}_0 = \{(s_1, \lambda, \#, \#w\#, \#, \lambda, \lambda) \mid w \in V_T^*\} \subseteq \mathcal{C}$ is called the set of* **initial configurations**. *The first component is the state, the second (ordered from right to left) and the last but one (ordered from left to right) components of a configuration are for storing the partial syntactic analysis. The last component is for storing the final syntactic analysis. The third and the fifth components are the work - stacks. The fourth component represents the current content of the input word. The* **transition relation** *($\vdash \subseteq \mathcal{C} \times \mathcal{C}$, sometimes denoted by $\vdash_{G_u BP(G)}$) between configurations is given by:*

1^0 *Shift-Shift:* $(s_1, \pi_1, \#\alpha, \#a\,u\,b\#, \beta\#, \pi_2, \lambda) \vdash (s_1, \pi_1, \#\alpha\,a, \#u\#, b\,\beta\#, \pi_2, \lambda)$;

2^0 *Reduce-Shift:* $(s_1, \pi_1, \#\alpha\,\beta, \#u\,b\#, \gamma\#, \pi_2, \lambda) \vdash (s_1, r_1\pi_1, \#\alpha\,A_{b',e'}, \#u\#, b\,\gamma\#, \pi_2, \lambda)$, *where* $r_1 = A \to h(\beta) \in P$, $e' = |\pi_1| + 1$, $b' = \min\{|\pi_1| + 1, \min\{b'' \mid C_{b'',e''} \in \beta\}\}$;

3^0 *Shift-Reduce:* $(s_1, \pi_1, \#\alpha, \#a\,u\#, \gamma\,\beta\#, \pi_2'\,\pi_2'', \lambda) \vdash (s_1, \pi_1, \#\alpha\,a, \#u\#, B_{b,e}\,\beta\#, \pi_2'\,r_2\pi_2'', \lambda)$, *where* $r_2 = B \to h(\gamma) \in P$, $e = |\pi_2'\pi_2''| + 1$, $b = \min\{|\pi_2'\,\pi_2''| + 1, \min\{b' \mid D_{b',e'} \in \gamma\}\}$.

4^0 *Reduce-Reduce:* $(s_1, \pi_1, \#\alpha\beta, \#u\#, \varepsilon\gamma\#, \pi_2'\,\pi_2'', \lambda) \vdash (s_1, r_1\pi_1, \#\alpha A_{b_1,e_1}, \#u\#, B_{b_2,e_2}\gamma\#,$
$\pi_2'\,r_2\,\pi_2'', \lambda)$, *where* $r_1 = A \to h(\beta) \in P$, $r_2 = B \to h(\varepsilon) \in P$, $e_1 = |\pi_1| + 1$, $e_2 = |\pi_2'\,\pi_2''| + 1$, $b_1 = \min\{|\pi_1| + 1, \min\{b_1' \mid C_{b_1',e_1'} \in \beta\}\}$ *and* $b_2 = \min\{|\pi_2'\,\pi_2''| + 1, \min\{b_2' \mid D_{b_2',e_2'} \in \varepsilon\}\}$;

5^0 *Shift-Stay:* $(s_1, \pi_1, \#\alpha, \#a\,u\#, \beta\#, \pi_2, \lambda) \vdash (s_1, \pi_1, \#\alpha\,a, \#u\#, \beta\#, \pi_2, \lambda)$;

6^0 *Reduce-Stay:* $(s_1, \pi_1, \#\alpha\,\beta, \#u\#, \gamma\#, \pi_2, \lambda) \vdash (s_1, r_1\pi_1, \#\alpha\,A_{b',e'}, \#u\#, \gamma\#, \pi_2, \lambda)$, *where* $r_1 = A \to h(\beta) \in P$, $e' = |\pi_1| + 1$, $b' = \min\{|\pi_1| + 1, \min\{b'' \mid C_{b'',e''} \in \beta\}\}$;

7^0 *Stay-Shift:* $(s_1, \pi_1, \#\alpha, \#u\,b\#, \beta\#, \pi_2, \lambda) \vdash (s_1, \pi_1, \alpha, \#u\#, b\,\beta\#, \pi_2, \lambda)$;

8^0 *Stay-Reduce:* $(s_1, \pi_1, \#\alpha, \#u\#, \gamma\,\beta\#, \pi_2'\,\pi_2'', \lambda) \vdash (s_1, \pi_1, \#\alpha, \#u\#, B_{b,e}\,\beta\#, \pi_2'\,r_2\pi_2'', \lambda)$, *where* $r_2 = B \to h(\gamma) \in P$, $e = |\pi_2'\pi_2''| + 1$, $b = \min\{|\pi_2'\pi_2''| + 1, \min\{b' \mid D_{b',e'} \in \gamma\}\}$;

9^0 *Possible-accept:* $(s_1, \pi_1, \#\alpha, \#\#, \beta\#, \pi_2, \lambda) \vdash (s_2, \pi_1, \#\alpha, \#\#, \beta\#, \pi_2, \lambda)$;

10^0 *Shift-Terminal:* $(s_2, \pi_1, \#\alpha, \#\#, a\,\beta\#, \pi_2, \pi_3) \vdash (s_2, \pi_1, \#\alpha\,a, \#\#, \beta\#, \pi_2, \pi_3)$;

11^0 *Shift-Nonterminal:* $(s_2, \pi_1, \#\alpha, \#\#, A_{b,e}\,\beta\#, \pi_2'\pi_2'', \pi_3) \vdash (s_2, \pi_1, \#\alpha\,A, \#\#, \beta\#, \pi_2'', \pi_2'\pi_3)$, *where* $\pi_2' = [r_b, r_{b+1}, ..., r_e]$ *(e.g.* $|\pi_2'| = e - b + 1$*)*;

12^0 *Reduce:* $(s_2, \pi_1'\pi_1'', \#\alpha\,\beta, \#\#, \gamma\#, \pi_2, \pi_3) \vdash (s_2, \pi_1'', \#\alpha\,A, \#\#, \gamma\#, \pi_2, r_1\pi_3\pi_1')$, *where* $r_1 = A \to h(\beta) \in P$, $\beta = u_1\,B_{b,e} \ldots u_m\,C_{b',e'}\,\beta'$, $\beta' \in V^*$, u_1, ..., $u_m \in V_T^*$, $|\pi_1'| = e' - b + 1$;

13^0 *Accept:* $(s_2, \lambda, \#S, \#\#, \#, \lambda, \pi) \vdash ACC$;

14^0 *Reject:* $(s_1, \pi_1, \#\alpha, \#u\#, \beta\#, \pi_2, \lambda) \vdash REJ$ *and* $(s_2, \pi_1, \#\alpha, \#\#, \beta\#, \pi_2, \pi_3) \vdash REJ$ *if no transitions of type* 1^0, 2^0, ..., 13^0 *can be applied.*

Our model is a two-stack machine ([5]). The differences consist in the existence of two heads (instead of only one) which may read the input tape, and two output tapes which can be accessed only in write style, and has only two states. Therefore, our model can simulate a Turing machine.

Theorem 2.1 *([3]) Let G be a context free grammar, S being its start symbol. Then* $(s_1, \lambda, \#, \#w\#, \#, \lambda, \lambda) \overset{*}{\underset{G_u BP(G)}{\vdash}} (s_2, \lambda, \#S, \#\#, \#, \lambda, \pi) \overset{13^0}{\underset{G_u BP(G)}{\vdash}} ACC$ *iff* $S \overset{\pi}{\underset{G,rm}{\Longrightarrow}} w$.

3 Parallel Approach for General Bidirectional Parsing

In this section, we present a very convenient parallel approach for describing the general up-to-up bidirectional parsing strategy. Our model is a MIMD (multiple instruction stream and multiple data stream) computer. We consider two processors `P1` and `P2` which operate asynchronously and share a common memory ([1]).

In the following, we shall present a parallel algorithm which describes the general up-to-up bidirectional parsing strategy. In fact, our parallel algorithm (denoted by (PAR-UUBP)) clearly follows Definition 2.1 of the general up-to-up bidirectional parser. Our algorithm uses the following variables:

- `w`$\in V_T^*$ the input word, `n`$= |$`w`$|$;
- `i1`, `i2` two counters for the current positions in `w`;
- `accept` a boolean variable which takes the true value iff `w`$\in L(G)$;
- `Stack1`, `Stack2` two working stacks for `P1` and `P2`;
- `Output_tape1`, `Output_tape2` the output tapes of `P1` and `P2` for storing the partial syntactic analysis;
- `Output_tape` the output tape for storing the global syntactic analysis;
- `exit` a boolean variable which is true iff `P1` or `P2` detect the non-acceptance of `w`.

The variables `w`, `i1`, `i2`, `accept`, `Output_tape`, `exit` are stored in the common memory. We use some predefined procedures, such as:

- `pop(Stack,`α`)` - the value of α will be the string of length $|\alpha|$ starting from the first symbol of `Stack`; after that, the string α is removed from `Stack`;
- `push_first(Stack,A)` - add to the content of `Stack` the symbol `A`; `A` will be the new top of `Stack`;
- `push_last(Stack,`α`)` - add to the content of `Stack`, starting from the last symbol of `Stack`, the string α; `Stack` will have the same top.

Now, the method of parallel algorithm (PAR-UUBP) can be pointed out (we suppose that the context free grammar $G = (V_N, V_T, S, P)$ is already read):

```
begin
 read(n); read(w); i1:=1; i2:=n; accept:=false; exit:=false;
 repeat in parallel
  action1(P1); action2(P2)
 until (i1>=i2) or (exit=true);
 if (exit = true) then accept := false
 else repeat action3(P1, P2) until (exit = true);
 if (accept = true) then
  write('w is accepted and has the right syntactical analysis',Output_tape);
 else write('w is not accepted');
end.
```

It remains to describe the procedures `action1(P1)`, `action2(P2)` and `action3(P1,P2)`.

```
procedure action1(P1);
begin
  case
    if (∃ r1 = A→ h(α) ∈ P, α is in Stack1 starting from the top) then begin
      /* reduce action */
      let α := u1 C_{b,e} ... um D_{b',e'} α', where u1, ..., um ∈ V_T*, α' ∈ V* − V'*;
      e'' := |Output_tape1| + 1;
```

```
      if (α does not contain any symbol from V' − V)  then  b'' := e''
      else b'' := min{e'', b};
      pop(Stack1,α); push_first(Stack1,A_{b'',e''}); push_first(Output_tape1,r1);
    end;
    if (i1 <= i2) then begin     /* shift action */
      push_first(Stack1,w[i1]); i1 := i1+1;
    end
    otherwise: begin    /* backtrack is needed; */
      if (all the backtrack steps are over) and (still no reduce or shift
        action could be made) then exit := true;
    end
end;
```

The description of procedure `action2(P2)` is very similar to `action1(P1)`.

```
procedure action2(P2);
begin
  case
    if (∃ r2 = A→ h(β) ∈ P, β is in Stack2 starting from the top) then begin
      /* reduce action */
      let β := u_1 C_{b,e} ... u_m D_{b',e'} β', where u_1, ..., u_m ∈ V_T^*, β' ∈ V^* − V'^*
      e'' := |Output_tape2| + 1;
      if (β does not contain any symbol from V' − V) then b'' := e''
      else b'' := min{e'', b};
      pop(Stack2,β); push_first(Stack2,B_{b'',e''}); push_first(Output_tape2, r2);
    end;
    if (i1 < i2) then begin     /* shift action */
      push_first(Stack2,w[i2]); i2 := i2-1;
    end
    otherwise: begin    /* backtrack is needed; */
      if (all the backtrack steps are over) and (still no reduce or shift
        action could be made) then exit := true;
    end
end;
```

Finally, we describe the procedure `action3(P1,P2)` in a sequential way. The goal is to simulate the transitions 10^0, 11^0, 12^0 and 13^0 for $G_u BP(G)$ from Definition 2.1. The input tape is now empty, i.e. `w` has been already read (of course, if `exit` has the value $false$), but we read symbols from `Stack2`

(send by processor P2) modifying the content of the Output_tape1 and Output_tape2 putting the results in Output_tape.

```
procedure action3(P1,P2);
begin
  case
    if (∃ r1 = A→ h(α) ∈ P, α is in Stack1 starting from the top) then begin
      /* reduce action */
      let α = u1 C_{b,e} ... α' D_{b',e'} α'', where u ∈ V_T*, α'' ∈ V*, α' ∈ (V ∪ V')*;
      let π1' from Output_tape1 such that |π1'| = e' − b + 1;
      pop(Output_tape1,π1'); pop(Stack1,α); push(Stack2,A);
      push_first(Output_tape,r1); push_last(Output_tape,π1');
    end;
    if (top of Stack2 is a  terminal symbol) then begin
      /* shift-terminal action */
      pop(Stack2,a), where a ∈ V_T; push_first(Stack1,a);
    end;
    if (top of Stack2 is from V') then begin /*shift-nonterminal action*/
      pop(Stack2,A_{b,e}); push_first(Stack1,A);
      let π2' from Output_tape2 such that |π2'| = e − b + 1;
      pop(Output_tape2,π2'); push_first(Output_tape,π2');
    end;
    if (Output_tape1=∅) and (Output_tape2=∅) and (Stack1=S) and (Stack2=∅)
    then begin  accept:=true; exit:=true  end;
  otherwise: begin    /* backtrack is needed; */
    if (all the backtrack steps are over) and (still no reduce or shift
      action could be made) then exit := true
  end;
end;
```

Theorem 3.1 *([3]) - finiteness of Algorithm (PAR-UUBP))*
The Algorithm (PAR-UUBP) performs a finite number of steps until terminates its execution.

Theorem 3.2 *([3]) - correctness of Algorithm (PAR-UUBP))*
For a given context free grammar $G = (V_N, V_T, S, P)$ and $w \in V_T^*$ as its input, Algorithm (PAR-UUBP) gives the answer "`w is accepted and has the right syntactical analysis` π*" if* $S \underset{G,rm}{\overset{\pi}{\Longrightarrow}} w$ *and "*`w is not  accepted.`*" otherwise.*

4 Deterministic Bidirectional Parsing for LR-RL Grammars

Deterministic (and linear) parallel algorithms (as particular cases of the general up-to-up bidirectional parsing algorithm) for solving the membership problem, can be derived for some "combinations" of subclasses of context free languages. The deterministic up-to-up bidirectional parser has the same device as the general model. The only difference is the uniqueness of choosing the production `r` from the set of the given productions of the input grammar (i.e. no backtrack step is needed). We use the definitions of $LR(k)$ ([6]) grammars.

Definition 4.1 *Let G be a context free grammar and k be a natural number. We say that G is $RL(k)$ if $\widetilde{G}$ is a $LR(k)$ grammar. A language L is $RL(k)$ if there exists a $RL(k)$ grammar which generates L.*

In a similar way to Definition 4.1, we can define the "*mirror*" of classical subclasses of $LR(k)$ grammars useful in compiler techniques. We combine the $LR(k)$ and $RL(k)$ styles for obtaining the deterministic up-to-up bidirectional parsing for context free languages.

Definition 4.2 *Let G be a context free grammar and k_1, $k_2 \in \mathbf{N}$. We say that G is a $LR(k_1)-RL(k_2)$ grammar iff G is both a $LR(k_1)$ and $RL(k_2)$ grammar. A language L is called $LR(k_1) - RL(k_2)$ if there exists G a $LR(k_1) - RL(k_2)$ grammar for which $L = L(G)$.*

We can remark that G is a $LR(k_1) - RL(k_2)$ grammar iff G is a $RL(k_2) - LR(k_1)$ grammar. Of course, because the class of $LR(k)$ languages, for $k \geq 1$, equals to the class of $LR(1)$ languages ([6]), then the above definition has practical interest for k_1, $k_2 \in \{0, 1\}$.

Particularly, the deterministic up-to-up bidirectional parsing is similar to the general parallel approach, the only difference being the lack of backtrack steps. The procedures `action1(P1)` and `action2(P2)` are related to classical sequential syntactical analysis algorithms for $LR(1)$ (or the subclasses $LR(0)$, $SLR(1)$ and $LALR(1)$) and $RL(1)$. We also get a linear running time for the procedures `action1(P1)` and `action2(P2)` instead of an exponential sequential running time (backtrack steps are no more required).

Obviously, the procedure `action3(P1,P2)` has no backtrack steps for the subclasses of grammars given in Definition 4.2. Consequently, the procedure `action3(P1,P2)` is deterministic and has a linear running time.

The correctness of the deterministic parallel algorithms is ensured by the correctness of the general parallel algorithm and the correctness of each of the sequential syntactic analysers for the mentioned subclasses of context free grammars (such as $LR(1)$, $RL(1)$).

Theorem 4.1 *([3]) - the complexity of the deterministic parallel algorithms)*

Let us denote with $T_1(n)$, $T_2(n)$ and $T_3(n)$ the running time of the sequential procedures `action1(P1)`, `action2(P2)` *and* `action3(P1,P2)`, *where n is the length of the input word. Supposing that the routing time is zero, the parallel running time $t(n)$ satisfies the relations: $\frac{\min\{T_1(n),T_2(n)\}}{2} + T_3(n) \leq t(n) \leq \max\{T_1(n), T_2(n)\} + T_3(n)$, and $t(n) \in \mathcal{O}(n)$.*

5 A Natural Language Example

We present a modified context free grammar similar to the one presented in [12] for describing syntactic dependencies for English language.

The term 'grammatical category' covers the parts of speech and types of phrases, such as noun phrase and prepositional phrase. For convenience, we will abbreviate them, so that 'NOUN' becomes 'N', 'NOUN PHRASE' becomes 'NP', 'DETERMINER' becomes 'D', etc. Let us consider the following context free grammar:

$G = (\{S, NP, VP, D, A, N, PP, V, P\}, V_T, S, Prod)$, where $V_T \subseteq \Sigma^*$, Σ being the English alphabet and $Prod$ being the following set of productions:

1. $S \to NP\ VP$	7. $NP \to N$	13. $D \to the$	21. $N \to hunter$
2. $NP \to D\ A\ A\ N\ PP$	8. $VP \to V\ NP\ PP$	14. $D \to some$	22. $V \to attack$
3. $NP \to A\ A\ N\ PP$	9. $VP \to V\ NP$	15. $A \to big$	23. $V \to ate$
4. $NP \to D\ A\ N\ PP$	10. $VP \to V\ PP$	16. $A \to brown$	24. $V \to watched$
5. $NP \to A\ N\ PP$	11. $VP \to V$	17. $A \to old$	25. $P \to for$
6. $NP \to D\ N$	12. $PP \to P\ NP$	18. $N \to birds$	26. $P \to beside$
		19. $N \to fleas$	27. $P \to with$
		20. $N \to dog$	

In [12], the productions 1,..., 12 are called **rules**, and 13, ..., 27 are called **lexicon**.

Using a JAVA implementation for our bidirectional parsing algorithm, we determine the viable prefix automaton for grammar G which has 88 states corresponding to $LR(1)$ items with no conflicts (reduce-reduce, reduce-shift), so G is a $LR(1)$ grammar. Moreover, we construct the viable prefix automaton for grammar $\widetilde{G}$ which has 70 states corresponding to $RL(1)$ items with no conflicts (reduce-reduce, reduce-shift), so $\widetilde{G}$ is a $RL(1)$ grammar.

According to Definition 4.2, it follows that grammar G is a $LR(1) - RL(1)$ grammar. We consider now the input word:

$w = the\ big\ brown\ dog\ with\ fleas\ watched\ the\ birds\ beside\ the\ hunter$

We shall simulate a possible execution of the bidirectional parsing algorithm for this example (processors `P1` and `P2` work synchronously). We have $n = |\mathtt{w}|$ and `i1` and `i2` two pointers between 1 and 12. We shall point out the iterations of the procedures `action(P1)` and `action(P2)`. The execution of these procedures will finish when `i1` > `i2`.

Processor P1	Processor P2
Action: Initial	Action: Initial
Stack1 = [ ]	Stack2 = [ ]
i1 = 1	i2 = 12
Output_tape1 = [ ]	Output_tape2 = [ ]
Action: Shift	Action: Shift
Stack1 = [the]	Stack2 = [hunter]
i1 = 2	i2 = 11
Output_tape1 = [ ]	Output_tape2 = [ ]
Action: Reduce	Action: Reduce
Stack1 = $[D_{1,1}]$	Stack2 = $[N_{1,1}]$
i1 = 2	i2 = 11
Output_tape1 = [13]	Output_tape2 = [21]
Action: Shift	Action: Shift
Stack1 = $[D_{1,1}, big]$	Stack2 = $[the, N_{1,1}]$
i1 = 3	i2 = 10
Output_tape1 = [13]	Output_tape2 = [21]
Action: Reduce	Action: Reduce
Stack1 = $[D_{1,1}, A_{2,2}]$	Stack2 = $[D_{2,2}, N_{1,1}]$
i1 = 3	i2 = 10
Output_tape1 = [15, 13]	Output_tape2 = [21, 13]
Action: Shift	Action: Reduce
Stack1 = $[D_{1,1}, A_{2,2}, brown]$	Stack2 = $[NP_{1,3}]$
i1 = 4	i2 = 10
Output_tape1 = [15, 13]	Output_tape2 = [6, 21, 13]
Action: Reduce	Action: Shift
Stack1 = $[D_{1,1}, A_{2,2}, A_{3,3}]$	Stack2 = $[beside, NP_{1,3}]$
i1 = 4	i2 = 9
Output_tape1 = [16, 15, 13]	Output_tape2 = [6, 21, 13]
Action: Shift	Action: Reduce
Stack1 = $[D_{1,1}, A_{2,2}, A_{3,3}, dog]$	Stack2 = $[P_{4,4}, NP_{1,3}]$
i1 = 5	i2 = 9
Output_tape1 = [16, 15, 13]	Output_tape2 = [6, 21, 13, 26]
Action: Reduce	Action: Reduce
Stack1 = $[D_{1,1}, A_{2,2}, A_{3,3}, N_{4,4}]$	Stack2 = $[PP_{1,5}]$
i1 = 5	i2 = 9
Output_tape1 = [20, 16, 15, 13]	Output_tape2 = [12, 6, 21, 13, 26]
Action: Shift	Action: Shift
Stack1 = $[D_{1,1}, A_{2,2}, A_{3,3}, N_{4,4}, with]$	Stack2 = $[birds, PP_{1,5}]$
i1 = 6	i2 = 8
Output_tape1 = [20, 16, 15, 13]	Output_tape2 = [12, 6, 21, 13, 26]

Procesor P1	Procesor P2
Action: Reduce	Action: Reduce
Stack1 = $[D_{1,1}, A_{2,2}, A_{3,3}, N_{4,4}, P_{5,5}]$	Stack2 = $[N_{6,6}, PP_{1,5}]$
i1 = 6	i2 = 8
Output_tape1 = [27, 20, 16, 15, 13]	Output_tape2 = [12, 6, 21, 13, 26, 18]
Action: Shift	Action: Shift
Stack1 = $[D_{1,1}, A_{2,2}, A_{3,3}, N_{4,4}, P_{5,5}, fleas]$	Stack2 = $[the, N_{6,6}, PP_{1,5}]$
i1 = 7	i2 = 7
Output_tape1 = [27, 20, 16, 15, 13]	Output_tape2 = [12, 6, 21, 13, 26, 18]
Action: Reduce	Action: Reduce
Stack1 = $[D_{1,1}, A_{2,2}, A_{3,3}, N_{4,4}, P_{5,5}, N_{6,6}]$	Stack2 = $[D_{7,7}, N_{6,6}, PP_{1,5}]$
i1 = 7	i2 = 7
Output_tape1 = [19, 27, 20, 16, 15, 13]	Output_tape2 = [12, 6, 21, 13, 26, 18, 13]
Action: Reduce	Action: Reduce
Stack1 = $[D_{1,1}, A_{2,2}, A_{3,3}, N_{4,4}, P_{5,5}, NP_{6,7}]$	Stack2 = $[NP_{6,8}, PP_{1,5}]$
i1 = 7	Stack2 = $[NP_{6,8}, PP_{1,5}]$
Output_tape1 = [7, 19, 27, 20, 16, 15, 13]	Output_tape2 = [12, 6, 21, 13, 26, 6, 18, 13]
Action: Reduce	Action: Shift
Stack1 = $[D_{1,1}, A_{2,2}, A_{3,3}, N_{4,4}, PP_{5,8}]$	Stack2 = $[watched, NP_{6,8}, PP_{1,5}]$
i1 = 7	i2 = 6
Output_tape1 = [12, 7, 19, 27, 20, 16, 15, 13]	Output_tape2 = [12, 6, 21, 13, 26, 6, 18, 13]
Action: Reduce	Action: Reduce
Stack1 = $[NP_{1,9}]$	Stack2 = $[V_{9,9}, NP_{6,8}, PP_{1,5}]$
i1 = 7	i2 = 6
Output_tape1 = [2, 12, 7, 19, 27, 20, 16, 15, 13]	Output_tape2 = [12, 6, 21, 13, 26, 6, 18, 13, 24]

Now, the processors P1 and P2 meet in the "middle" of the input word, and only processor P1 will be strongly active. Processor P2 only sends the data (i.e. Stack2, Output_tape2) to the internal memory of P1. In fact, we shall illustrate the execution steps made by the procedure action(P1,P2).

Action: Shift-Nonterminal	Action: Shift-Nonterminal
Stack1 = $[NP_{1,9}, V]$	Stack1 = $[NP_{1,9}, V, NP, PP]$
Stack2 = $[NP_{6,8}, PP_{1,5}]$	Stack2 = [ ]
Output_tape1 = [2, 12, 7, 19, 27, 20, 16, 15, 13]	Output_tape1 = [2, 12, 7, 19, 27, 20, 16, 15, 13]
Output_tape2 = [12, 6, 21, 13, 26, 6, 18, 13]	Output_tape2 = [ ]
Output_tape = [24]	Output_tape = [12, 6, 21, 13, 26, 6, 18, 13, 24]

Action: Shift-Nonterminal	Action: Reduce
Stack1 = $[NP_{1,9}, V, NP]$	Stack1 = $[NP_{1,9}, VP]$
Stack2 = $[PP_{1,5}]$	Stack2 = [ ]
Output_tape1 = [2, 12, 7, 19, 27, 20, 16, 15, 13]	Output_tape1 = [2, 12, 7, 19, 27, 20, 16, 15, 13]
Output_tape2 = [12, 6, 21, 13, 26]	Output_tape2 = [ ]
Output_tape = [6, 18, 13, 24]	Output_tape = [8, 12, 6, 21, 13, 26, 6, 18, 13, 24]

The final transition is:

Action: Reduce
Stack1 = $[S]$
Stack2 = [ ]
Output_tape1 = [ ]
Output_tape2 = [ ]
Output_tape = [1, 8, 12, 6, 21, 13, 26, 6, 18, 13, 24, 2, 12, 7, 19, 27, 20, 16, 15, 13]

In conclusion, the word w is accepted by the bidirectional parser and it has the right syntactic analysis $\pi_{rm} = [1, 8, 12, 6, 21, 13, 26, 6, 18, 13, 24, 2, 12, 7, 19, 27, 20, 16, 15, 13]$.

We can extend our example according to agreement restrictions between words and phrases. For example, the *NP* "a dogs" is not correct English because the article "a" indicates a single object while the noun "dogs" indicates a plural object. There are many other forms of agreement (subject - verb, gender, etc). To use such agreement restrictions, the context free grammar is extended to allow constituents to have **features** ([2]). For example, we might define a feature $NUMBER$ that may take a value of either "s" (for singular) or "p" (for plural), and we then might write an augmented context free grammar rule such as:

$NP \rightarrow D\ N$ only when $NUMBER(D) = NUMBER(N)$.

There is an interesting issue of whether an augmented context free grammar can describe languages that cannot be described by a simple context free grammar. The answer depends on the constraints on what can be a feature value. If the set of feature values is finite, then it would always be possible to create new constituent categories for every combination of features. Thus it is expressively equivalent to a context free grammar. If the set of feature values is unconstrained, however, then such grammars have arbitrary computational power. In practice, even when the set of values is not explicitly restricted, this power is not used, and the standard parsing algorithms can be used on grammars that includes features ([2]).

6 Conclusions

We have seen that our bidirectional parser model is in fact a two-stack machine ([5]) and therefore our model can simulate a Turing machine. In this paper the bidirectional parsing uses $LR(k)$ style (and its "*mirror*" $RL(k)$ style, of course).

Open problems and future work: to get a more precise estimation of the running time of the deterministic parallel algorithm; to find further closure and inclusion properties of $LR-RL$ grammars and languages; to extend our bidirectional approach to non context free languages - with applications to computational linguistic and natural language processing.

We want to thank to the unknown referees for their useful remarks which improve the paper.

References

[1] Akl, S.: *Parallel Computation. Models and Methods.* Prentice Hall (1997)

[2] Allen, P. *Natural Language Understanding.* Second Edition, The Benjamin/Cummings Publishing Company, Inc. (1995)

[3] Andrei, Ştefan: Bidirectional Parsing. *Ph-D Thesis*, Fachbereich Informatik, Universität Hamburg, URL: `http://www.sub.uni-hamburg.de/disse/134/inhalt.html` (2000)

[4] Andrei, Ştefan, Kudlek, M.: Bidirectional Parsing for Linear Languages. *Developments in Language Theory*, Fourth International Conference, July 6-9, 1999, Aachen, Germany, *Preproceedings*, W. Thomas Ed. (1999) 331-344

[5] Hopcroft, J.E., Ullman, J.D.: *Introduction to Automata Theory, Languages and Computation.* Addison - Wesley Publishing Company (1979)

[6] Knuth, D.E.: On the translation of languages from left to right. *Information Control.* 8 (1965) 607-639

[7] Kulkarni, S.R., Shankar, P.: Linear time parsers for classes of non context free languages. *Theoretical Computer Science.* 165 (1996) 355-390

[8] Loka, R.R.: A note on parallel parsing. *SIGPLAN Notices.* 19 (1) (1984) 57-59

[9] Tseytlin, G.E., Yushchenko, E.L.: Several aspects of theory of parametric models of languages and parallel syntactic analysis. In: *Methods of Algorithmic Language Implementation.* A.Ershov, C.H.A.Koster (Eds.), LNCS 47, Springer Verlag, Berlin, Germany (1977) 231-245

[10] Salomaa, A.: *Formal Languages.* Academic Press. New York (1973)

[11] Satta, G., Stock, O.: Bidirectional context-free grammar parsing for natural language processing. *Artificial Intelligence.* 69 (1994) 87-103

[12] Sag, I.A., Wasow, T.: *Syntactic Theory: A Formal Introduction.* CSLI Publications. Leland Stanford Junior University (1997)