B.6.5.2.2.1. Set $a'_\imath := b$, and $a'_{\bar\imath} := a$.

B.6.5.2.2.2. Set $P := P \cup \{((I_0^{(\mathrm{shift}X)\boldsymbol{d}}, Y), a'_0, a'_1)\}$.

B.7. Return P.

We will now settle Theorem 16.1 by showing the correctness of Algorithm B.

Lemma 16.2 *Algorithm B always terminate.*

Proof: All the iterations are easily bounded, so the only critical step is the recursion at step B.6.4.2.3. By definition in step B.6.4.1 the action sequence $\boldsymbol{c}$ contains no shifts, and this property is inherited by $\mathbf{d}$ in step B.6.4.2. Hence, by definition, in step B.6.4.2.1, the parse tree v has an empty frontier, implying that the length of the frontier of t'_0 and t'_1 is one less than that of t_0 and t_1. Thus, the total depth of the recursion caused by step B.6.4.2.3 cannot be bigger than the length of the frontier of t_0 and t_1. We may therefore conclude that Algorithm B always terminates. ■

Lemma 16.3 *Suppose the parse tree pair (t'_0, t'_1) follows by substitution from the parse tree pair (t_0, t_1). Then the projection of (t'_0, t'_1) is included in the set of projections returned by Algorithm B.*

Proof: The result is trivial if $t_0 = t_1$, for then also $t'_0 = t'_1$, so the projection of (t'_0, t'_1) is $\perp$, which is returned in step B.1 Thus we may assume that $t_0 \neq t_1$. Notice that this does not in general imply $t'_0 \neq t'_1$. However, if $t'_0 \neq t'_1$ and $\boldsymbol{a}'^-$ is the longest common prefix of the action sequences of t'_0 and t'_1, then $\mathrm{state}(t_0, |\boldsymbol{a}'^-|) = \mathrm{state}(t_1, |\boldsymbol{a}'^-|)$ and $\mathrm{input}(t_0, |\boldsymbol{a}'^-|) = \mathrm{input}(t_1, |\boldsymbol{a}'^-|)$, and then

$$((\mathrm{state}(t_0, |\boldsymbol{a}'^-|), \mathrm{input}(t_0, |\boldsymbol{a}'^-|)), \mathrm{action}(t_0, |\boldsymbol{a}'^-|), \mathrm{action}(t_1, |\boldsymbol{a}'^-|))$$

is the projection of (t'_0, t'_1).

Let $X, X_1, \ldots, X_p, \boldsymbol{a}^-, a_0, a_1, \boldsymbol{a}_0^+, \boldsymbol{a}_1^+, k$ be as defined in steps B.3–B.5.

by the definition of substitution (page 10) there are parse trees $u, u_1, \ldots, u_p$ and a number l such that for $\imath = 0, 1$, the parse tree $t'_\imath$ results from $t_\imath$ by first, for $i = 1, \ldots, p$, rooting u_i in the ith non-ε leaf of $t_\imath$, and subsequently rooting the obtained parse tree in the lth non-ε leaf of u. Then X is the label of this lth leaf of u, and then X_i is the root symbol of u_i. Notice, that $u_1, \ldots, u_{k-1}$ cannot affect the projection of (t'_0, t'_1), so without loss of generality, we may assume, for $i = 0, \ldots, k-1$, that $u_i = \mathrm{p}\underline{\mathrm{t}}(X_i)$.

Let the action sequences $\boldsymbol{b}^-, \boldsymbol{b}^+$ be defined such that $\boldsymbol{b}^-(\mathrm{shift}X)\boldsymbol{b}^+$ is the action sequence of u with the shift$X$ being the lth shift. Similarly, let $\beta^- X \beta^+$ be the frontier of u. For $i = k, \ldots p$, denote by $\boldsymbol{b}_i$ the action sequence of u_i without the terminating accept. Moreover, denote by β_i the frontier of u_i. Thus, for $\imath = 0, 1$, if $\boldsymbol{a}_{\imath,0}(\mathrm{shift}X_1)\boldsymbol{a}_{\imath,1}(\mathrm{shift}X_2)\cdots(\mathrm{shift}X_p)\boldsymbol{a}_{\imath,p}$ (accept) is the action sequence of $t_\imath$, then

$$\boldsymbol{b}^- \boldsymbol{a}_{\imath,0}(\mathrm{shift}X_1)\boldsymbol{a}_{\imath,1}(\mathrm{shift}X_2)\boldsymbol{a}_{\imath,2}\cdots(\mathrm{shift}X_{k-1})\boldsymbol{a}_{\imath,k-1}\boldsymbol{b}_k\boldsymbol{a}_{\imath,k}\cdots\boldsymbol{b}_p\boldsymbol{a}_{\imath,p}\boldsymbol{b}^+$$

is the action sequence of $t'_\imath$; and, similarly, $\beta^- X_1 \cdots X_{k-1}\beta_k \cdots \beta_p\beta^+$ is the frontier of $t_\imath$.

Observe that $I_0 := \text{state}(u, |\boldsymbol{b}^-|)$ must be one of the instantiations of I_0 in step B.6, for u forces shift$X = \text{action}(u, |\boldsymbol{b}^-|)$ to be possible in state$(u, |\boldsymbol{b}^-|)$. In the remainder of the proof, we focus on the iteration with $I_0 = \text{state}(u, |\boldsymbol{b}^-|)$, and show that the projection of (t'_0, t'_1) is found within this iteration.

Let I_1 and L be as defined in step B.6.1 and B.6.2. As a consequence of our choice of I_0 relative to u, we have that u forces the first symbol Z of $\beta^+\$$ to be in L. Following the algorithm, the rest of the proof divides into cases depending on a_0 and a_1.

Both a_0 and a_1 are reductions (step B.6.3): Set $\boldsymbol{a}'^- = \boldsymbol{b}^-\boldsymbol{a}^-$. Clearly $\boldsymbol{a}'^-$ is the longest common prefix of the action sequences of t'_0 and t'_1. Then $\text{state}(t'_0, |\boldsymbol{a}'^-|) = I_0^{\boldsymbol{a}^-} = I_1$, and $\text{action}(t'_\imath, |\boldsymbol{a}'^-|) = a_\imath$ for $\imath = 0, 1$. Moreover, input$(t_0, |\boldsymbol{a}'^-|)$ is the first symbol of $\beta_k \cdots \beta_p\beta^+\$$ and hence of $\beta_k \cdots \beta_p Z$, but $Z \in L$, so input$(t_0, |\boldsymbol{a}'^-|) \in \mathit{FirstL}(X_k \cdots X_p, L)$. Thus the projection of (t'_0, t'_1) is found in step B.6.3.1.

$a_\imath = \text{shift}X_k$, where either $\imath = 0$, or $\imath = 1$ (step B.6.4): Then $a_{\bar\imath}$ must be a reduction. Let $\boldsymbol{c}$ be the prefix of $a_{\bar\imath}\mathbf{a}^+_{\bar\imath}$ defined in step B.6.4.1. Thus, shiftX_k is the action following $\boldsymbol{c}$ in $a_{\bar\imath}\boldsymbol{a}^+_{\bar\imath}$.

$\boldsymbol{b}_k$ is a prefix of $\boldsymbol{c}\,\boldsymbol{b}_k$: Then $\boldsymbol{b}_k$ is a prefix of $\boldsymbol{c}^\infty$, so there is a unique $n \geq 0$ and non-empty prefix $\boldsymbol{d}$ of $\boldsymbol{c}$ such that $\boldsymbol{b}_k = \boldsymbol{c}^n\,\boldsymbol{d}$. We claim that $n = 0$. Recall, that $\boldsymbol{b}_k$ (accept) is the action sequence of the parse tree u_k. This implies that the overall effect of $\boldsymbol{b}_k$ on the stack is to push the symbol X_k. At the same time, it implies that both $\boldsymbol{c}$ and $\boldsymbol{d}$ are prefix of the action sequence of some parse tree, but any such prefix pushes at least one symbol onto the stack. Thus we cannot have $n > 0$, so $\boldsymbol{b}_k = \boldsymbol{d}$.

Now $\boldsymbol{d}$ is a non-empty prefix of $\boldsymbol{c}$ and since $\boldsymbol{d} = \boldsymbol{b}_k$, in state $\mathit{INIT}(X_k)^{\boldsymbol{d}}$, it is possible to accept, so $\boldsymbol{d}$ has the same value as in one of the iterations in step B.6.4.2. With this instantiation of $\boldsymbol{d}$, in step B.6.4.2.1, we get $v = u_k$.

Now, with $t^v_\imath$ as defined in step B.6.4.2.2, it is clear that (t'_0, t'_1) follows from (t^v_0, t^v_1) by substitution. Recall from the proof of Lemma 16.2 that the frontier of $t^v_\imath$ is one shorter than that of $t_\imath$. Thus, inductively we may conclude that the projection of (t'_0, t'_1) is found in step B.6.4.2.3.

$\boldsymbol{b}_k$ is not a prefix of $\boldsymbol{c}\,\boldsymbol{b}_k$: Let the action sequence $\boldsymbol{x}$ and the actions a and b, where $a \neq b$, be defined such that $\boldsymbol{x}a$ is a prefix of $\boldsymbol{c}\,\boldsymbol{b}_k$ and $\boldsymbol{x}\,b$ is a prefix of $\boldsymbol{b}_k$. Set $\boldsymbol{a}'^- = \boldsymbol{b}^-\boldsymbol{a}^-\boldsymbol{x}$. Thus $\boldsymbol{a}'^-$ is the longest common prefix of t'_0 and t'_1, and then $\text{action}(t'_\imath, |\boldsymbol{a}'^-|) = b$ and $\text{action}(t'_{\bar\imath}, |\boldsymbol{a}'^-|) = a$.

Clearly $\boldsymbol{x}$ is a prefix of $\boldsymbol{c}^\infty$, so there is a unique $n \geq 0$ and proper prefix $\boldsymbol{d}$ of $\boldsymbol{c}$ such that $\boldsymbol{x} = \boldsymbol{c}^n\,\boldsymbol{d}$. Set

$$(I, J) = ((\overbrace{\cdots(I_1{}^{\boldsymbol{c}})\cdots)^{\boldsymbol{c}}}^{n}, (\overbrace{\cdots(\mathit{INIT}(X_k)^{\boldsymbol{c}})\cdots)^{\boldsymbol{c}}}^{n})$$

This is well-defined since $\boldsymbol{c}$ (if $n > 0$) and $\boldsymbol{d}$ are prefix of the action sequence $\boldsymbol{b}_k$ of u_k. With the set S as defined in step B.6.4.3, we have $(I, J) \in S$, so all of $I, J, \boldsymbol{d}$ have values as in some iteration of step B.6.4.4. Notice that also a has the value that it gets assigned in this iteration in step B.6.4.4.1.

The frontier β_k of u_k is non-empty: Denote by Y the first symbol in β_k. Thus $Y = \text{input}(t'_0, |\boldsymbol{a}'^-|)$. Set $b = \text{action}(u_k, |\boldsymbol{x}|)$. Clearly $b = \text{action}(u_k, |\boldsymbol{x}|)$ is possible in state $J^{\boldsymbol{d}} = \mathit{INIT}(X_k)^{\boldsymbol{x}} = \text{state}(u_k, |\boldsymbol{x}|)$ on input $Y = \text{input}(u_k, |\boldsymbol{x}|)$. Thus b and Y are instantiated as in some iteration of step B.6.4.4.2, and hence the projection of (t'_0, t'_1) is found in steps B.6.4.4.2.1–B.6.4.4.2.2.

The frontier β_k of u_k is empty: Denote by Y the first symbol of $\beta_{k+1} \cdots \beta_p \beta^+ \$$. Thus $Y = \text{input}(t'_0, |\boldsymbol{a}'^-|)$. Moreover, we then have $Y \in \mathit{FirstL}(X_{k+1} \cdots X_p, L)$ since L contains the first symbol Z in $\beta^+\$$. Set $b = \text{action}(u_k, |\boldsymbol{x}|)$. Clearly b is possible in state $J^{\boldsymbol{d}} = \text{state}(u_k, |\boldsymbol{x}|)$ on input $\$ = \text{input}(u_k, |\boldsymbol{x}|)$. Thus b and Y are instantiated as in some iteration of step B.6.4.4.3, and hence the projection of (t'_0, t'_1) is found in steps B.6.4.4.3.1–B.6.4.4.3.2.

$a_\imath = \text{accept}$, where either $\imath = 0$, or $\imath = 1$ (step B.6.5): Then $a_{\bar\imath}$ must be a reduction, and moreover, we must have $I_0^{\boldsymbol{a}^-} = I_0^{\text{shift}X}$. Let $\boldsymbol{c}$ be as defined in step B.6.5.1. Thus $\boldsymbol{c}(\text{accept}) = a_{\bar\imath}\boldsymbol{a}^+_{\bar\imath}$. Let the action sequence $\boldsymbol{x}$ and the actions a and b, where $a \neq b$, be defined such that $\boldsymbol{x}a$ is a prefix of $\boldsymbol{c}\,\boldsymbol{b}^+$ and $\boldsymbol{x}\,b$ is a prefix of $\boldsymbol{b}^+$. Set $\boldsymbol{a}'^- = \boldsymbol{b}^-\boldsymbol{a}^-\boldsymbol{x}$. Thus $\boldsymbol{a}'^-$ is the longest common prefix of t'_0 and t'_1, and then $\text{action}(t'_\imath, |\boldsymbol{a}'^-|) = b$ and $\text{action}(t'_{\bar\imath}, |\boldsymbol{a}'^-|) = a$. Clearly $\boldsymbol{x}$ is a prefix of $\boldsymbol{c}^\infty$, so there is a unique $n \geq 0$ and proper prefix $\boldsymbol{d}$ of $\boldsymbol{c}$ such that $\boldsymbol{x} = \boldsymbol{c}^n\,\boldsymbol{d}$.

Notice that $(\text{shift}X)\boldsymbol{c}(\text{accept})$ constructs a parse tree generating $X \to^* X$, and hence so does $(\text{shift}X)\boldsymbol{c}^n(\text{accept})$. Thus we have

$$\text{state}(t_0, |\boldsymbol{a}'^-|) = I_0^{\boldsymbol{a}^-\boldsymbol{x}} = I_0^{(\text{shift}X)\boldsymbol{c}^n\boldsymbol{d}} = I_0^{(\text{shift}X)\boldsymbol{d}}.$$

Let Y be the first symbol in $\beta^+\$$. Then $\text{input}(t_0, |\boldsymbol{a}'^-|) = Y$, and, moreover, $Y \in L$. The parse tree u forces b to be possible in state $I_0^{(\text{shift}X)\boldsymbol{d}}$ on input Y, since these are the action, the state, and the input after the initial action sequence $\boldsymbol{b}^-(\text{shift}X)\boldsymbol{x}$ has been carried out. Thus b and Y are instantiated as in some iteration of step B.6.5.2.2, so the projection of (t'_0, t'_1) is found in steps B.6.5.2.2.1–B.6.5.2.2.2.

■

Lemma 16.4 *For all p in the set P returned by Algorithm B, there is parse tree pair (t'_0, t'_1) that follows from (t_0, t_1) by substitution and which projects to p.*

Proof: It is easy, but tideous, to verify that the proof of Lemma 16.3 can be reversed, so that for any projection p, we find parse trees $u, u_k, \cdots, u_p$ giving us the the desired parse tree pair (t'_0, t'_1). All we need is the property of universal CLR(1) parsers that any partial parsing can be extended to a successful parsing generating some parse tree. ■

Proof of Theorem 16.1: By Lemmas 16.2, 16.3, and 16.4, Algorithm B always terminates finding the set of projections of all the parse tree pairs following by substitution from the input parse tree pair. Thus with Algorithm A and Algorithm B together, given a universal CLR(1) parser $\mathcal{U}$ and a finite set $\mathcal{E}$ of parse tree pairs, we can decide if $\mathcal{E}^{\prec}$ spans an ordering of $\mathcal{U}$, and if so, construct this ordering. ■

Concerning our example (4), for the parse tree pairs in $\mathcal{E}_{(4)}$ we get

$$P\left(E\ ,\ \begin{array}{c} E \\ | \\ F \\ | \\ E \end{array} \right) = \left\{ ((2,\$), \text{accept}, \text{reduce } F{\to}E) \right\},$$

$$P\left(F\ ,\ \begin{array}{c} F \\ | \\ E \\ | \\ F \end{array} \right) = \left\{ \begin{array}{l} ((3,E), \text{shift}E, \text{reduce } E{\to}F) \\ ((3,F), \text{shift}F, \text{reduce } E{\to}F) \\ ((5,E), \text{shift}E, \text{reduce } E{\to}F) \\ ((5,E), \text{reduce } F{\to}FF, \text{reduce } E{\to}F) \\ ((5,F), \text{shift}F, \text{reduce } E{\to}F) \\ ((5,F), \text{reduce } F{\to}FF, \text{reduce } E{\to}F) \\ ((5,\$), \text{reduce } F{\to}FF, \text{reduce } E{\to}F) \\ ((7,E), \text{shift}E, \text{reduce } E{\to}F) \\ ((7,F), \text{shift}F, \text{reduce } E{\to}F) \\ ((7,\$), \text{accept}, \text{reduce } E{\to}F) \end{array} \right\} \text{ and}$$

$$P\left(\begin{array}{l} F \\ |\backslash \\ FF \\ \ \ |\backslash \\ \ \ FF \end{array}\ ,\ \begin{array}{l} \ \ F \\ \ \ |\ \backslash \\ \ \ F\ F \\ \ \ |\backslash \\ \ \ FF \end{array} \right) = \left\{ \begin{array}{l} ((5,E), \text{shift}E, \text{reduce } F{\to}FF) \\ ((5,F), \text{shift}F, \text{reduce } F{\to}FF) \end{array} \right\}.$$

17 Tracing the minimal non-canonical parse trees

This section addresses problem 3, page 33. As mention in Section 12, we will not solve the problem exactly as stated. However, when we come to Section 20, where the concrete description of the algorithm generalizing the canonical LR(1) technique is given, it will be clear that the results of this section give a sufficient substitution for an exact solution to problem 3.

A set of parse trees is said to be *stable* if it contains all sub-parse trees of parse trees in the set. If the set of canonical parse trees of some parser is stable, we say that the parser itself is *stable*. Notice that if a set $\mathcal{E}$ of parse tree pairs prioritizes a parser $\mathcal{P}$, then the $\mathcal{P}$-canonical parse trees are exactly the parse trees that have no sub-parse tree among the second coordinates in $\mathcal{E}$. Hence any prioritized parser must be stable.

More locally, if for some canonical parse tree, all sub-parse trees are canonical, then we say that this parse tree is *stable canonical*. Thus for a stable parser all canonical parse trees are stable canonical. This simple observation is very useful when looking for prioritized and hence stable parsers.

A parser $\mathcal{P}$ is said to be *basis complete* if all trivial parse trees $\mathrm{p\underline{t}}(X)$ are canonical. Notice that checking for basis completeness is easy. Also notice that if $\mathcal{P}$ is not basis complete, it means that there is a grammar symbols X for which the trivial parse tree $\mathrm{p\underline{t}}(X)$ is not canonical, but this implies that there cannot be any stable canonical parse tree for $X \rightarrow^* X$. Thus, if a parser $\mathcal{P}$ is not basis complete, it cannot be prioritized at all.

The aim for this section is to show the following result which is our substitution for an exact solution to problem 3:

Theorem 17.1 *Given a basis complete determinization of the universal CLR(1) parser, we can represent a set M of parse trees satisfying the following conditions:*

- *M contains all minimal non-canonical parse trees*
- *If the determinization is stable then M is exactly the set of minimal non-canonical parse trees.*
- *It is decidable whether M is infinite.*

We are going to give a concrete construction for such a set M, given any basis complete determinization. Notice, that basis completeness means that all non-canonical parse trees are non-trivial.

Some technical definitions are needed. Given a parse tree, by a *leaf rightside* we understand a sequence of leaves corresponding to the rightside of some production. More precisely, $l_1, \ldots, l_p$ is a leaf rightside if it is the sequence of sons of some internal node and all of $l_1, \cdots, l_p$ are leaves. In particular, any ε-leaf constitutes a leaf rightside. Notice that any non-trivial parse tree has a leaf rightside, and that if we remove a leaf rightside of a parse tree, the result is a parse tree. Repeating this "pruning" process, eventually we will arrive at the trivial parse tree consisting of the root symbol.

A parse tree is said to be *good* if it is non-canonical, but removal of any leaf rightside results in a canonical parse tree. Trivially, all minimal non-canonical parse trees are good. Our first step in constructing M will be to introduce traces. Traces are special kinds of partial parsings. The successful traces construct exactly the good parse trees. Loosely speaking, the actions of a successful trace will correspond to the suffix of the actions sequence of a good parse tree starting from the first leaf rightside. All preceding actions must be shifts, so this suffix determines the good parse tree uniquely.

From the traces we will construct a finite trace graph representing M. The directed walks (possibly looping dipaths) in this graph will correspond to traces. The graph will contain a source vertex and some terminal vertices, and any directed walk from the source to a terminal vertex will correspond to a successful trace for a parse tree in M. Thus M is infinite if and only if is there a loop in a directed walk from a source vertex to some terminal vertex.

We will now go ahead with the formal definition of *traces*. All further motivation is deferred to the following lemmas and their proofs. An initial *trace configuration* is an initial partial configuration. All other trace configurations are triples (s, G, B) where s is a partial configuration, where either G is the symbol '$*$', or G is a set of input symbols, and where B is a set of input symbols. We will refer to G and B as the *goal* and *block*.

Trace actions are grammar productions with a dot somewhere in the rightside, like the items from Section 14. To a trace action $X \rightarrow \alpha \bullet \beta$ corresponds the sequence of normal actions which first shifts β and then reduce $X \rightarrow \alpha\beta$. This sequence of normal actions is denoted $\overline{\mathrm{a}}(X \rightarrow \alpha \bullet \beta)$.

From an initial trace configuration $(\boxed{I})$, by $X \rightarrow \bullet \beta$ we can get to the trace configuration

$$((\boxed{I} X \boxed{GOTO(I, X)}), G, B)$$

if the following conditions are satisfied:

(i) From partial configuration $(\boxed{I})$, it is possible to carry out $\overline{\mathrm{a}}(X \rightarrow \bullet \beta)$ arriving at the partial configuration $(\boxed{I} X \boxed{GOTO(I, X)})$.

(ii) From state I, it is canonical to shift X.

(iii) If, by shifting β from state I, we encounter a non-canonical shift, then $G = *$ and $B = \emptyset$; otherwise, both G and B equal the set of input symbols Y such that in state $GOTO(I, \beta)$ on input Y, it is possible, but not canonical, to reduce $X \rightarrow \beta$.

From the trace configuration (s, G, B), by $X \rightarrow \alpha \bullet \beta$ we can get to the trace configuration (s', G', B') if the following conditions are satisfied:

(iv) If $G = *$, then $\alpha \neq \varepsilon$.

(v) If $G = *$ and $\beta = Y\gamma$, then $Y \notin B$.

(vi) If $G \neq *$ and $\beta = Y\gamma$, then $Y \in G$.

(vii) If $G \neq *$ and $\alpha = \varepsilon$, then $X \notin B$.

(viii) From partial configuration s, it is canonical to shift β thus arriving at a partial configuration $(\boxed{I_0} X_1 \cdots X_n \boxed{I_n})$.

(ix) In state I_n on some input Y it is possible to reduce $X \rightarrow \alpha\beta$. Denote by C is the set of input symbols Y on which in state I it is possible but not canonical to reduce $X \rightarrow \alpha\beta$.

(x) Set $p = |\alpha\beta|$. If $n \geq p$ then $s' = (\boxed{I_0} X_1 \cdots X_{n-p} \boxed{I_{n-p}} X \boxed{GOTO(I_{n-p}, X}$); otherwise, $s' = (\boxed{I} X \boxed{GOTO(I, X})$), where, from state I, canonically we can shift the first $n - p$ symbols of α thus arriving at state I_0. In this case we say that $X \rightarrow \alpha \bullet \beta$ *transcends* c with these first $n - p$ symbols of α.

(xi) If $G \neq *$ and $\alpha = \varepsilon$, then, from partial configuration c, it is canonical to shift X.

(xii) If $G = *$, then $G' = *$.

(xiii) If $G \neq *$ and $\beta = \varepsilon$, then $G' = G \setminus C$.

(xiv) If $G \neq *$ and $\beta = Y\gamma$, then $G' = *$.

(xv) If $\beta = \varepsilon$, then $B' = B \cup C$; otherwise $B' = C$.

A trace configuration (c, G, B) is *accepting* if the following condition is satisfied:

(xvi) In the state of c on input \$ it is canonical to accept, and, moreover, either $\$ \in G$, or $G = *$ and $\$ \notin B$.

A *trace* is a sequence of the form $c_0\,A_0\,c_1 \cdots A_{n-1}\,c_n$ where $c_0, \ldots, c_n$ are trace configuration, where $A_0, \ldots, A_{n-1}$ are trace actions, where c_0 is initial, and where for $i = 0, \ldots, n-1$, we can get from c_i to c_{i+1} by A_i. If, moreover, c_n is accepting, we say the trace is *successful.*

We associate with a successful trace $T = c_0\,A_0 \cdots A_{n-1}\,c_n$ the action sequence

$$(\text{shift } \delta_{n-1} \cdots \delta_0)\,\overline{\text{a}}(A_0) \cdots \overline{\text{a}}(A_{n-1})\,(\text{accept})$$

where δ_i is the sequence of grammar symbols with which a_i transcends c_i if A_i transcends c_i (cf. condition (x)); otherwise $\delta_i = \varepsilon$. From conditions (i), (viii)–(x) it follows that this action sequence is the action sequence of a successful parsing. In fact, in this parsing, the state from which we carry out the first action in $\overline{\text{a}}(A_i)$ is exactly the state of the partial configuration in c_i. In particular, if $c_0 = (\boxed{I_0})$ and $A_{n-1} = X_{n-1} \rightarrow \alpha_{n-1} \bullet \beta_{n-1}$, then $I_0 = \mathit{INIT}(X_{n-1})^{\text{shift } \delta_{n-1} \cdots \delta_0}$. We say that our successful trace *constructs* the parse tree produced by this parsing.

The following statement indicates—with no clue to a finite representation—the relationship between traces and the set M from Theorem 17.1:

Proposition 17.2 *Denote by M the set of parse trees t satisfying that t is constructed by a successful trace T and that there is no other accepting trace T' such that the sequence of trace actions of T' is a proper prefix of the sequence of trace actions of T. Then M contains all minimal non-canonical parse trees.*

If the set of canonical parse trees is stable, then M contains exactly the minimal non-canonical parse trees.

Several lemmas are needed before we can settle the proposition. The first two state that a parse tree is constructed by a successful trace if and only if it is good.

Lemma 17.3 *If a trace is successful, it constructs a good parse tree.*

Proof: Let T be a successful trace, and let t be the parse tree it constructs. We want to show that t is good, i.e. that t is non-canonical, but that if we remove any leaf rightside from t the results is a canonical parse tree.

Write T in the form $(s_0)(X_0 \to \alpha_0 \bullet \beta_0)(s_1, G_1, B_1) \cdots (X_{n-1} \to \alpha_{n-1} \bullet \beta_{n-1})(s_1, G_1, B_1)$. Then the action sequence $\boldsymbol{a}$ of t is of the from $(\text{shift}\,\delta)\,\overline{\text{a}}(X_0 \to \alpha_0 \bullet \beta_0) \cdots \overline{\text{a}}(X_{n-1} \to \alpha_{n-1} \bullet \beta_{n-1})\,(\text{accept})$.

Notice that the trace action $X_i \to \alpha_i \bullet \beta_i$ corresponds to a leaf rightside labeled $\alpha_i \beta_i$ if and only if $\alpha_i = \varepsilon$ (recall that $\alpha_0 = \varepsilon$ by definition). Thus to show that t is good we need to show the following:

(a) $\boldsymbol{a}$ contains a non-canonical action.

(b) For any i with $\alpha_i = \varepsilon$, replacing $\overline{\text{a}}(X_i \to \bullet\, \beta_i)$ with $(\text{shift} X_i)$ in $\boldsymbol{a}$ gives a canonical action sequence.

The proof divides into cases:

shift β_0 contains a non-canonical shift: Clearly (a) is satisfied. By condition (iii) we have $G_1 = *$, and hence by condition (xii) we have $G_i = *$ for all $i > 0$. By condition (iv), this implies that $\alpha_i \neq \varepsilon$ for all $i > 0$. Thus, in order to verify (b), it satisfies to show that $\boldsymbol{a}^0 = (\text{shift}\,\delta)\,(\text{shift}\,X_0)\,\overline{\text{a}}(X_1 \to \alpha_1 \bullet \beta_1) \cdots \overline{\text{a}}(X_{n-1} \to \alpha_{n-1} \bullet \beta_{n-1})\,(\text{accept})$ is canonical.

Clearly $(\text{shift}\,\delta)$ is canonical by condition (x), and $(\text{shift}\,X_0)$ is canonical by condition (ii). Moreover, condition (viii) guarantees that all other shifts in $\boldsymbol{a}^0$ are canonical, and condition (xvi) states that the final accept is canonical. Thus the only remaining concern is if for some $i > 0$, the reduction with $X_i \to \alpha_i \beta_i$ is non-canonical. By conditions (ix) and (xv), this can only be the case if the input symbol for this reduction is contained in B_{i+1}, but this possibility is excluded by conditions (v) and (xvi). Thus we may conclude that (b) is satisfied, and hence that, indeed, t is a good parse tree.

shift β_0 does not contain a non-canonical shift: We will settle (a) by showing that the reduction with $X_0 \to \beta_0$ is non-canonical. By condition (iii), this is the case if the input symbol for this reduction is contained in G_1, but this is guaranteed by conditions (vi) and (xiii). Thus (a) holds.

Our first step in verifying (b) is to show that $\boldsymbol{a}^0 = (\text{shift}\,\delta)\,(\text{shift}\,X_0)\,\overline{\text{a}}(X_1 \to \alpha_1 \bullet \beta_1) \cdots \overline{\text{a}}(X_{n-1} \to \alpha_{n-1} \bullet \beta_{n-1})\,(\text{accept})$ is canonical. As in the last case, it is easy to see that all the shifts and the final accept are canonical. Again the problem is if, for some $i > 0$, the reduction with $X_i \to \alpha_i \beta_i$ is non-canonical. As in condition (viii), denote by C the set of input symbols for which this reduction is non-canonical. Moreover, let Y be the actual input symbol for the reduction. Thus we want to show that $Y \notin C$.

By conditions (xii)–(xv) we have either $G_{i+1} \neq *$ and $G_{i+1} \cap C = \emptyset$, or $G_{i+1} = *$ and $B_{i+1} \supseteq C$. In the former case conditions (vi) and (xiii) ensure that Y is in G_{i+1}, hence not in C, and in the latter case conditions (v) and (xv) ensure that Y is not in B_{i+1}, hence not in C. Thus Y is outside C, as desired, so $\boldsymbol{a}^0$ is indeed canonical.

It remains to be shown that if, for some $k > 0$, we have $\alpha_i = \varepsilon$, then $\boldsymbol{a}^k = (\text{shift}\,\delta)\,\bar{\text{a}}(X_0 \to \alpha_0 \bullet \beta_0) \cdots \bar{\text{a}}(X_{k-1} \to \alpha_{k-1} \bullet \beta_{k-1})\,(\text{shift}\,X_k)\,\bar{\text{a}}(X_{k+1} \to \alpha_{k+1} \bullet \beta_{k+1}) \cdots \bar{\text{a}}(X_{n-1} \to \alpha_{n-1} \bullet \beta_{n-1})\,(\text{accept})$ is canonical. The final accept and all shifts but shiftX_k are shown to be canonical as above. Moreover, shiftX_k is canonical by condition (xi). By condition (iv) we have $G_k \neq *$, and hence by condition (xii), for $i = 1, \ldots, k$, we have $G_i \neq *$. Thus by condition (xiv), for $i = 1, \ldots, k-1$, we have $\beta_i = \varepsilon$. So X_k is the input symbol for all the preceding reductions. Now, by conditions (iii), (ix), and (xv), the set B_{k-1} contains all input symbols for which any of the preceding reductions are non-canonical, but by condition (vii), we do not have X in B_{k-1}. Thus all the reductions preceding shiftX_k are canonical. Concerning the reductions succeeding shiftX_k, they can all be shown to be canonical using the same argument as was applied to the reductions in $\boldsymbol{a}^0$, so we may conclude that $\boldsymbol{a}^k$ is canonical. Thus (b) holds, so, indeed, t is a good parse tree.

■

Lemma 17.4 *If a parse tree is good, it is constructed by a unique successful trace.*

Proof: Let t be a good parse tree with action sequence $\boldsymbol{a}$. Suppose that $\boldsymbol{a}$ is of the form $\boldsymbol{a}_0\,\bar{\text{a}}(X_x \to \bullet \beta_x)\,\boldsymbol{a}_1\,\bar{\text{a}}(X_y \to \bullet \beta_y)\,\boldsymbol{a}_2$. Thus t has two leaf rightsides. We claim that all shifts in $\boldsymbol{a}$ are canonical, and that $\boldsymbol{a}_1$ contains no shifts. Since t is good, $\boldsymbol{a}$ is non-canonical, but for any $i \in \{x, y\}$, if we replace $\bar{\text{a}}(X_x \to \bullet \beta_x)$ with shift X_x the resulting action sequence $\boldsymbol{a}^x$ becomes canonical. Clearly, all non-canonical actions in $\boldsymbol{a}$ have to be contained in $\boldsymbol{a}_0\,\bar{\text{a}}(X_x \to \bullet \beta_x)$, for otherwise $\boldsymbol{a}^x$ would be non-canonical like $\boldsymbol{a}$. However, $\boldsymbol{a}_0\,\bar{\text{a}}(X_x \to \bullet \beta_x)$ cannot contain a non-canonical shift, for then $\boldsymbol{a}^y$ would be non-canonical like $\boldsymbol{a}$. Thus, $\boldsymbol{a}_0\,\bar{\text{a}}(X_x \to \bullet \beta_x)$ contains a non-canonical reduction. Clearly, replacing $\bar{\text{a}}(X_y \to \bullet \beta_y)$ with shiftX_y can only affect the input symbol of this non-canonical reduction—making it canonical—if there are no shifts in $\boldsymbol{a}_1$. Hence, if there are shifts in $\boldsymbol{a}_1$ then $\boldsymbol{a}^y$ is non-canonical like $\boldsymbol{a}$. Thus the claim follows.

As a first consequence of the claim, we have that for any two consequetive reductions in $\boldsymbol{a}$, if β is the sequence of symbols shifted between them, the second reduction is with a production $X \to \alpha\beta$, where possibly α is empty. In other words, β cannot contain any symbols that are not reduced by the second reduction. This implies that $\boldsymbol{a}$ is of the form $(\text{shift}\,\delta)\,\bar{\text{a}}(X_0 \to \alpha_0 \bullet \beta_0) \cdots \bar{\text{a}}(X_{n-1} \to \alpha_{n-1} \bullet \beta_{n-1})\,(\text{accept})$. From the fact that $\boldsymbol{a}$ can be written in this form, it follows that if we ignore all conditions but (i), (viii)–(x), and (vi), and, moreover, substitute "possible" for "canonical" in these conditions, then there is a unique trace

$$T = (s_0)(X_0 \to \alpha_0 \bullet \beta_0)\,(s_1, G_1, B_1) \cdots (X_{n-1} \to \alpha_{n-1} \bullet \beta_{n-1})(s_1, G_1, B_1)$$

constructing t. Here s_0 is the stack consisting of the state $\mathit{INIT}(X)^{\text{shift}\,\delta}$, and for $i > 0$, we have that s_i is a suffix of the stack of the parsing starting in $\mathit{INIT}(X_{n-1})$ after having carried out the action sequence $(\text{shift}\,\delta)\,\bar{\text{a}}(X_0 \to \alpha_0 \bullet \beta_0) \cdots \bar{\text{a}}(X_{i-1} \to \alpha_{i-1} \bullet \beta_{i-1})$. The exact extent of the suffix is determined by conditions (i),(viii), and (x). All the G_is and B_i are determined by conditions (iii), and (xii)–(xv).

In order to complete the proof of the lemma, we need to show that T satisfies all the original conditions. Basically, this just amounts to reversing the argument from the proof of Lemma 17.3, so we will restrict ourselves to an example: proving that condition (iv) is satisfied. Suppose that $\boldsymbol{a}$ contains a non-canonical shift. Then, by our claim, t has only one leaf rightside. Thus $\alpha_i = \varepsilon$ for all $i > 0$, so condition (iv) is always satisfied. Suppose, conversely, that $\boldsymbol{a}$ contains no non-canonical shifts. Then by condition (iii), we have $G_i \neq *$. If (iv) is false, for some $k > 0$, we have $G_k = *$ and $\alpha_k = \varepsilon$. By conditions (xiii) and (xiv) the former implies that for some i, where $0 < i < k$, we have $\beta_i \neq \varepsilon$. Thus there is a shift between the first leaf rightside and the leaf rightside corresponding to $\overline{\mathrm{a}}(X_k \rightarrow \bullet\ \beta_k)$, contradicting our claim. Hence condition (iv) holds. ■

Lemma 17.5 *Assume that the set of canonical parse trees is stable, and let t be a good parse tree. If t is not minimal non-canonical, there is only one of the sons of the root from which a non-trivial parse tree t^r descends, and then t^r is good.*

Proof: Assume that t is not minimal non-canonical; then t has a proper sub-parse tree t^- which is non-canonical.

The parse tree t^- must contain all leaf rightsides of t, for, if t^- did not contain a leaf rightside l, then t^- would be contained in the parse tree t^l which is t without l. Since t is good, t^l is canonical, and by assumption the set of canonical parse trees is stable, so all sub-parse trees of t^l are canonical. This contradict that the sub-parse tree t^- is non-canonical. Thus, as claimed, t^- contains all leaf rightsides of t.

Since t has a proper sub-parse tree t^- containing all leaf rightsides of t, there can only be one non-trivial parse tree t^r descending from a son of the root of t, and then t^- must be a sub-parse tree of t^r.

It remains to be shown that t^r is good. First, since t^r contains t^-, which is non-canonical, by stability, t^r is itself non-canonical.

Now, let l be any leaf rightside of t^r, and denote by t^{rl} the result of removing l from t^r. Similarly, denote by t^l the result of removing the leaf rightside l from t. Since t is good, t^l is canonical, and t^{rl} is contained in t^l, so by stability, t^{rl} is canonical. Thus t^r is good. ■

Proof of Proposition 17.2: As in the proposition, denote by M the set of parse trees t such that t is constructed by a successful trace T_1 and there is no other successful trace T_2 such that the sequence of trace actions of T_2 is a proper prefix of the sequence of trace actions of T_2. We want to show the following:

(a) M contains all minimal non-canonical parse trees.

(b) If the set of canonical parse trees is stable, then M contains exactly the minimal non-canonical parse trees.

Let T_1 and T_2 be successful traces constructing parse trees t_1 and t_2 respectively. It is easy to check if the sequence of trace actions of T_1 is a proper prefix of the sequence of trace actions of T_2 then t_1 is a proper sub-parse tree of t_2. Now, all good parse trees are non-canonical and all minimal non-canonical parse trees are good, so (a) follows directly from Lemma 17.3 and Lemma 17.4 which together say that a parse tree is good if and only if it is constructed by a successful trace.

Assume that the set of canonical parse trees is stable, and let t be a good parse tree which is not minimal non-canonical. Moreover, let t^r be the good sub-parse tree provided by Lemma 17.5. Thus t^r is the only non-trivial parse tree descending from a son of the root of t. By Lemma 17.4, we have unique traces T and T^r such that T constructs t and T^r constructs t^r. The sequence of trace actions of T^r is the proper prefix of the sequence of trace actions of T obtained by removing the last trace action corresponding to the root production, so t is not in M. Thus (b) holds. ∎

The traces have another property which is important in finding a finite representation of our set M.

Lemma 17.6 *Only finitely many different trace configurations occur in traces.*

Proof: We prove the result by bounding the number of states in the stacks of the partial configurations in the trace configurations of an arbitrary, possibly unsuccessful trace

$$(s_0)\,(X_0 \rightarrow \bullet\, \beta_1)\,(s_1, G_1, B_1)\,(X_1 \rightarrow \alpha_1 \bullet \beta_1) \cdots (X_{n-1} \rightarrow \alpha_{n-1} \bullet \beta_{n-1})a_{n-1}\,(s_n, G_n, B_n).$$

Denote by $|s|$ the number of states in the stack of the partial configuration s. Then $|s_0| = 1$, and, for $i = 1, \ldots, n$, we have $|s_i| = \max(1, |s_{i-1}| + 1 - |\alpha_i|)$. Thus the result follows if we can bound the number of indices i for which $\alpha_i = \varepsilon$.

Denote by k the last index for which $\alpha_k = \varepsilon$. Thus, we only need consider indices strictly smaller than k. If $k = 0$ we are done, so assume that $k > 0$. Then by condition (iv), we have $G_k \neq *$, so by condition (xii), for all $i \leq k$, we have $G_i \neq *$. By condition (xiv), this implies for $i < k$, that $\beta_i = \varepsilon$. Thus, if $\alpha_i = \varepsilon$ for some $i < k$, then $a_i = X_i \rightarrow \bullet\, \varepsilon$.

The desired bound on the number of indices $i < k$ for which $\alpha_i = \varepsilon$ is now achieved by showing that there cannot exist indices i, j and a grammar symbol X such that $i < j < k$, and $a_i = a_j = X \rightarrow \bullet\, \varepsilon$; for suppose that i, j, and X satisfy this condition. By conditions (ii) and (xi), from s_i on input X the canonical action is to shift X, and hence it is non-canonical to reduce $X \rightarrow \varepsilon$. Thus, by conditions (ix) and (xv), we have $X \in B_{i+1}$. Recall that $\beta_l = \varepsilon$ for all $l < k$. By condition (xv), this implies $B_i \subseteq B_j$, so, in particular, $X \in B_j$. However, by condition (xi) this contradict $a_j = X \rightarrow \bullet\, \varepsilon$. ∎

Now, we are ready to construct the announced trace graph giving an adequate representation of M. The *trace graph* is the directed graph G generated by the following conditions:

- The vertices are sets of trace configurations, and the arcs are labeled with trace actions.
- There is a source vertex containing all initial trace configurations.
- A vertex is terminal if it includes an accepting trace configuration.

- Let v be a vertex in G that is not terminal and which includes a trace configuration c. Moreover, let a be a trace action and c' a trace configuration such that we can get from c to c' by a. Then there is a vertex v' in G consisting of all trace configurations c_1' such that for some trace configuration c_1 in v, we can get from c_1 to c_1' by a.

Observation 17.7 *Given a sequence $\boldsymbol{A}$ of trace actions; then the following are equivalent:*

- *$\boldsymbol{A}$ is the sequence of trace actions on a path from the source vertex to some terminal vertex.*

- *$\boldsymbol{A}$ is the sequence of trace actions of some successful trace, but no proper prefix of $\boldsymbol{A}$ is the sequence of trace actions of an accepting trace.*

Proof of Theorem 17.1: Let the trace graph G be as defined above; then G is finite, for, by definition, all vertices are subsets of the set of configurations occurring in traces, and by Lemma 17.6 this set is finite.

Denote by M the set of parse trees t such that t corresponds to the sequences of trace actions on a path from the source vertex to some terminal vertex. Proposition 17.2 together with Observation 17.7 give us firstly, that the set M contains all minimal non-canonical parse trees, and secondly, that if the set of canonical parse trees is stable, then M contains only the minimal non-canonical parse trees.

We can test if the set M is infinite simply by testing if there is a loop on a path from the source vertex to some terminal vertex. Thus G gives us the desired representation. ∎

The trace graph for our example (4) is shown below. Notice, that it represents exactly the second coordinates of our prioritizing set $\mathcal{E}_{(4)}$, page 20. (As a kind of optimization, we have excluded all trace configurations with $G = \emptyset$ since these cannot lead to a successful trace.)

$$\left\{\begin{array}{l}(\boxed{1})\\(\boxed{2})\\(\boxed{3})\\(\boxed{4})\\(\boxed{5})\\(\boxed{6})\\(\boxed{7})\end{array}\right\} \xrightarrow{F\to\bullet FF} \left\{\begin{array}{l}(\boxed{1}F\boxed{3},\{E,F\},\{E,F\})\\(\boxed{3}F\boxed{5},\{E,F\},\{E,F\})\\(\boxed{5}F\boxed{5},\{E,F\},\{E,F\})\\(\boxed{6}F\boxed{7},\{E,F\},\{E,F\})\\(\boxed{7}F\boxed{5},\{E,F\},\{E,F\})\end{array}\right\} \xrightarrow{F\to F\bullet F} \left\{\begin{array}{l}(\boxed{1}F\boxed{3},*,\{E,F\})\\(\boxed{3}F\boxed{5},*,\{E,F\})\\(\boxed{5}F\boxed{5},*,\{E,F\})\\(\boxed{6}F\boxed{7},*,\{E,F\})\\ \hline (\boxed{7}F\boxed{5},*,\{E,F\})\end{array}\right\}$$

$$\left\{\begin{array}{l}(\boxed{1})\\(\boxed{2})\\(\boxed{3})\\(\boxed{4})\\(\boxed{5})\\(\boxed{6})\\(\boxed{7})\end{array}\right\} \xrightarrow{F\to\bullet E} \left\{\ (\boxed{1}F\boxed{3},\{\$\},\{\$\})\ \right\} \xrightarrow{E\to F\bullet} \left\{\ \underline{(\boxed{1}E\boxed{2},\{\$\},\{\$,E,F\})}\ \right\}$$

$$\left\{\begin{array}{l}(\boxed{1})\\(\boxed{2})\\(\boxed{3})\\(\boxed{4})\\(\boxed{5})\\(\boxed{6})\\(\boxed{7})\end{array}\right\} \xrightarrow{E\to\bullet F} \left\{\begin{array}{l}(\boxed{1}E\boxed{2},\{E,F\},\{E,F\})\\(\boxed{3}E\boxed{4},\{E,F,\$\},\{E,F,\$\})\\(\boxed{5}E\boxed{4},\{E,F,\$\},\{E,F,\$\})\\(\boxed{6}E\boxed{4},\{E,F,\$\},\{E,F,\$\})\\(\boxed{7}E\boxed{4},\{E,F,\$\},\{E,F,\$\})\end{array}\right\} \xrightarrow{F\to E\bullet} \left\{\begin{array}{l}(\boxed{1}F\boxed{3},\{E,F\},\{E,F\})\\(\boxed{3}F\boxed{5},\{E,F,\$\},\{E,F,\$\})\\(\boxed{5}F\boxed{5},\{E,F,\$\},\{E,F,\$\})\\(\boxed{6}F\boxed{7},\{E,F,\$\},\{E,F,\$\})\\ \hline (\boxed{7}F\boxed{5},\{E,F,\$\},\{E,F,\$\})\end{array}\right\}$$

18 Loop free parsing

This section addresses problem 4, page 33. By an *ε-loop* in a parsing with a universal CLR(1) parser we understand two reductions with the same production $X \to \varepsilon$ without any shifts in between. By an *ε-protected* parsing with a CLR(1) parser $\mathcal{P}$ we understand a parsing that halts unsuccessfully when encountering an ε-loop.

Theorem 18.1 *Let a $\mathcal{P}$ be an $(\mathcal{R}^{\prec})^{\mathcal{U}}$-minimal determinization of a universal CLR(1) $\mathcal{U}$. Consider the ε-protected parsing with $\mathcal{P}$ on input (X, α). The parsing terminates after $O(|\alpha|)$ actions, and if not successfully, (X, α) is not generated by any stable canonical parse tree.*

The theorem is divided into two propositions that are proved independently.

Proposition 18.2 *Given a determinization of a universal CLR(1) parser, the parsing corresponding to a stable canonical parse tree is always ε-loop free.*

Proof: Consider a canonical parse tree t where in the corresponding parsing, for some grammar production $X \to \varepsilon$, there are two reductions of $X \to \varepsilon$ without any shifts in between. We want to prove that t is not stable canonical.

Denote by I the state of the configuration from which the first of the above reductions are done. Moreover, denote by t_1 and t_2 the results of removing from t the empty leaf rightsides corresponding to the first and the second, respectively, of the above reductions with $X \to \varepsilon$.

Since both t_1 and t_2 are sub-parse trees of t, if t was stable canonical, both t_1 and t_2 would be stable canonical. In the parsings corresponding to t_1 in state I on input X we shift X, whilst in the parsing corresponding to t_2 in state I on input X we reduce $X \rightarrow \varepsilon$. But only one action is canonical in state I on input X, so t_1 and t_2 cannot both be canonical. Thus t is not stable canonical. ■

Proposition 18.3 *Given a universal CLR(1) parser and an $(\mathcal{R}^{\prec})^{\mathcal{U}}$-minimal determinization. The number of actions carried out in an ε-protected $\mathcal{P}$-parsing of a production $X \rightarrow \alpha$ is $O(|\alpha|)$.*

Proof: We prove the proposition by showing that before the nth shift of an ε-protected $\mathcal{P}$-parsing there is $O(n)$ actions.

By the definition of ε-protected parsing, the number of reductions of empty strings is less than or equal to $k_1 n \in O(n)$, where k_1 is the number of different productions of the empty string.

Only shifts and reductions of empty strings increase the size of the stack. As there are at most $O(n)$ of these operations, there can at most be $O(n)$ operations decreasing the size of the stack. Thus, there is at most $O(n)$ reductions of more than one symbol, i.e. reductions with productions $X \rightarrow X_1 \cdots X_n$ where $n > 1$.

Denote by k_2 the number of different grammar symbols. We complete the proof by showing that there cannot be $k_2 \in O(1)$ consecutive reductions of single symbols.

Suppose we, at some stage in the canonical parsing, make k_2 consecutive reductions of single symbols, then a sub-sequence of these reductions must be of the form

$$\text{reduce } X_k \rightarrow X_0, \ldots, \text{reduce } X_0 \rightarrow X_1$$

starting and finishing in some state I. Notice, that if we prefix this sequence of actions with the action shift X_0 and suffix it with the action accept, then we get the action sequence of the non-trivial parse tree (which laid down looks like) $X_0 \rightarrow X_1 \rightarrow \cdots X_k \rightarrow X_0$ for the derived production $X_0 \rightarrow^* X_0$. Denote this parse tree by r.

By the definition of the universal CLR(1) parser, we can find a parse tree t such that for some index l we have $\text{action}(t, l-1) = \text{shift}X_0$ and $\text{state}(t, l) = I$. Let t be a minimal such parse tree; then after configuration l in t we do not carry out the above sequence of actions, so one of the subsequent k actions must be non-canonical.

Denote by t' the result of rooting the parse tree r in the above occurrence of X_0 in the frontier of t. Then, by definition, $t\,\mathcal{R}^{\prec}\,t'$, implying $t\,(\mathcal{R}^{\prec})^{\mathcal{U}}\,t'$.

Up to configuration l, the parsings of t and t' coincide, but from there we carry out k canonical actions in t' while in t one of the following k actions is non-canonical. Since our determinization is $(\mathcal{R}^{\prec})^{\mathcal{U}}$-minimal, this implies $t'\,(\mathcal{R}^{\prec})^{\mathcal{U}}\,t$, contradicting $t\,(\mathcal{R}^{\prec})^{\mathcal{U}}\,t'$ since $(\mathcal{R}^{\prec})^{\mathcal{U}}$ is a strict partial ordering. Thus there cannot be k_2 consecutive reductions of single symbols. ■

19 The resolved parser

In this section we solve problem 5, page 33. By

Theorem 19.1 *Given a universal CLR(1) parser, let $\mathcal{P}$ be a complete $(\mathcal{R}^{\prec})^{\mathcal{U}}$-minimal determinization with a stable set of canonical parse trees. Then $\mathcal{P}$ is a linear time parser.*

Proof: The result follows if we can prove that any parsing with $\mathcal{P}$ is ε-loop free, for then by Proposition 18.3 the parsing carries out a number of actions at most linear in the size of the leftside of the input production, and the time spent on one action is a constant depending on the original grammar.

Consider some successful parsing with $\mathcal{P}$, i.e. a parsing corresponding to a canonical parse tree. By assumption, all canonical parse trees are stable canonical, so by Proposition 18.2, the parsing is ε-loop free.

Now consider an arbitrary incomplete parsing with $\mathcal{P}$. Suppose that we have carried out the sequence of actions $\boldsymbol{a}$, and that we now are in state I on input X with a as the canonical action. We want to prove that $\boldsymbol{a}a$ is ε-loop free.

By the definition of universal CLR(1) parsers, there is a parse tree t where $\boldsymbol{a}$ is a prefix of the action sequence, $\text{state}(t, |\boldsymbol{a}|) = I$, $\text{input}(t, |\boldsymbol{a}|) = X$, and $\text{action}(t, |\boldsymbol{a}|) = a$.

Denote by $X \rightarrow^* \alpha$ the production derived by t. Since $X \rightarrow^* \alpha$ is a derived production, by completeness, the parsing of $X \rightarrow^* \alpha$ with $\mathcal{P}$ is successful, and hence ε-loop free. But this parsing starts with the sequence of actions $\boldsymbol{a}$ arriving at state I with input X; from there it carries out action a. Thus $\boldsymbol{a}a$ is a prefix of the action sequence for some successful parsing with $\mathcal{P}$, and hence $\boldsymbol{a}a$ must be ε-loop free, as desired. ■

20 The CLR(1) generalization

Finally, we are in the position to describe the generalization of canonical LR(1) technique to deal with ambiguous grammars.

Algorithm C: Let a grammar $\mathcal{G}$ and a finite set $\mathcal{E}_{in}$ of parse tree pairs be given. Denote by $\mathcal{U}$ the universal CLR(1) parser for $\mathcal{G}$. The algorithm decides if there exists a *matching* pair, meaning a pair $(\mathcal{P}, \mathcal{E}_{out})$ where $\mathcal{P}$ is a determinization of $\mathcal{U}$ and where $\mathcal{E}_{out}$ is a finite set of parse tree pairs such that $(\mathcal{E}_{in} \cup \mathcal{E}_{out})$ $\mathcal{U}$-prioritizes $\mathcal{P}$. In this case, it finds such a matching pair $(\mathcal{P}, \mathcal{E}_{out})$, where $\mathcal{P}$ is a linear time parser, and $\mathcal{E}_{out}$ is minimal given $\mathcal{P}$.

C.1. Construct the set $\mathcal{R} = \mathcal{R}(\mathcal{G})$.

C.2. Check that $(\mathcal{E}_{in} \cup \mathcal{R})^{\prec}$ spans a universal parser ordering $\mathcal{O}$. If not, report failure.

C.3. Search among the basis complete $\mathcal{O}$-minimal determinizations $\mathcal{P}$ for one for which the following sub-routine successfully finds a matching set $\mathcal{E}_{out}$ of parse tree pairs; if no such $\mathcal{P}$ exists, report failure:

C.3.1. Construct a representation of a set M such that

- M contains all minimal non-canonical parse trees.
- If the determinization is stable then M is exactly the set of minimal non-canonical parse trees.
- It is decidable whether M is infinite.

If M is infinite, try another determinization.

C.3.2. Let M^- be the set of parse trees in M that are not second coordinates in $\mathcal{E}_{in}$.

C.3.3. Apply ε-protected parsing with $\mathcal{P}$ to all derived productions generated by parse trees in M^-. If one of these derived productions is not accepted, try another determinization.

C.3.4. Set $\mathcal{E}_{out} := \{(t,u)|u \in M^-$ and t is the canonical equivalent of $u\}$.

C.3.5. Check that $(\mathcal{E}_{in} \cup \mathcal{E}_{out} \cup \mathcal{R})^{\prec}$ spans a universal parser ordering in which $\mathcal{P}$ is minimal; if not, try another $\mathcal{O}$-minimal determinization.

C.3.6. Return successfully the matching pair $(\mathcal{P}, \mathcal{E}_{out})$.

The algorithm will be illustrated by some examples in the next section.

Lemma 20.1 *All steps in Algorithm C are implementable.*

Proof: Proposition 13.1 shows that Step C.1 can be implemented, and Theorem 16.1 shows that step C.2 and step C.3.5 can be implemented. The loop in Step C.3 only considers basis complete determinizations, so Step C.3.1 can be implemented by Theorem 17.1. Also, all the considered determinizations are minimal with respect to $\mathcal{O} = ((\mathcal{E}_{in} \cup \mathcal{R})^{\prec})^{\mathcal{U}}$ which contains $(\mathcal{R}^{\prec})^{\mathcal{U}}$, so steps C.3.3–C.3.4 are implementable by Theorem 18.1. All the other steps are trivially implementable, so this completes the proof of the lemma. ∎

We will now turn to the proof of correctness. Recall from Corollary 13.4, that a determinization $\mathcal{P}$ and and a finite set $\mathcal{E}_{out}$ of parse tree pairs match each other if and only if

(i) All $(\mathcal{E}_{in} \cup \mathcal{E}_{out})^{\prec}$-minimal parse trees are $\mathcal{P}$-canonical.

(ii) $(\mathcal{E}_{in} \cup \mathcal{E}_{out} \cup \mathcal{R})^{\prec}$ spans an ordering of $\mathcal{U}$ in which $\mathcal{P}$ is minimal.

Lemma 20.2 *If Algorithm C successfully return $(\mathcal{P}, \mathcal{E}_{out})$ in step C.3.6, then $\mathcal{P}$ and $\mathcal{E}_{out}$ match each other, $\mathcal{P}$ is not matched by any cardinality-wise smaller set $\mathcal{E}'_{out}$, and $\mathcal{P}$ is a linear time parser.*

Proof: By construction of $\mathcal{E}_{out}$ in steps C.3.1–C.3.2, all minimal non-canonical parse trees are second coordinates either in $\mathcal{E}_{in}$ or in $\mathcal{E}_{out}$, so no non-canonical parse trees can be $(\mathcal{E}_{in} \cup \mathcal{E}_{out})^{\prec}$-minimal. Thus (i) follows. Moreover, condition (ii) follows directly from the test in step C.3.5, so, indeed, we may conclude that $\mathcal{P}$ and $\mathcal{E}_{out}$ match each other.

Let $\mathcal{E}'_{out}$ be any set of parse tree pairs matching $\mathcal{P}$. Since $\mathcal{P}$ is prioritized, the set of canonical parse trees is stable, so the set M constructed in step C.3.1 contains exactly the set of minimal non-canonical parse trees. By condition (i) this implies that $\mathcal{E}'_{out}$ has to contain as second coordinates all second coordinates of $\mathcal{E}_{out}$, so $\mathcal{E}'_{out}$ cannot have smaller cardinality than $\mathcal{E}_{out}$.

From condition (ii), it follows that $\mathcal{P}$ is $(\mathcal{R}^{\prec})^{\mathcal{U}}$-minimal. Moreover, since $\mathcal{P}$ is prioritized, it follows that $\mathcal{P}$ is complete, and that the set of canonical parse trees is stable. Thus, by Theorem 19.1, we have that $\mathcal{P}$ is a linear time parser. ■

Lemma 20.3 *If Algorithm C reports failure, there is no matching pair.*

Proof: Suppose that the algorithm reports failure, but that there exists a matching pair $(\mathcal{P}', \mathcal{E}'_{out})$. Denote by (i)$'$ and (ii)$'$ condition (i) and (ii) with $\mathcal{P}'$ and $\mathcal{E}'_{out}$ in place of $\mathcal{P}$ and $\mathcal{E}_{out}$, respectively. Thus we have that $\mathcal{E}'_{out}$ is finite, and that conditions (i)$'$ and (ii)$'$ are satisfied.

Condition (ii)$'$ implies that we pass the test in step C.2 arriving at the loop in step C.3. Thus, reporting failure implies that we have tried an iteration with all basis complete $((\mathcal{E}_{in} \cup \mathcal{R})^{\prec})^{\mathcal{U}}$-minimal determinization. Since $\mathcal{P}'$ is prioritized, it is complete, hence basis complete. Moreover, by condition (ii)$'$, $\mathcal{P}'$ is $((\mathcal{E}_{in} \cup \mathcal{R})^{\prec})^{\mathcal{U}}$-minimal, so there will be an iteration with $\mathcal{P} = \mathcal{P}'$.

Since $\mathcal{P}'$ is prioritized, the set of canonical parse trees is stable, so the set M constructed in step C.3.1 contains exactly the set of minimal non-canonical parse trees. By condition (ii)$'$ the two finite sets $\mathcal{E}_{in}$ and $\mathcal{E}'_{out}$ contain as second coordinates all parse trees in M, so M is finite, and we pass the test in step C.3.2.

Again since $\mathcal{P}'$ is prioritized, all parse trees have stable canonical equivalents. Thus, by Theorem 18.1, all the ε-protected parsings are accepting, and hence we pass the test in step C.3.3.

Since all the first coordinates in the set $\mathcal{E}_{out}$ constructed in step C.3.4 are canonical, $\mathcal{E}_{out}$ is contained in $\mathcal{E}^{\prec}$, for any set $\mathcal{E}$ prioritizing $\mathcal{P}$. Thus $(\mathcal{E}_{in} \cup \mathcal{E}_{out} \cup \mathcal{R})^{\prec} \subseteq (\mathcal{E}_{in} \cup \mathcal{E}'_{out} \cup \mathcal{R})^{\prec}$, so condition (ii)$'$ implies that we pass the check in step C.3.5, but this means that the iteration will be successful, contradicting that we reported failure. ■

Proof of Main Theorem 6.1 for the canonical LR(1) technique: Follows directly from the correctness of Algorithm C, and this correctness is given by the above lemmas. ■

We have now completed the proof of Main Theorem 6.1 for the canonical LR(1) technique. Similar constructive proofs can be given for all the other LL and LR techniques. In the next section we will go through some examples illustrating Algorithm C, and in Section 22, there will be a short discussion of the efficiency of the algorithm.

21 Examples

We will now apply the generalized CLR(1) technique to some of our previous examples. Basically, we will just sum up what we already know about these examples, by relate them to Algorithm C. First we will consider the grammar from example (4) that we have used throughout this part, and afterwards, we will discuss how the algorithm reports failure for the "bad" grammars from Section 11. In all examples, we will assume that $\mathcal{E}_{in} = \emptyset$.

Recall that the productions of the grammar from example (4) are $E \rightarrow F$, $F \rightarrow E$, and $F \rightarrow FF$. As we saw in Section 14, the grammar has the following universal CLR(1) parser:

$\mathcal{U}_{(4)}$:

INIT	STATE	*ACTIONS*			*GOTO*	
		E	F	\$	E	F
E	1	s	s	$\div$	2	3
	2	r $F \rightarrow E$	r $F \rightarrow E$	a, r $F \rightarrow E$	$\div$	$\div$
	3	s, r $E \rightarrow F$	s, r $E \rightarrow F$	r $E \rightarrow F$	4	5
	4	r $F \rightarrow E$	r $F \rightarrow E$	r $F \rightarrow E$	$\div$	$\div$
	5	s, r $F \rightarrow FF$, r $E \rightarrow F$	s, r $F \rightarrow FF$, r $E \rightarrow F$	r $F \rightarrow FF$, r $E \rightarrow F$	4	5
F	6	s	s	$\div$	4	7
	7	s, r $E \rightarrow F$	s, r $E \rightarrow F$	a, r $E \rightarrow F$	4	5

Now, the input for Algorithm C is ready. Step C.1 tells us to construct the set $\mathcal{R}$:

$$\mathcal{R}_{(4)} : \left\{ \left(E \,,\, \begin{array}{c} E \\ | \\ F \\ | \\ E \end{array} \right) , \left(F \,,\, \begin{array}{c} F \\ | \\ E \\ | \\ F \end{array} \right) \right\} .$$

Step C.2 asks us to check that $\mathcal{R}^{\prec}_{(4)}$ spans an ordering of $\mathcal{U}_{(4)}$. From the example in Section 16, it follows that $\mathcal{R}^{\prec}_{(4)}$ spans the ordering:

$\mathcal{O}_{(4)}$:

STATE	*ACTIONS*		
	E	F	\$
1	$\div$	$\div$	$\div$
2	$\div$	$\div$	a $\prec$ r $F \rightarrow E$
3	s $\prec$ r $E \rightarrow F$	s $\prec$ r $E \rightarrow F$	$\div$
4	$\div$	$\div$	$\div$
5	s $\prec$ r $E \rightarrow F$, r $F \rightarrow FF$ $\prec$ r $E \rightarrow F$	s $\prec$ r $E \rightarrow F$, r $F \rightarrow FF$ $\prec$ r $E \rightarrow F$	r $F \rightarrow FF$ $\prec$ r $E \rightarrow F$
6	$\div$	$\div$	$\div$
7	s $\prec$ r $E \rightarrow F$	s $\prec$ r $E \rightarrow F$	a $\prec$ r $E \rightarrow F$

For the loop in step C.3, we have to pick some basis complete $\mathcal{O}_{(4)}$ minimal determinization. The only freedom left in this choice is in state I on input E or F, where we can choose between reducing with $F \rightarrow FF$ and shifting. In both cases we we choose reduce $F \rightarrow FF$.

Thus our determinization $\mathcal{P}_{(4)}$ is obtained by choosing the first action in each entry of the action table in $\mathcal{U}_{(4)}$. Concerning step C.3.1, in Section 17, we found the following trace graph representation of the set M:

$$
M_{(4)}:\quad
\begin{tikzcd}
 & v_1 \arrow[r, "E\to F\bullet"] & t_1 \\
s_1 \arrow[ur, "F\to\bullet E"] \arrow[r, "F\to\bullet FF"] \arrow[dr, "E\to\bullet F"'] & v_2 \arrow[r, "F\to F\bullet F"] & t_2 \\
 & v_3 \arrow[r, "F\to E\bullet"] & t_3
\end{tikzcd}
$$

Now, as the set $\mathcal{E}_{out}$ constructed in steps C.3.2–C.3.4, we get

$$
\mathcal{E}_{(4)}:\left\{
\left(\begin{array}{ccc} & & E \\ & & | \\ E & , & F \\ & & | \\ & & E \end{array}\right),
\left(\begin{array}{ccc} & & F \\ & & | \\ F & , & E \\ & & | \\ & & F \end{array}\right),
\left(\begin{array}{ccc} F & & F \\ |\backslash & & |\ \backslash \\ FF & , & F\ F \\ \ \ |\backslash & & \ \ |\backslash \\ \ \ FF & & \ \ FF \end{array}\right)
\right\}.
$$

Notice that in this example, we have $\mathcal{E}_{in}\cup\mathcal{E}_{out}\cup\mathcal{R}=\mathcal{E}_{(4)}$. Thus step C.3.5 asks us to check that $\mathcal{E}_{(4)}^{\prec}$ spans a universal parser ordering in which $\mathcal{P}$ is minimal. From the example in Section 16 it follows that indeed this is the case. Thus, in step C.3.6, successfully we return the matching pair $(\mathcal{P}_{(4)},\mathcal{E}_{(4)})$.

We will now discuss how and why the algorithm reports failure for the bad grammars from Section 11. First, we have the dangling else grammar (10) with production set $\{E\to\mathtt{a}\,E\,\mathtt{b}\,E,\ E\to\mathtt{a}E\}$. In the universal CLR(1) parser there will be a conflict on input $\mathtt{b}$ between shifting and reducing with $E\to\mathtt{a}\,E\,\mathtt{b}\,E$. If we choose to shift, we get the following trace graph representation for M:

$$
M_{(10)}:\quad
\begin{tikzcd}
s \arrow[r, "E\to\bullet\mathtt{a}E"] & v \arrow[loop right, "E\to\mathtt{a}E\,\mathtt{b}E\bullet"] \arrow[d, "E\to\mathtt{a}E\bullet\mathtt{b}E"] \\
 & t
\end{tikzcd}
$$

The trace graph contains a loop, so the set M is infinite, meaning that the test in step C.3.1 fails. Notice that M is exactly the second coordinates of the infinite prioritizing set $\mathcal{E}_{(10)}$, page 29. If instead we had chosen reduce $E\to\mathtt{a}\,E\,\mathtt{b}\,E$ over shift, we would have got a trace graph representing the first coordinates. In either case, we have to report failure.

For the grammar in (11) with production set $\{A\to BB, B\to\varepsilon\}$, the universal parser will contain a conflict in state $\mathit{INIT}(A)$ on input B between shifting and reducing with $B\to\varepsilon$. Depending on how we resolve this conflict we get the either the set

$$
\mathcal{E}_{(11)}:\left\{
\left(\begin{array}{ccc} A & & A \\ |\backslash & & |\backslash \\ BB & , & BB \\ | & & \ \ | \\ \varepsilon & & \ \ \varepsilon \end{array}\right)
\right\},
$$

or the same set but with the parse trees in the pair swapped. In either case Algorithm B will make a recursive call with the parse tree pair

$$\left(\begin{array}{c} A \\ |\backslash \\ BB \\ |\ | \\ \varepsilon\ \varepsilon \end{array} , \begin{array}{c} A \\ |\backslash \\ BB \\ |\ | \\ \varepsilon\ \varepsilon \end{array}\right)$$

returning $\bot$. Thus the test in step C.3.5 always fails since neither of the possibilities for $\mathcal{E}_{out}$ spans a universal parser ordering, and hence we have to report failure.

The grammar from example (12) has production set $\{S{\rightarrow}\mathtt{aaa}, S{\rightarrow}E\,\mathtt{ab}, E{\rightarrow}\mathtt{a}\}$. In the universal CLR(1) parser there will be a conflict in state $\mathit{GOTO}(\mathit{INIT}(S), \mathtt{a})$ on input $\mathtt{a}$. No matter how we resolve the conflict, there will be one of the parse trees from $M = M^-$ that cannot be parsed in step C.3.3, so we have to report that the determinization is not complete.

The grammar in example (13) has production set $\{L{\rightarrow}LL, L{\rightarrow}\varepsilon\}$. For this grammar, in step C.1, we get

$$\mathcal{R}_{(13)} : \left\{ a_{(13)} : \left(L \ , \begin{array}{c} L \\ |\backslash \\ LL \\ | \\ \varepsilon \end{array} \right) ,\ b_{(13)} : \left(L \ , \begin{array}{c} L \\ |\backslash \\ LL \\ \ \ | \\ \ \ \varepsilon \end{array} \right) \right\}$$

Recalling the analysis of this example, Algorithm B will give us

$$((\mathit{GOTO}(\mathit{GOTO}(\mathit{INIT}(L), L), L), \$), \text{reduce } L{\rightarrow}LL, \text{reduce } L{\rightarrow}\varepsilon) \in P(a_{(13)})$$
$$((\mathit{GOTO}(\mathit{GOTO}(\mathit{INIT}(L), L), L), \$), \text{reduce } L{\rightarrow}\varepsilon, \text{reduce } L{\rightarrow}LL) \in P(b_{(13)}).$$

Thus $\mathcal{R}_{(13)}$ does not span a universal parser ordering. This failure will be reported already in Step C.2.

22 Some technical remarks

In the presentation of our generalization of the canonical LR(1) technique, we have not worried about efficiency, but only about computability. Clearly, our algorithm is exponential in the size of the grammar, as follows just from the fact that we use the canonical LR(1) technique as a front end. In any case, a formal complexity analysis does not make much sense, since grammars are not arbitrary, but written by humans. However, there is a real issue in avoiding trying too many determinizations in the loop of Algorithm C in step C.3.

A first approach to getting down the number of relevant determinizations was already discussed in Section 12. The idea is to let the user works modularly, as advocated in Section 9, introducing only a little new ambiguity in each step, for then set $\mathcal{E}_{in}$ together with $\mathcal{R}$ will largely determine the parsing, keeping the number of relevant determinizations small.

Of course, we cannot rely on any discipline on the user's part. Instead we can improve the algorithm itself by exploiting the fact that not all choices are independent. Take the situation from example (4) where we had the choice in state I on input E or F between reducing with $F \rightarrow FF$ and shifting. It should be clear that we have to make the same choice for both inputs, for otherwise, there is no chance of getting a set $\mathcal{E}_{out}$ spanning a universal parser ordering. Exploiting such observations algorithmically is possible. Notice, that, if in the above choice, we had picked to shift instead of reducing with $F \rightarrow FF$, the iteration would still have been successful, though the returned parser would have been right associative instead of left associative. This situation is expected to be typical, i.e. that if we take dependencies between choices into consideration, then often any determinization is good, so we need only one iteration of the loop. Finally, one could introduce some heuristics that could guide the search for a good determinization.

An other possibility for improving the efficiency is to use as front end the LALR(1) techniques rather than the idealistic canonical LR(1) technique, and then apply exactly the same algorithm for the generalization. Any successfully returned pair would still satisfy all the desired properties, but sometimes, the algorithm would discard determinizations unduly. The problem is that, when applied directly to a universal LALR(1) parser, our spanning algorithm might find some extra projections which turn out to be incompatible with the true projections. The reason for this problem is that universal LALR(1) parsers do not satisfy the full completabily property of universal CLR(1) parsers. Instead they satisfy the following weaker version: suppose in a parsing we arrive at a configuration of the form (α, β). Then for some β', from configuration (α, β'), the parsing can be completed successfully. In order to recognize all good determinizations of a universal LALR(1) parser, it seems as if we need to do some simulation of the canonical LR(1) technique. Such a perfect solution for the LALR(1) technique makes no sense from a practical view point, for the LALR(1) technique is itself a clever example of sacrificing generality for speed. Thus a quick fix, just applying Algorithm C directly to the LALR(1) technique, would suit it just right.

Summary

Since the sixties, for large classes of unambiguous grammars we have had the LL and LR techniques that can check their (semantic) unambiguity and construct complete linear time parsers. Up to this time no similarly powerful techniques have been presented for ambiguous grammars.

We generalized the notion of context-free grammars by changing the basis from sentences to derived productions, formalized identification of ambiguity by generating sets of parse tree pairs, and introduced the idea of prioritizing parsing. Prioritizing parsing means finding a generating set of parse tree pairs prioritizing a complete parser; viewed as priorities, the parse tree pairs describe exactly the canonical parse trees found by the parser for the sentences.

We saw that the canonical LR(1) techniques can be generalized to prioritize the parsing for large classes of ambiguous grammars, giving us a generating set of parse tree pairs prioritizing a complete linear time parser. All the other LL and LR techniques can be generalized similarly. The user can insert his own priorities thereby specifying a the canonical parse trees of the generated parser. Thus, now we have as powerful techniques for ambiguous grammars as we had for unambiguous grammars.

Acknowledgement

Special thanks to Mark-Jan Nederhof, Eljas Soisalon-Soininen, and "Referee A" from TOPLAS, all for having put a very large effort into helping me with the presentation of this work. Many others have been helpful with suggestions, comments, and encouragement during the years I have been working on the theory and algorithms just presented. Among these are Andrew Appel, Dines Bjørner, Andrezej Blikle, Alan Demers, Martin Farach, Tony Hoare, Neil Jones, Bernard Lang, Bill McColl, Colin McDiarmid, Hans Rischel, Søren Riis, Mary Ryan, Barbara Ryder, Vincent Sgro, Anders Thorup, and various anonymous referees.

References

[1] Aho, A.V., Johnson, S.C, and Ullman, J.D. Deterministic parsing of ambiguous grammars. *Comm. ACM 18, 8* (1975), 441–452.

[2] Aho, A.V., Sethi, R., and Ullman, J.D. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, Reading, Mass., 1986.

[3] Blikle, A. Denotational engineering. *Science of Computer Programming 12* (1989), 207–253.

[4] Blikle, A., Tarlecki, A., and Thorup, M. On conservative extensions of syntax in system development. In *Images of Programming, Dedicated to the memory of A.P. Erchov.* D. Bjørner, and V. Kotov, Eds., North Holland, Amsterdam, 1991, pp. 209–235.

[5] Blikle, A., and Thorup, M. On conservative extensions of syntax in the process of system development. In *VDM'90, VDM and Z—Formal Methods in Software Development, Proceedings; Lect. Notes Comp. Science 428.* D. Bjørner, C.A.R. Hoare, and H. Langmaack, Eds., Springer-Verlag, Berlin, London, 1990, pp. 504–525.

[6] Chomsky, N. Three models for the description of language. *IRE Trans. on Information Theory IT2, 3* (1956), 113–124.

[7] Chomsky, N., and Schützenberger, M.P. The algebraic theory of context-free languages. In *Computer Programming and Formal Systems; Studies in logic and the foundations of mathematics.* P. Braffort, and D. Hirschberg, Eds., North Holland, Amsterdam, 1963, pp. 118–161.

[8] Demers, A.J. Skeletal LR parsing. In *IEEE Conf. Rec. of the 15th Ann. Symp. on Switching and Automata Theory.* 1974, pp. 185–198.

[9] Earley, J. Ambiguity and precedence in syntax description. *Acta Informatica 4, 2* (1975), 183–192.

[10] Futatsugi, K., Goguen, J.A., Jouannaud, J.-P., and Meseguer, J. Principles of OBJ2. *Proceedings of the 12th ACM Symposium on Principles of Programming Languages.* ACM Press, New York, 1985, pp. 52–66.

[11] Ghezzi, C. and Mandrioli, D. Incremental Parsing. *Trans. Progr. Lang. Syst. 1,* 1 (1979), 58–71.

[12] Goguen, J.A., and Meseguer, J. Completeness of many-sorted equational logic. *Houston Journal of Mathematics 11,* 3 (1985), 307–334.

[13] Johnson, S.C. *Yacc—Yet Another Compiler Compiler.* Computer Science Technical Report 32, AT&T Bell Laboratories, Murray Hill, NJ, 1975.

[14] Heering, J., Hendriks, P.R.H., Klint, P., and Rekers, J. Incremental Generation of Parsers. *IEEE Tans. on Software Engineering 16,* 12 (1990), 1344–1351.

[15] Heering, J., Klint, P., and Rekers, J. The syntax definition formalism SDF—reference manual. *SIGPLAN Notices 24,* 11 (1989), 43–75.

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

[17] Kruskal, J.B. Well-quasi-ordering, the tree theorem, and Vazsonyi's Conjecture. *Trans. Amer. Math. Soc. 95* (1960), 210–225.

[18] Nash-Williams, C.S.J.A. On well-quasi-ordering finite trees. *Proc. Camb. Phil. Soc. 59* (1963), 833–835.

[19] Pennello, T. and DeRemer, F. A forward move algorithm for LR error recovery. *Proceedings of the 5th ACM Symposium on Principles of Programming Languages.* ACM Press, New York, 1979, pp. 241–254.

[20] Sippu, S., and Soisalon-Soininen, E. *Parsing Theory, Vol 1: Languages an Parsing; EATCS Monographs on Theoretical Computer Science 15.* Springer-Verlag, Berlin, London, 1988.

[21] Sippu, S., and Soisalon-Soininen, E. *Parsing Theory, Vol 2: LR(k) and LL(k) Parsing; EATCS Monographs on Theoretical Computer Science 20.* Springer-Verlag, Berlin, London, 1990.

[22] Simpson, S.G. Nonprovability of certain combinatorial properties of finite trees. In *Harvey Friedman's Research on the Foundations of Mathematics.* L.A. Harrington, M.D. Morley, A. Ščedrov, and S.G. Simpson, Eds. North Holland, Amsterdam, 1985, pp. 87–119.

[23] Soisalon-Soininen, E., and Tarhio, J. Looping LR parsers. *Information Processing Letters 26* (1988), 251–253.

[24] Stoy, J.D. *Denotational Semantics—The Scott–Stratchey Approach to Programming Language Theory.* MIT Press, Cambridge, Mass., 1977.

[25] Tarhio, J. LR parsing of some ambiguous grammars. *Information Processing Letters 14*,3 (1982), 101–103.

[26] Thorup, M. Fully Controlled Ambiguity of some Context-Free Grammars. Technical Report PRG-TR-17-91, Programming Research Group of Oxford University, Oxford, 1991.

[27] Thorup, M. Ambiguity for Incremental Parsing and Evaluation. Technical Report PRG-TR-24-92, Programming Research Group of Oxford University, Oxford, 1992.

[28] Wharton, R.W. Resolution of ambiguity in parsing. *Acta Informatica 6, 4* (1976), 387-395.

[29] Wirth, N. The programming language PASCAL. *Acta Informatica 1* (1971), 35–63.

[30] Wirth, N. *Programming in Modula-2 (3rd corr. ed.).* Springer-Verlag, Berlin, New York, 1985.