THIS DOCUMENT IS THE APPENDIX TO THE FOLLOWING PAPER:

Controlled Grammatic Ambiguity

MIKKEL THORUP
University of Copenhagen

The body of this paper is to appear in *ACM Transactions on Programming Languages and Systems*, Vol. 16, No. 3, May 1994.

APPENDIX A. THE PROPERTIES OF $\mathcal{R}$

This appendix is devoted to proving Theorem 8.1, on which hings the correctness of the techniques presented in this paper. Recall the statement of Theorem 8.1:

> *Let $\mathcal{P}$ be a determinization of a universal LL or LR parser $\mathcal{U}$, and let $\mathcal{E}$ be a set of parse tree pairs. Then $\mathcal{E}$ $\mathcal{U}$-prioritizes $\mathcal{P}$ if and only if the following conditions are satisfied:*
> *(i) All $\mathcal{E}^{\prec}$-minimal parse trees are $\mathcal{P}$-canonical.*
> *(ii) $(\mathcal{E} \cup \mathcal{R})^{\prec}$ spans an ordering of $\mathcal{U}$ in which $\mathcal{P}$ is minimal.*

Our first step in the proof is to show the following simple result:

OBSERVATION A.1. *If a set $\mathcal{E}$ of parse tree pairs prioritizes a parser $\mathcal{P}$, then $\mathcal{R} \subseteq \mathcal{E}^{\prec}$.*

PROOF. Assume that $\mathcal{E}$ prioritizes $\mathcal{P}$. Then, in particular, $\mathcal{P}$ is complete, and the set of $\mathcal{P}$-canonical parse trees is stable (i.e. closed under the sub-parse trees operation). Consider any parse tree pair $(\underline{\mathrm{pt}}(X), t) \in \mathcal{R}$. Since $\mathcal{E}$ prioritizes $\mathcal{P}$ and since $\underline{\mathrm{pt}}(X)$ has no proper sub-parse trees, if $\underline{\mathrm{pt}}(X)$ is not $\mathcal{P}$-canonical, it must be a second coordinate in $\mathcal{E}$. Hence no parse tree containing $\underline{\mathrm{pt}}(X)$ can be canonical, so, in particular, there cannot be a canonical parse tree for $X \rightarrow^* X$. This contradicts that $\mathcal{P}$ is complete, so we may conclude that $\underline{\mathrm{pt}}(X)$ is a $\mathcal{P}$-canonical parse tree. Thus, since $\mathcal{E}$ prioritizes $\mathcal{P}$, we have $(\underline{\mathrm{pt}}(X), t) \in \mathcal{E}^{\prec}$, as desired. □

The main step in establishing Theorem 8.1 is to prove the following:

KEY LEMMA A.2. *Let $\mathcal{E}$ be a set of parse tree pairs. If $(\mathcal{E} \cup \mathcal{R})^{\prec}$ is irreflexive, then is $\mathcal{E}^{\prec}$ is well-founded.*

Before proving the key lemma, first we will see how it implies Theorem 8.1.

PROOF THAT KEY LEMMA A.2 IMPLIES THEOREM 8.1.

$\Leftarrow$: Assume (i) and (ii). Clearly, the two together imply that the $\mathcal{P}$-canonical and the $\mathcal{E}^{\prec}$-mimimal parse trees coincide. Moreover, since all universal parser orderings are strict, by (ii) we have that $(\mathcal{E} \cup \mathcal{R})^{\prec}$ is irreflexive, and hence, by

Key Lemma A.2, that $\mathcal{E}^{\prec}$ is well-founded. Thus, by definition, $\mathcal{E}$ prioritizes $\mathcal{P}$, which, together with (ii), implies that $\mathcal{E}$ $\mathcal{U}$-prioritizes $\mathcal{P}$, as desired.

$\underline{\Rightarrow}$: Assume that $\mathcal{E}$ $\mathcal{U}$-prioritizes $\mathcal{P}$. Then, in particular, $\mathcal{E}$ prioritizes $\mathcal{P}$, so by Observation A.1, we have $\mathcal{R} \subseteq \mathcal{E}^{\prec}$. Thus $\mathcal{E}^{\prec} = (\mathcal{E} \cup \mathcal{R})^{\prec}$, so (i) and (ii) follows directly from the definition of $\mathcal{U}$-prioritizing.

□

The rest of this appendix is devoted to proving Key Lemma A.2. First some technical definitions. Let t be a non-trivial parse tree with $(n_1, \ldots, n_p)$ being the sequence of sons of the root. Then for $i = 1, \ldots, p$, by $t(i)$ we denote the sub-parse tree descending from n_i, i.e. $t = \underline{\text{pt}}(X; t(1), \ldots, t(p))$ for some X. An auxiliary tree embedding relation $\sqsubseteq$ is defined as the reflexive partial ordering generated by the following conditions:

—If t is a non-trivial parse tree with the root production $X \rightarrow X_1 \cdots X_p$, then for $i = 1, \ldots, p$, we have $t(i) \sqsubseteq t$.

—If t, u are non-trivial parse trees with the common root production $X \rightarrow X_1 \cdots X_p$, and for $i = 1, \ldots, p$, we have $t(i) \sqsubseteq u(i)$, then $t \sqsubseteq u$.

Now, the following lemma can be seen as a variant for parse trees of Kruskal's Tree Theorem [Kruskal 1960].

LEMMA A.3. *Given an infinite sequence $t_0 \cdots t_k \cdots$ of parse trees there exist two numbers i, j such that $i < j$ and $t_i \sqsubseteq t_j$.*

PROOF. (essentially that of Nash-Williams [Nash-Williams 1963]) In the proof we identify a set I of numbers (≥ 0) with the sequence consisting of the numbers in ascending order, and then by $I[j]$ we denote the $(j+1)$th number in the sequence.

Suppose there exists a counterexample to the lemma. Let $\boldsymbol{t}$ be a minimal counterexample in the lexicographic ordering induced by $\sqsubseteq$: starting with the empty sequence, suppose we have obtained the sequence $t_0 \cdots t_{k-1}$ of parse trees; then let t_k be a $\sqsubseteq$-minimal parse tree such that $t_0 \cdots t_k$ is a prefix of a counterexample.

There are only finitely many grammar symbols, so $\boldsymbol{t}$ will contain infinitely many non-trivial parse trees. Moreover, there are only finitely many grammar productions, so there is a grammar production $X \rightarrow X_1 \cdots X_p$ which is root production of infinitely many of the parse trees in $\boldsymbol{t}$. Denote by I the infinite set of indices of parse trees with $X \rightarrow X_1 \cdots X_p$ as root production.

Fix $l \in \{1, \ldots, p\}$ and let J be any infinite subset I. We will now find an infinite subset L of J so that the infinite subsequence $t_{L[0]}(l) \cdots t_{L[k]}(l) \cdots$ of $\boldsymbol{t}$ is $\sqsubseteq$-ascending.

Denote by K the subset of indices k from J such that for some parse tree t which is $\sqsubseteq$-minimal in $\{t_j(l) | j \in J\}$, we have that k is the smallest index satisfying $t_k(l) = t$. We want to prove that K is finite, so suppose, to the contrary, that K is infinite, and consider the infinite sequence

$$\boldsymbol{u} = u_0 \cdots u_k \cdots = t_0 \cdots t_{K[0]-1} t_{K[0]}(l) \cdots t_{K[k']}(l) \cdots$$

By the definition of $\sqsubseteq$, we have that $t_{K[0]}(l) \sqsubseteq t_{K[0]}$, so $\boldsymbol{u}$ is less than $\boldsymbol{t}$ in the $\sqsubseteq$-lexicographic ordering, but $\boldsymbol{t}$ was a minimal counterexample to the lemma, so $\boldsymbol{u}$ is not a counterexample to the lemma. Thus, for some i, j where $i < j$, we have $u_i \sqsubseteq u_j$.

$\underline{i, j < K[0]}$: Then $u_i = t_i$ and $u_j = t_j$, so $t_i \sqsubseteq t_j$ follows from $u_i \sqsubseteq u_j$, contradicting that $\boldsymbol{t}$ is a counterexample to the lemma.

$\underline{i < K[0] \leq j}$: Set $j' := K[j - K[0]]$. Then $t_i = u_i \sqsubseteq u_j = t_{j'}(l)$. By the definition of $\sqsubseteq$ we have $t_{j'}(l) \sqsubseteq t_{j'}$ so by transitivity of $\sqsubseteq$, we get $t_i \sqsubseteq t_{j'}$. But $j' \geq j > i$ so $t_i \sqsubseteq t_{j'}$ contradicts that $\boldsymbol{t}$ is a counterexample to the lemma.

$\underline{K[0] \leq i, j}$: Set $i' := K[i - K[0]]$ and $j' := K[j - K[0]]$. Then $t_{i'}(l) = u_i \sqsubseteq u_j = t_{j'}(l)$, so by the definition of K, since $i' < j'$, we have that j' cannot be in K.

Thus K is finite.

Since K is finite and J infinite, we can select a $k \in K$ such that the set $J^k = \{j \in J | k < j, t_k(l) \sqsubseteq t_j(l)\}$ is infinite. Now, let k be the first element of L, i.e. set $L[0] = k$. We construct the rest of L by applying the above procedure recursively with with the infinite subset J^k of J in place of J.

We have now seen that for all $l \leq p$, if J is an infinite subset of I, there is an infinite subset L of J such that the infinite sequence $t_{L[0]}(l) \cdots t_{L[k]}(l) \cdots$ of parse trees is $\sqsubseteq$-ascending. In particular, this implies that there are infinite sets $L_1, \ldots, L_p$ of indices such that $I \supseteq L_1 \supseteq \cdots \supseteq L_p$, and such that for $l = 1, \ldots, p$, the sequence $t_{L_l[0]}(l) \cdots t_{L_l[k]}(l) \cdots$ is $\sqsubseteq$-ascending.

From the construction of L_p it follows that for $l = 0, \ldots, n$, the infinite sequence $t_{L_p[0]}(l) \cdots t_{L_p[k]}(l) \cdots$ is $\sqsubseteq$-ascending. Hence, by the definition of $\sqsubseteq$, we have that $t_{L_p[0]} \cdots t_{L_p[k]} \cdots$ is $\sqsubseteq$-ascending. Thus, for example, $t_{L_p[0]} \sqsubseteq t_{L_p[1]}$ contradicting that $\boldsymbol{t}$ is a counterexample to the lemma. □

Denote by $\mathcal{R}^{\preceq}$ the reflexive closure of $\mathcal{R}^{\prec}$. Thus $t \mathcal{R}^{\preceq} u$ means that t results from u by contraction of sub-parse trees with the root symbol as the only symbol in the frontier.

LEMMA A.4. *If two parse trees t and u are equivalent and $t \sqsubseteq u$, then $t \mathcal{R}^{\preceq} u$*

PROOF. First we observe an alternative but equivalent definition of $\sqsubseteq$, that for all parse trees t, u, we have $t \sqsubseteq u$ if and only if there is a sequence $t = t_0, \ldots, t_k = u$ of parse trees satisfying the following conditions:

(a) t_1 consists of t_0 rooted in some parse tree v_0 which might be trivial.

(b) For $i = 1, \ldots, k-1$, there are three parse trees u_i, v_i, w_i where v_i and w_i have common root symbol X_i, and where, moreover, X_i is the label of some leaf k_i of u_i and of some leaf l_i of v_i, and such that
—t_i consists of w_i rooted in k_i in u_i.
—t_{i+1} consists of w_i rooted in l_i in v_i which in turn is rooted in k_i in u_i.
Thus t_i is obtained from t_{i+1} by contraction of v_i.

Assume that $t_0 = t$ and $t_k = u$ are equivalent. We want to show that all the t_is are equivalent. This will imply the lemma, because then, for $i = 0, \ldots, k-1$, the root symbol of v_i would be the one and only symbol in the frontier of v_i, and hence we would have $t_i \mathcal{R}^{\preceq} t_{i+1}$. Thus, $t_0 \mathcal{R}^{\preceq} t_k$ would follow by transitivity.

For $i = 1, \ldots, k-1$. the parse trees t_i and t_{i+1} have the same root; namely the root of u_i. Moreover, t_0 and t_k have the same root since they are equivalent. Hence all of the t_is have the same root. For $i = 0, \ldots, k-1$, all the symbols in the frontier of t_i occur in in the frontier of t_{i+1} in the same order. Thus, since t_0 and t_k have the same frontier, we may conclude that all the t_is have the same frontier, and hence that they are equivalent. □

PROOF OF KEY LEMMA A.2. Putting Lemma A.3 and Lemma A.4 together, immediately we get the following result:

> *Given an infinite sequence $t_0 \cdots t_k \cdots$ of equivalent parse trees, there exist two numbers i, j such that $i < j$ and $t_i \mathcal{R}^{\preceq} t_j$.*

Now, let $\mathcal{E}$ be a set of parse tree pairs, and assume that $(\mathcal{E} \cup \mathcal{R})^{\prec}$ is irreflexive. The latter gives that $\mathcal{E}^{\prec}$ is a strict partial ordering. In order to establish the key lemma, we have to prove that $\mathcal{E}^{\prec}$ is well-founded.

Suppose that $\mathcal{E}^{\prec}$ is not well-founded. Then, by definition, there is then an infinite $\mathcal{E}^{\prec}$-descending sequence $t_0 \cdots t_k \cdots$ of parse trees. By the above result, for some numbers i, j, we have $i < j$ and $t_i \mathcal{R}^{\preceq} t_j$. Thus, either $t_i \mathcal{R}^{\prec} t_j$, or $t_i = t_j$; but by definition of the sequence we have $t_j \mathcal{E}^{\prec} t_i$, so both cases contradict that $(\mathcal{E} \cup \mathcal{R})^{\prec}$ is irreflexive. Thus $\mathcal{E}^{\prec}$ is well-founded, as desired. □

APPENDIX B. UNIVERSAL CLR(1) PARSERS

Recall that by *the universal CLR(1) parser of a grammar*, we refer to the generally non-deterministic LR(1) parser returned by the canonical LR(1) technique [Knuth 1965] when applied to the grammar. As usual, we assume that the technique has been modified to deal with derived productions. Essentially this parser $\mathcal{U}$ is defined by the following properties:

universality. There corresponds a unique parsing to any parse tree.

completability. Suppose we in a parsing arrive at a configuration of the form $(\alpha I, X\beta)$ and $a \in ACTIONS(I, X)$. Then for some β', from configuration $(\alpha I, X\beta')$, the parsing can be completed successfully starting with the action a.

no junk. For all states I there is an input symbol X such that $ACTIONS(I, X) \neq \emptyset$, and for all such X, there is a parsing encountering a configuration of the form $(\alpha I, X\beta)$.

minimality. It is the least general LR(1) parser satisfying the above conditions, i.e. if $\mathcal{U}'$ is another such parser, there is a surjection μ from the states in $\mathcal{U}'$ to the states in $\mathcal{U}$ such that for all states in I' in $\mathcal{U}'$ and for all input symbols X, if $I = \mu(I')$, we have $ACTIONS_{\mathcal{U}'}(I', X) = ACTIONS_{\mathcal{U}}(I, X)$ and $GOTO_{\mathcal{U}'}(I', X) = GOTO_{\mathcal{U}}(I, X)$.

We will now, in a rather condensed manner, recall the construction of the universal CLR(1) parser from Knuth [1965].

An *accept production* is an auxiliary production on the form $X' \rightarrow X$ where X is any grammar symbol. An *item* is a grammar production or an accept production with a '•' somewhere in the rightside. Let I be any set of pairs of items and input symbols. By $CLOSURE(I)$ we denote the closure of I under the condition that if $(X_1 \rightarrow \beta \bullet X\gamma, Y_1) \in CLOSURE(I)$ and either $Y \in FIRST(\gamma) \setminus \$$, or $Y = Y_1$ and $\$ \in FIRST(\gamma)$, then for all grammar productions $X \rightarrow \alpha$, we have $(X \rightarrow \bullet\, \alpha, Y) \in CLOSURE(I)$. Here

$$FIRST(\varepsilon) = \$$$
$$FIRST(X\alpha) = \{X\} \cup \bigcup \{FIRST(\beta\alpha) | X \rightarrow \beta \text{ is a production}\}.$$

Now, the *goto graph* of a grammar is a directed graph where the vertices are sets of pairs of items and input symbols, and where the arcs are labeled with grammar symbols. It is generated by the following conditions:

—For each grammar symbol X, there is a vertex $CLOSURE(\{X' \rightarrow \bullet\, X, \$\})$.
—If I is a vertex and $(X \rightarrow \alpha \bullet Y\beta, Z) \in I$, there is a vertex

$$J = CLOSURE(\{(X \rightarrow \alpha Y \bullet \beta, Z) | (X \rightarrow \alpha \bullet Y\beta, Z) \in I\}),$$

and an arc from I to J labeled Y.

The universal CLR(1) parser is constructed directly from the goto graph:

—The states are the set of vertices.
—For all grammar symbols X, the state $INIT(X)$ equals $CLOSURE(\{X' \rightarrow \bullet X, \$\})$.
—For all states I and grammar symbols X, we have $GOTO(I, X)$ defined if there is an arc labeled X from the state I to some state J, in which case $GOTO(I, X) = J$.
—Let I be a state and X an input symbol; then in $ACTIONS(I, X)$ we have the action
accept if $X = \$$ and for some Y we have $(Y' \rightarrow Y \bullet, \$) \in I$.
shift if for some Y, α, β, Z, we have $(Y \rightarrow \alpha \bullet X\beta, Z) \in I$.
reduce $Y \rightarrow \alpha$ if for some Y, α, we have $(Y \rightarrow \alpha \bullet, X) \in I$.

It follows from the analysis in Knuth [1965], that, indeed, this construction is always finite, and that it satisfies all the properties that was mentioned in the beginning of this appendix.

For grammar (4) page 14, we get the goto graph in Figure 1. The numbers at the bottom right corner of each vertex indicate the corresponding state numbers from $\mathcal{U}_{(4)}$ on page 14.

APPENDIX C. PARTIAL PARSING AND OTHER USEFUL NOTIONS

In this appendix we will introduce some notation and some concepts in connection with universal CLR(1) parsers that will facilitate more involved reasoning.

A first important change is that we consider the symbol shifted by the action shift as being part of the action. Thus the action shift on input X becomes shift X.

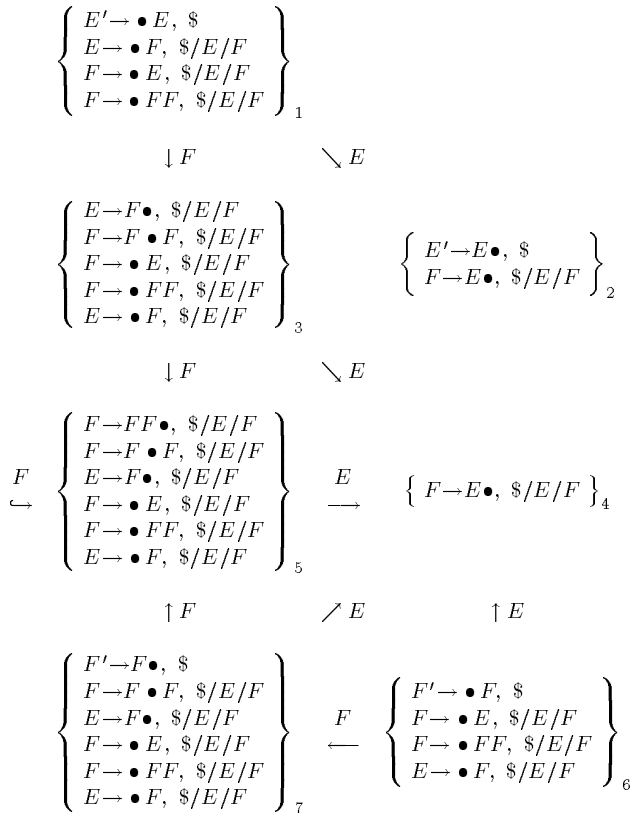
For notational convinience, by shift $X_1 \cdots X_n$ we denote the action sequence (shift $X_1 \cdots$ shift X_n), and by $GOTO(I, X_1 \cdots X_n)$ we denote $GOTO(\cdots GOTO(I, X_1) \cdots X_n)$.

Let t be a parse tree and i a number such that $i \leq |t|$, i.e. i is at most the number of non-ε nodes in t. Then state(t, i), input(t, i), and action(t, i) denote the state of, the input of, and the action from configuration i in the parsing corresponding to t. Notice that the first action is action$(t, 0)$. Also notice that we always have action$(t, |t|) =$ accept, for the $non - \varepsilon$ leaves of t correspond to shifts and the internal nodes of t correspond to the leftsides of reductions.

Now, the completability and no junk properties of the universal CLR(1) parser can be stated together as follows. Let I be a state, X an input symbol and a an action. Then $a \in ACTIONS(I, X)$ if and only if for some parse tree t and some $i \leq |t|$, we have $I =$ state(t, i), $X =$ input(t, i), and $a =$ action(t, i). Actually, this does not quite rule out superfluous states, so in order to get the full no junk property, we need to add that there is no state I such that $ACTIONS(I, X) = \emptyset$ for all input symbols X.

We need to be able to talk about *partial parsings*. A partial configuration consists of a suffix of a stack starting with a state. An initial partial configuration is a state

Fig. 1: Goto graph ($X \to \alpha \bullet \beta,\ X_0, / \cdots / X_n$ is a shorthand for $(X \to \alpha \bullet \beta, X_0), \ldots, (X \to \alpha \bullet \beta, X_n)$)



only. The effect of the actions on the partial configurations is as the effect on the stacks of the normal configurations. If we are in a partial configuration of the form $(st \boxed{I})$, then an action a is possible if for some input symbol X we have

$$a \in ACTIONS(I, X) \setminus \{\text{accept}\}$$

and, moreover, in the case of a reduction with a production $X \to \alpha$, the stack contains all of α.

Since we changed the action shift to include the symbol shifted, a partial parsing is uniquely characterized by the state of the initial partial configuration and the sequence of actions.

We say that *from state I it is possible to carry out the sequence of actions $\boldsymbol{a}$ arriving at state J* if there is a partial parsing with initial configuration $\boxed{I}$, sequence $\boldsymbol{a}$ of actions, and state J in the final configuration. In this case we denote J by $I^{\boldsymbol{a}}$.

If I is a state, $\boldsymbol{a}$ and $\boldsymbol{b}$ are action sequences, and $I^{\mathbf{ab}}$ is defined, then trivially $I^{\boldsymbol{a}}$ is defined, but $(I^{\boldsymbol{a}})^{\boldsymbol{b}}$ might be undefined, e.g. if $\mathbf{a} = \text{shift}\,\alpha$ and $\boldsymbol{b} = \text{reduce}\ X \to \alpha$.

Partial parsings have some important qualities. Suppose from state I that it is possible to carry out the sequence of actions $\boldsymbol{a}$ arriving at state J, and moreover that the action a is possible in state J on input X. From the completability and no junk

properties of universal CLR(1) parsers, it follows that there is a successful parsing that at some stage arrives at a configuration with state I, from there carries out the sequence $\boldsymbol{a}$ of actions arriving at state J with input X, and from there carries out the action a. Thus, for universal CLR(1) parsers, any partial parsing can be extended to a successful parsing. Notice that if our partial parsing starts in an initial state, then the extension can be chosen with the partial parsing as a prefix.

For the next appendix it is important to observe that if the action sequence $\boldsymbol{a}$ is a proper prefix of the action sequence for some parse tree with root symbol X, and, moreover, for the state I we have $I^{\mathrm{shift}X}$ defined, then $I^{\boldsymbol{a}}$ is defined.

APPENDIX D. SPANNING A UNIVERSAL PARSER ORDERING

The whole appendix is devoted to proving the following result solving problem 1, page 20:

THEOREM D.1. *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.*

Denote by $\mathcal{E}^S$ the closure under substitution of $\mathcal{E}$. Then $\mathcal{E}^{\prec}$ is the transitive closure of $\mathcal{E}^S$, and all universal orderings are transitive, so $\mathcal{E}^{\prec}$ and $\mathcal{E}^S$ span the same ordering of $\mathcal{U}$, if any. Now, for all parse tree pairs (t, u), let $P(t, u)$ denote the finite set of projections (cf. page 15) into $\mathcal{U}$ of the parse tree pairs in the typically infinite set $\{(t, u)\}^S$. Assuming that $P(t, u)$ is computable, we have the following trivial algorithm for the computation described in Theorem D.1

Algorithm A: Given $\mathcal{U}$ and $\mathcal{E}$, and assuming that $P(t, u)$ can be computed for all $(t, u) \in \mathcal{E}$, the algorithm constructs the ordering of $\mathcal{U}$ spanned by $\mathcal{E}$ if it is defined; otherwise it reports that no such ordering is spanned.

A.1.Set $P := \bigcup\{P(t, u) | (t, u) \in \mathcal{E}\}$.

A.2.If $\bot \in P$ then report "$\mathcal{E}$ does not span an ordering of $\mathcal{U}$".

A.3.For all states I and input symbols X:

A.3.1. Set $\mathcal{O}_{I,X} := \{(a, b) | ((I, X), a, b) \in P\}$.

A.3.2. Set $\mathcal{O}_{I,X} :=$ the transitive closure of $\mathcal{O}_{I,X}$.

A.3.3. If $\mathcal{O}_{I,X}$ is not irreflexive then report "$\mathcal{E}$ does not span an ordering of $\mathcal{U}$".

A.4.Return $\mathcal{O}$.

Thus, Theorem D.1 follows if we can compute $P(t, u)$ for all $(t, u) \in \mathcal{E}$. An algorithm doing this is stated formally below. We use the notation that $\overline{0} = 1$ and $\overline{1} = 0$. Also we use the following auxiliary function from pairs of sequences of grammar symbols and sets of input symbols to sets of input symbols:

$$\mathit{FirstL}(\alpha, L) = \begin{cases} \mathit{FIRST}(\alpha) & \text{if } \$ \notin \mathit{FIRST}(\alpha) \\ (\mathit{FIRST}(\alpha) \setminus \{\$\}) \cup\ L & \text{otherwise.} \end{cases}$$

Here *FIRST* is as defined on page A–4. The correctness of the algorithm will be argued after its formal statement.

Algorithm B: Given $\mathcal{U}$, the algorithm computes $P(t_0, t_1)$ for any parse tree pair (t_0, t_1), i.e. it computes the set of projections in $\mathcal{U}$ of all the parse tree pairs that follows from (t_0, t_1) by substitution.

B.1.If $t_0 = t_1$ then return $\{\perp\}$.

B.2.Set $P := \emptyset$.

B.3.Let $X \rightarrow^* X_1, \ldots, X_p$ be the derived production generated by t_0 and t_1.

B.4.Let the sequences of actions $\boldsymbol{a}^-, \boldsymbol{a}_0^+, \boldsymbol{a}_1^+$, and the actions a_0, a_1, where $a_0 \neq a_1$, be defined such that for $\imath = 0, 1$, the action sequence $\boldsymbol{a}^- a_\imath \boldsymbol{a}_\imath^+$(accept) is the action sequence of $t_\imath$. Thus $\boldsymbol{a}^-$ is the largest common prefix of the action sequences of t_0 and t_1.

B.5.Define k such that $X_1, \ldots, X_{k-1}$ are the symbols shifted by $\boldsymbol{a}^-$.

B.6.For all states I_0 such that shift$X$ is defined:

B.6.1. Set $I_1 := I_0^{\boldsymbol{a}^-}$.

B.6.2. Denote by L the set of possible inputs in state $I_0^{\text{shift}X}$.

B.6.3. If both a_0 and a_1 are reductions then:

B.6.3.1. For all input symbols $Y \in \mathit{FirstL}(X_k \cdots X_p, L)$, set $P := P \cup \{((I_1, Y), a_0, a_1)\}$.

B.6.4. If $a_\imath = \text{shift}X_k$, where either $\imath = 0$, or $\imath = 1$, then:

B.6.4.1. Denote by $\boldsymbol{c}$ the largest prefix of $a_{\bar{\imath}} \boldsymbol{a}_{\bar{\imath}}^+$ that does not contain any shifts.

B.6.4.2. For all non-empty prefix $\boldsymbol{d}$ of $\boldsymbol{c}$ such that in state $\mathit{INIT}(X_k)^{\boldsymbol{d}}$ it is possible to accept:

B.6.4.2.1. Denote by v the parse tree with action sequence $\boldsymbol{d}$(accept).

B.6.4.2.2. For $\imath = 0, 1$, denote by $t_\imath^v$ the result of rooting v in the kth non-ε leaf in $t_\imath$.

B.6.4.2.3. Set $P := P \cup P(t_0^v, t_1^v)$.

B.6.4.3. Construct the set S which is the closure of $\{(I_1, \mathit{INIT}(X_k))\}$ under the condition that if (I, J) belongs to S and $J^{\mathbf{c}}$ is defined then $(I^{\boldsymbol{c}}, J^{\boldsymbol{c}})$ belongs to S.

B.6.4.4. For all $I, J, \boldsymbol{d}$ such that $(I, J) \in S$ and $\boldsymbol{d}$ is a proper prefix of $\boldsymbol{c}$ with ${}_J{}^{\boldsymbol{d}}$ defined:

B.6.4.4.1. Denote by a the action in $\boldsymbol{c}$ following $\boldsymbol{d}$.

B.6.4.4.2. For all Y, b such that $Y \neq \$$, $b \notin \{a, \text{accept}\}$, and in state ${}_J{}^{\boldsymbol{d}}$ on input Y the action b is possible:

B.6.4.4.2.1. Set $a_\imath' := b$, and $a_{\bar{\imath}}' := a$.

B.6.4.4.2.2. Set $P := P \cup \{((I^{\boldsymbol{d}}, Y), a_0', a_1')\}$.

B.6.4.4.3. For all Y, b such that $Y \in \mathit{FirstL}(X_{k+1} \cdots X_p, L)$, $b \notin \{a, \text{accept}\}$, and in state ${}_J{}^{\boldsymbol{d}}$ on input \$ the action b is possible:

B.6.4.4.3.1. Set $a_\imath' := b$, and $a_{\bar{\imath}}' := a$.

B.6.4.4.3.2. Set $P := P \cup \{((I^{\boldsymbol{d}}, Y), a_0', a_1')\}$.

B.6.5. If $a_\imath = \text{accept}$, where either $\imath = 0$, or $\imath = 1$, then:

B.6.5.1. Denote by $\boldsymbol{c}$ the prefix of $a_{\bar{\imath}} \boldsymbol{a}_{\bar{\imath}}^+$ without the accept.

B.6.5.2. For all $\boldsymbol{d}$ such that $\boldsymbol{d}$ is a proper prefix of $\boldsymbol{c}$:

B.6.5.2.1. Denote by a the action in $\boldsymbol{c}$ following $\boldsymbol{d}$.

B.6.5.2.2. For all Y, b such that $Y \in L$, $b \neq a$, and b is possible in state $I_0^{(\text{shift}X)\boldsymbol{d}}$ on input Y:

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

B.6.5.2.2.2. Set $P := P \cup \{((I_0^{(\text{shift}X)\boldsymbol{d}}, Y), a_0', a_1')\}$.

B.7.Return P.

We will now settle Theorem D.1 by showing the correctness of Algorithm B.

LEMMA D.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 D.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 $\text{state}(t_0', |\boldsymbol{a}'^{-}|) = \text{state}(t_1', |\boldsymbol{a}'^{-}|)$ and $\text{input}(t_0', |\boldsymbol{a}'^{-}|) = \text{input}(t_1', |\boldsymbol{a}'^{-}|)$, and then

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

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 11) 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 $u_i = \underline{\text{pt}}(X_i)$.

Let the action sequences $\boldsymbol{b}^{-}, \boldsymbol{b}^{+}$ be defined such that $\boldsymbol{b}^{-}(\text{shift}X)\boldsymbol{b}^{+}$ 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 $\boldsymbol{a}_{\imath,0}(\text{shift}X_1)\boldsymbol{a}_{\imath,1}(\text{shift}X_2)\cdots(\text{shift}X_p)\boldsymbol{a}_{\imath,p}(\text{accept})$ is the action sequence of $t_\imath$, then

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

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 $\text{state}(u, |\boldsymbol{b}^{-}|)$. 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 state$(t'_0, |\boldsymbol{a}'^-|)$ $= I_0^{\boldsymbol{a}^-} = I_1$, and 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.

$a_\imath = \mathrm{shift}X_k$, 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 $t_\imath^v$ as defined in step B.6.4.2.2, it is clear that (t'_0, t'_1) follows from (t_0^v, t_1^v) by substitution. Recall from the proof of Lemma D.2 that the frontier of $t_\imath^v$ 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 action$(t'_\imath, |\boldsymbol{a}'^-|) = b$ and 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 $Y = \mathrm{input}(t'_0, |\boldsymbol{a}'^-|)$. Set $b = \mathrm{action}(u_k, |\boldsymbol{x}|)$. Clearly $b = \mathrm{action}(u_k, |\boldsymbol{x}|)$ is possible in state $J^{\boldsymbol{d}} = \mathit{INIT}(X_k)^{\boldsymbol{x}} = \mathrm{state}(u_k, |\boldsymbol{x}|)$ on input $Y = \mathrm{input}(u_k, |\boldsymbol{x}|)$. 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 $Y = \mathrm{input}(t'_0, |\boldsymbol{a}'^-|)$. 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 $X \rightarrow^* X$, 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 D.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. Using the completability and no-junk properties of universal CLR(1) parsers, it is straightforward, but tidious, to verify that the proof of Lemma D.3 can be reversed, so that for any projection p, we find parse trees $u, u_k, \cdots, u_p$ giving us a parse tree pair (t'_0, t'_1) which projects to p. □

PROOF OF THEOREM D.1. By Lemmas D.2, D.3, and D.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 with grammar (4), for the parse tree pairs in $\mathcal{E}_{(4)}$ we get

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

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

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

APPENDIX E. TRACING THE MINIMAL NON-CANONICAL PARSE TREES

This appendix addresses problem 2, page 20. As mentioned in Section 8, we will not solve the problem exactly as stated. However, when we come to Appendix H, where the concrete description of the algorithm generalizing the canonical LR(1) technique is given, it will be clear that the results of this appendix give a sufficient substitution for an exact solution to problem 2.

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 $\underline{\text{pt}}(X)$ 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 symbol X for which the trivial parse tree $\underline{\text{pt}}(X)$ 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 appendix is to show the following result which is our substitution for an exact solution to problem 2:

THEOREM E.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 Appendix B. 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 $I_0 = INIT(X_{n-1})^{\text{shift}\, \delta_{n-1} \cdots \delta_0}$. 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 E.1:

PROPOSITION E.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 E.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 $(s_0)(X_0 \rightarrow \alpha_0 \bullet \beta_0)(s_1, G_1, B_1) \cdots (X_{n-1} \rightarrow \alpha_{n-1} \bullet \beta_{n-1})(s_n, G_n, B_n)$. Then the action sequence $\boldsymbol{a}$ of t is of the from $(\text{shift}\,\delta)\,\overline{\text{a}}(X_0 \rightarrow \alpha_0 \bullet \beta_0) \cdots \overline{\text{a}}(X_{n-1} \rightarrow \alpha_{n-1} \bullet \beta_{n-1})\,(\text{accept})$.

Notice that the trace action $X_i \rightarrow \alpha_i \bullet \beta_i$ 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 $\overline{\text{a}}(X_i \rightarrow \bullet\, \beta_i)$ 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 $\boldsymbol{a}^0 = (\text{shift}\,\delta)\,(\text{shift}\,X_0)\,\overline{\text{a}}(X_1 \rightarrow \alpha_1 \bullet \beta_1) \cdots \overline{\text{a}}(X_{n-1} \rightarrow \alpha_{n-1} \bullet \beta_{n-1})\,(\text{accept})$ 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 $X_i \rightarrow \alpha_i\beta_i$ 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 settle (a) by showing that the reduction with $X_0 \rightarrow \beta_0$ 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 $\boldsymbol{a}^0 = (\text{shift}\,\delta)\,(\text{shift}\,X_0)\,\overline{\text{a}}(X_1 \rightarrow \alpha_1 \bullet \beta_1) \cdots \overline{\text{a}}(X_{n-1} \rightarrow \alpha_{n-1} \bullet \beta_{n-1})\,(\text{accept})$ 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 $X_i \rightarrow \alpha_i\beta_i$ 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 $\boldsymbol{a}^k = (\text{shift}\,\delta)\,\overline{\text{a}}(X_0 \rightarrow \alpha_0 \bullet \beta_0) \cdots \overline{\text{a}}(X_{k-1} \rightarrow \alpha_{k-1} \bullet \beta_{k-1})\,(\text{shift}\,X_k)\,\overline{\text{a}}(X_{k+1} \rightarrow \alpha_{k+1} \bullet \beta_{k+1}) \cdots \overline{\text{a}}(X_{n-1} \rightarrow \alpha_{n-1} \bullet \beta_{n-1})\,(\text{accept})$ 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 E.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 $\boldsymbol{a}_0\,\overline{\text{a}}(X_x \rightarrow \bullet \beta_x)\,\boldsymbol{a}_1\,\overline{\text{a}}(X_y \rightarrow \bullet \beta_y)\,\boldsymbol{a}_2$. 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 $\overline{\text{a}}(X_x \rightarrow \bullet\, \beta_x)$ 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 $\boldsymbol{a}_0\,\overline{\text{a}}(X_x \rightarrow \bullet\, \beta_x)$, for otherwise $\boldsymbol{a}^x$ would be non-canonical like $\boldsymbol{a}$. However, $\boldsymbol{a}_0\,\overline{\text{a}}(X_x \rightarrow \bullet\, \beta_x)$ cannot contain a non-canonical shift, for then $\boldsymbol{a}^y$ would be non-canonical like $\boldsymbol{a}$. Thus, $\boldsymbol{a}_0\,\overline{\text{a}}(X_x \rightarrow \bullet\, \beta_x)$ contains a non-canonical reduction. Clearly, replacing $\overline{\text{a}}(X_y \rightarrow \bullet\, \beta_y)$ 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 consecutive reductions in $\boldsymbol{a}$, if β is the sequence of symbols shifted between them, the second reduction is with a production $X \rightarrow \alpha\beta$, 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 $(\text{shift}\,\delta)\,\overline{\text{a}}(X_0 \rightarrow \alpha_0 \bullet \beta_0) \cdots \overline{\text{a}}(X_{n-1} \rightarrow \alpha_{n-1} \bullet \beta_{n-1})\,(\text{accept})$. 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

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

constructing t. Here s_0 is the stack consisting of the state $INIT(X)^{\text{shift}\,\delta}$, and for

$i > 0$, we have that s_i is a suffix of the stack of the parsing starting in $INIT(X_{n-1})$ after having carried out the action sequence $(\text{shift}\,\delta)\,\overline{\text{a}}(X_0 \rightarrow \alpha_0 \bullet \beta_0) \cdots \overline{\text{a}}(X_{i-1} \rightarrow \alpha_{i-1} \bullet \beta_{i-1})$. 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 E.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 $\overline{\text{a}}(X_k \rightarrow \bullet\, \beta_k)$, contradicting our claim. Hence condition (iv) holds. □

LEMMA E.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 contradicts 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 E.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_1. 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 E.3 and Lemma E.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 E.5. Thus t^r is the only non-trivial parse tree descending from a son of the root of t. By Lemma E.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 E.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:

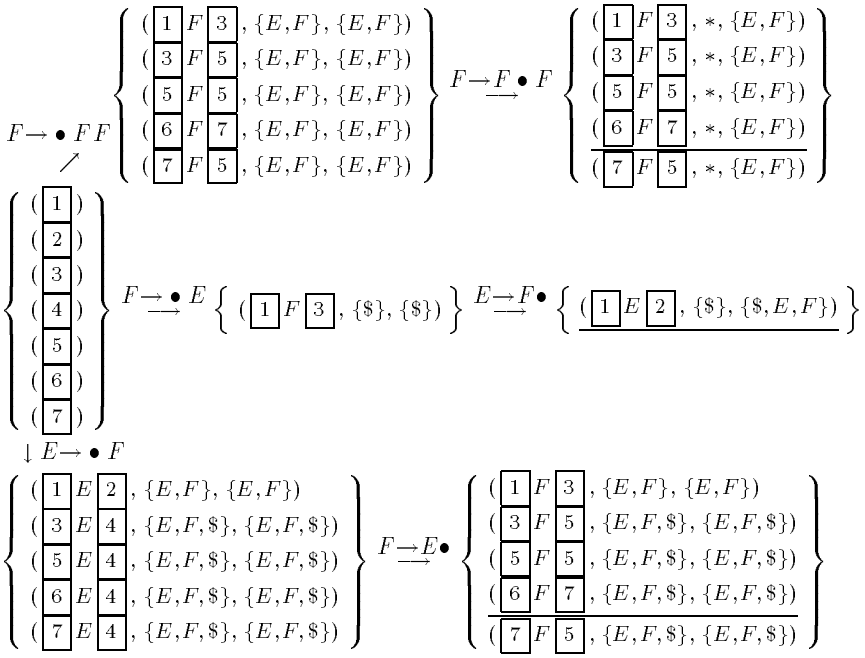
—The vertices are sets of trace configurations, and the arcs are labeled with trace actions.

—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.

Fig. 2. Trace graph



OBSERVATION E.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 E.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 E.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 E.2 together with Observation E.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 the determinization $\mathcal{P}_{(4)}$ of the universal CLR(1) parser $\mathcal{U}_{(4)}$ (cf. page 14) is depicted in Figure 2. However, as a kind of optimization, we have excluded all trace configurations with $G = \emptyset$ since these cannot lead to a successful trace. Notice, that it represents exactly the second coordinates of our prioritizing set $\mathcal{E}_{(4)}$, page 16.

APPENDIX F. LOOP FREE PARSING

This appendix addresses problem 3, page 20. 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 F.1. *Let $\mathcal{P}$ denote an $(\mathcal{R}^{\prec})^{\mathcal{U}}$-minimal determinization of a universal CLR(1) parser $\mathcal{U}$. Consider the ε-protected parsing with $\mathcal{P}$ on input (X, α). The parsing always terminates, 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 F.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 $X \to \varepsilon$. 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 F.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 $X \to \alpha$ is $O(|\alpha|)$.*

The $O(|\alpha|)$ bound is not needed here where we are only interested in termination. However, we shall use it in the next appendix to prove linearity of the generated parser.

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 at most $kn \in O(n)$, where k is the number of grammar symbols. Only shifts and reductions of empty strings increase the size of the stack. As there are at most $O(n)$ of these operations, there is at most $O(n)$ operations decreasing the size of the stack. Thus, we may conclude that there is at most $O(n)$ reductions with productions $X \to X_1 \cdots X_p$ where $p \neq 1$.

We complete the proof by showing that there cannot be k consecutive reductions of single symbols. Any such sequence would contain a sub-sequence $\boldsymbol{r} =$ reduce $X_k \to X_0$, reduce $X_{k-1} \to X_k, \ldots,$ reduce $X_0 \to X_1$. Suppose, for a contradiction, that we have a canonical parsing starting with the action sequence $\boldsymbol{ar}$. By the definition of universal CLR(1) parsers, there is a completion $\boldsymbol{b}$ such that $\boldsymbol{arb}$ is the action sequence of some parse tree t. Then $\boldsymbol{ab}$ is the action sequence of an

equivalent parse tree t^-. Clearly, we encounter a non-canonical action earlier in $\boldsymbol{ab}$ than in $\boldsymbol{arb}$, and $\mathcal{P}$ is $(\mathcal{R}^\prec)^{\mathcal{U}}$-minimal, so $(t^-, t) \notin (\mathcal{R}^\prec)^{\mathcal{U}}$. On the other hand, t^- is achieved from t by contraction of a parse tree with X_0 as both root and frontier, so by definition $(t^-, t) \in \mathcal{R}^\prec \subseteq (\mathcal{R}^\prec)^{\mathcal{U}}$. This completes the proof. □

APPENDIX G. THE RESOLVED PARSER

In this appendix we solve problem 4, page 21.

THEOREM G.1. *Given a universal CLR(1) parser, let $\mathcal{P}$ be a complete stable $(\mathcal{R}^\prec)^{\mathcal{U}}$-minimal determinization. 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 F.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. Notice that in the case of a successful parsing with $\mathcal{P}$, the corresponding parse tree is stable canonical, so the result follows directly from Proposition F.2.

Now consider an arbitrary incomplete parsing with input $X \rightarrow^? \alpha$. Suppose that we have carried out the actions sequence $\boldsymbol{a}$ shifting α_0, and that we have $\alpha_1\$$ left in the input buffer. Then $\alpha_0\alpha_1 = \alpha$. By the completability of universal CLR(1) parsers, there is a string α_1' that starts with the same symbol as α_1, and such that the parsing can be completed successfully with $\alpha_1'\$$ in the input buffer. Thus $X \rightarrow \alpha_0\alpha_1'$ is a derived production. Moreover, $\mathcal{P}$ is complete, so a parsing of $X \rightarrow^? \alpha_0\alpha_1'$ will be successful, and hence ε-loop free. But this parsing starts with the canonical action sequence $\boldsymbol{a}$, which is therefore itself ε-loop free. □

APPENDIX H. 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 $\mathcal{E}_{out} := \{(t,u) | u \in M^- \text{ and } t \text{ is the canonical equivalent of } u\}$.
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 appendix.

LEMMA H.1. *All steps in Algorithm C are implementable.*

PROOF. Theorem D.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 E.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 F.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 Theorem 8.1, 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 H.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 G.1, we have that $\mathcal{P}$ is a linear time parser. □

LEMMA H.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 F.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 5.1 for the CLR(1) technique. The result 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 5.1 for the canonical LR(1) technique. Similar constructive proofs can be given for all the other LL and LR techniques. In the next appendix we will go through some examples illustrating Algorithm C, and in Appendix J, there will be a short discussion of the efficiency of the algorithm.

APPENDIX I. 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 relating them to Algorithm C. First we will consider grammar (4) that we have used as an example throughout the paper, and afterwards, we will discuss how the algorithm reports failure for the dangling else grammar from Section 7. For simplicity, in both examples we will assume that $\mathcal{E}_{in} = \emptyset$.

Recall that grammar (4) has productions $E \rightarrow F$, $F \rightarrow E$, and $F \rightarrow FF$. As we saw in Appendix B, the grammar has the following universal CLR(1) parser:

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

INIT	State	*ACTIONS*			*GOTO*	
		E	F	\$	E	F
E	1	s	s	÷	2	3
	2	r $F \rightarrow E$	r $F \rightarrow E$	a, r $F \rightarrow E$	÷	÷
	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 $F \rightarrow E$	÷	÷
	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	÷	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 $\mathcal{R}$:

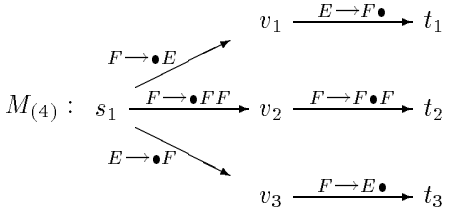
$$\mathcal{R}_{(4)}: \left\{ \begin{pmatrix} & E \\ E\ , & | \\ & F \\ & | \\ & E \end{pmatrix}\ ,\ \begin{pmatrix} & F \\ F\ , & | \\ & E \\ & | \\ & F \end{pmatrix} \right\}.$$

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

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

STATE	*ACTIONS*		
	E	F	\$
1	÷	÷	÷
2	÷	÷	a $\prec$ r $F \rightarrow E$
3	s $\prec$ r $E \rightarrow F$	s $\prec$ r $E \rightarrow F$	÷
4	÷	÷	÷
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	÷	÷	÷
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 5 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 Appendix E, we found the following trace graph representation of the set M:



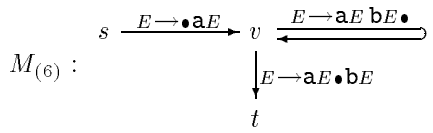
Now, as the set $\mathcal{E}_{out}$ constructed in steps C.3.2–C.3.4, we get

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

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 $\mathcal{E}^{\prec}_{(4)}$ spans a universal parser ordering in which $\mathcal{P}$ is minimal. From the example in Appendix D 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 dangling else grammar (6) with production set $\{E \rightarrow \mathtt{a}\,E\,\mathtt{b}\,E, E \rightarrow \mathtt{a}E\}$. In the universal CLR(1)

parser there will be a conflict on input b between shifting and reducing with $E \rightarrow \mathtt{a}\,E\,\mathtt{b}\,E$. If we choose to shift, we get the following trace graph representation for M:



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 $\mathcal{E}_{(6)}$, page 18. If instead we had chosen reduce $E \rightarrow \mathtt{a}\,E\,\mathtt{b}\,E$ over shift, we would have got a trace graph representing the first coordinates. In either case, we have to report failure.

APPENDIX J. SOME TECHNICAL REMARKS

In this appendix we will discuss efficiency and possible generalizations of our approach. So far, 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 8; namely to let the user works modularly, 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 our example with grammar (4), page 14, where we had the choice in state 5 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.

Unfortunately there are interesting grammars like those based on a dangling else for which the techniques presented in this paper will report failure. In that connection, it would be a significant improvement of our techniques to generalize them to deal with parsers $\mathcal{U}$-prioritized by infinite sets of parse tree pairs (cf. Definition 5.3). Indeed that would capture the dangling else grammar (6), page 18. The universal CLR(1) parser for grammar (6) contains only one conflict, which is between shifting and reducing. Let $\mathcal{P}_{(6)}$ be the determinization obtained by the standard resolution of the conflict in favor of shifting. As pointed out by Maddox [Maddox 1993], if e is any one of the parse tree pairs from $\mathcal{E}_{(6)}$, page 18, then setting $\mathcal{E}_{in} = \{e\}$ is sufficient for specifying $\mathcal{P}_{(6)}$ (cf. Section 6). Now, $\mathcal{P}_{(6)}$ is a complete linear time parser $\mathcal{U}$-prioritized by the infinite set $\mathcal{E}_{(6)}$, page 18, but how do we verify this automatically? At the moment we have a nice finite representation of $\mathcal{E}_{(6)}$: the second coordinates are succinctly represented by the three node trace graph $M_{(6)}$ that we found automatically at page A–25, and any one of the canonically equivelent first coordinates can be found in linear time using $\mathcal{P}_{(6)}$. The problems that remain are to verify automatically that all the infinitely many second coordinates really do have canonical equivalents, and to verify that all members of $\mathcal{E}_{(6)}$ project in the same way into the one and only conflict in $\mathcal{U}$.