Regional Least-Cost Error Repair

M. Vilares, V.M. Darriba, and F.J. Ribadas

Computer Science Department, Campus de Elviña s/n, 15071 A Coruña, Spain
vilares@udc.es, {darriba,ribadas}@dc.fi.udc.es

Abstract. We describe an algorithm to deal with automatic error repair over unrestricted context-free languages. The method relies on a regional least-cost repair strategy with validation, gathering all relevant information in the context of the error location. The system guarantees the asymptotic equivalence with global repair strategies.

1 Introduction

Until recently, errors were simply recovered by the consideration of fiducial symbols to provide mile-posts for error recovery, which allows a reduction in time and space bounds, although it is not always easy to determine if all relevant information to the error recovery process has been seen [3]. The significant reduction in cost processing has propitiated a renewable interest in methods that take into account the constraints on context. We can differentiate [3] two families of algorithms: one class, called *local repair*, make modifications to the input so that at least one more original input symbol can be accepted by the parser. The simplicity of these methods sometimes causes them to choose a poor repair [4, 1].

In contrast to local techniques, the *global repair* algorithms examine the entire program and make a minimum of changes to repair all the syntax errors, although they expend equal effort on all parts of the program, including areas that contain no errors. In between the local and global methods, Levi [2], suggested *regional repair* algorithms that fix a portion of the program including the error and as many additional symbols as needed to assure a good repair. In relation to global and local methods, the regional algorithms must answer the additional question of determining just how large a region to repair.

In addition, when several repairs are available, the system must provide some method of choosing among them, and a common strategy is to assign individual costs. A repair algorithm that guarantees finding the lowest-cost repair possible, is called a *least-cost* repair algorithm. Our proposal is a regional least-cost strategy which applies a dynamic validation in order to avoid cascaded errors.

2 A Dynamic Frame for Parsing

We introduce our parsing frame, as implemented in Ice [5]. Our aim is to parse sentences in the language $\mathcal{L}(\mathcal{G})$ generated by a context-free grammar

S. Yu and A. Păun (Eds.): CIAA 2000, LNCS 2088, pp. 293–301, 2001.

$\mathcal{G} = (N, \Sigma, P, S)$, where N is the set of non-terminals, Σ the set of terminal symbols, P the rules and S the start symbol. The empty string will be represented by ε.

2.1 The Operational Model

We assume that we produce a *push-down automaton* (PDA) from $\mathcal{G}$. In practice, we chose an LALR(1) device, which is possibly non-deterministic, for the language $\mathcal{L}(\mathcal{G})$. Formally, a PDA is a 7-tuple $\mathcal{A} = (\mathcal{Q}, \Sigma, \Delta, \delta, q_0, Z_0, \mathcal{Q}_f)$ where: $\mathcal{Q}$ is the set of states, Σ the set of input symbols, Δ the set of stack symbols, q_0 the initial state, Z_0 the initial stack symbol, $\mathcal{Q}_f$ the set of final states, and δ a finite set of transitions of the form $p\ X\ a \mapsto q\ Y$ with $p, q \in \mathcal{Q}$, $a \in \Sigma \cup \{\varepsilon\}$ and $X, Y \in \Delta \cup \{\varepsilon\}$. Let the PDA be in a configuration $(p, X\alpha, ax)$, where p is the current state, $X\alpha$ is the stack contents with X on the top, ax is the remaining input where the symbol a is the next to be shifted, $x \in \Sigma^*$. The application of $p\ X\ a \mapsto q\ Y$ results in a new configuration $(q, Y\alpha, x)$ where the terminal symbol a has been scanned, X has been popped, and Y has been pushed.

The algorithm proceeds by building a collection of *items*, compact representations of the recognizer stacks, by applying transitions to existing ones, until no new application is possible. The algorithm associates a set of items S^w_i, called *itemset*, for each input symbol w_i at the position i in the input string of length n, $w_{1..n}$. An item is of the form $[p, X, S^w_j, S^w_i]$, where $p \in \mathcal{Q}$, $X \in \Delta$, S^w_j is the *back pointer* to the itemset associated to the symbol w_i at which we began to look for that configuration of the automaton, and S^w_i is the current itemset.

2.2 The Recognizer

Formally, given a transition $\tau = \delta(p, X, a) \ni (q, Y)$, we translate it to items of the following form:

1. $\tilde{\delta}([p, X, S^w_j, S^w_i], a) \ni [q, \varepsilon, S^w_i, S^w_i]$, **if** $Y = X$
2. $\tilde{\delta}([p, X, S^w_j, S^w_i], a) \ni [p, Y, S^w_i, S^w_{i+1}]$, **if** $Y = a$
3. $\tilde{\delta}([p, X, S^w_j, S^w_i], a) \ni [p, Y, S^w_i, S^w_i]$, **if** $Y \in N$
4. $\tilde{\delta}([p, \varepsilon, S^w_j, S^w_i], a) \ni \tilde{\delta}_d([q, \varepsilon, S^w_l, S^w_j], a) \ni [q, \varepsilon, S^w_l, S^w_i]$, **if** $Y = \varepsilon$
   $\forall q \in \mathcal{Q}$ such that $\exists\ \delta(q, X, \varepsilon) \ni (p, X)$

with $\tilde{\delta} : \mathrm{It} \times \Sigma \cup \{\varepsilon\} \longrightarrow \{\mathrm{It} \cup \tilde{\delta}_d\}$ and $\tilde{\delta}_d : \mathrm{It} \times \Sigma \cup \{\varepsilon\} \longrightarrow \mathrm{It}$, where *It* is the set of all items developed in the parsing process and $\tilde{\delta}_d$ is called the set of *dynamic transitions.* Succinctly, we can describe the preceding cases as follows:

1. A goto action from the state p to state q under transition X.
2. A push of a from state p. The new item belongs to itemset S^w_{i+1}.
3. A push of non-terminal Y from state p.
4. A pop action from state p, where q is an ancestor of state p under transition X. We generate a *dynamic transition* $\tilde{\tau}_d$ to treat the absence of information about the rest of the stack. This transition is applicable not only to the

configuration resulting from the first one, but also on those to be generated and sharing the same syntactic structure.

Fairness and completeness of the dynamic construction are guaranteed by an equitable selection order *It*. To ignore redundant items we use a subsumption relation based on equality. Authors prove in [5] that time and space bounds are, in the worst case, $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2)$ respectivley, for inputs $w_{1..n}$.

3 Regional Least-Cost Error Repair

Following Mauney and Fischer in [3], we talk about the *error* in a portion of the input to mean the difference between what was intended and what actually appears in the input. So, we can talk about the *point of error* as the point at which the difference occurs. The *point of detection* is the point at which the parser detects that there is an error in the input and calls the repair algorithm.

Definition 1. *Let $w_{1..n}$ be an input string, we say that w_i is a* point of error *iff:* $\nexists\ [p, \varepsilon, S^w_l, S^w_i] \in S^w_i / \delta(p, X, w_i) = (q, w_i)$

The point of error is easily fixed by the parser itself and, in order to locate the origin of the error at minimal cost, we should try to limit the impact on the parse, focusing on the context of subtrees close to the point of error.

Definition 2. *Let w_i be a point of error for the input string $w_{1..n}$, we define the set of* points of detection associated to *w_i, as follows:*

$$detection(w_i) = \{w_{i'} / \exists A \in N,\ A \overset{+}{\Rightarrow} w_{i'} \alpha w_i\}$$

and we say that $A \overset{+}{\Rightarrow} w_{i'} \alpha w_i$ is a derivation defining the point of detection *$w_{i'} \in detection(w_i)$.*

Intuitively, the error is located in the immediate left parse context, represented by the closest viable node, or in the immediate right context, represented by the lookahead. However, sometimes can be usefull to isolate the parse branch in which the error appears.

Definition 3. *Let w_i be a point of error for $w_{1..n}$, we say that $[p, X, S^w_l, S^w_i] \in S^w_i$ is an* error item *iff: $\exists\ a \in \Sigma,\ \delta(p, \varepsilon, a) \neq \emptyset$, and we say that $[p, \varepsilon, S^w_{i'}, S^w_{i'}] \in S^w_{i'}$ is a* detection item *associated to w_i iff $\exists\ a \in \Sigma, \delta(p, A, a) \neq \emptyset,\ A \in N$ defining w_i, such that:*

$$\begin{array}{ll}
\delta(q_1, \varepsilon, w_{i'}) \ni (q_1, B_2), & \delta(q_1, B_2, w_{i'}) \ni (q_2, \varepsilon) \\
\vdots & \vdots \\
\delta(q_{n-1}, \varepsilon, w_{i'}) \ni (q_{n-1}, B_n), & \delta(q_{n-1}, B_n, w_{i'}) \ni (q_n, \varepsilon) \\
\delta(q_n, \varepsilon, w_{i'}) \ni (q_n, w_{i'}), & B_i \overset{+}{\Rightarrow} \varepsilon,\ \forall i \in [1, n]
\end{array}$$

We talk about *error and detection items*, when they represent nodes including the recognition of points of error and detection, respectively. The condition for error items implies that no scan action is possible for token w_i. In the detection case, conditions look for items recognizing a point of detection on a parse branch including an error item in w_i, disregarding empty reductions which are not relevant for this purpose.

Definition 4. *A* modification M to a string *of length* n*,* $w_{1..n} = w_1 \ldots w_n$*, is a series of edit operations,* $E_1 \ldots E_n E_{n+1}$*, in which each* E_i *is applied to* w_i *and possibly consists of a series of insertions before* w_i*, replacements or deletion of* w_i*. The string resulting from the application of the modification* M *to the string* w *is written* $M(w)$*.*

We now restrict the notion of modification to focus on a given zone of the input string, introducing the concept of error repair in this space. Intuitively, we look for conditions that guarantee the ability to recover the parse from the error, at the same time as it allows us to isolate repair branches by using the concept of reduction. We are also interested in minimizing the structural impact in the parse tree, and finally in introducing the notion of scope as the lowest reduction summarizing the process at a point of detection.

Definition 5. *Let* x *be a valid prefix in* $\mathcal{L}(\mathcal{G})$*, and* $w \in \Sigma^*$*, such that* xw *is not a valid prefix in* $\mathcal{L}(\mathcal{G})$*. We define a* repair of w following x *as* $M(w)$*, so that:*

$$\exists A \in N \;\Bigg/\; \begin{array}{l} S \overset{+}{\Rightarrow} x_{1..i-1}A \overset{+}{\Rightarrow} x_{1..i-1}x_{i..m}M(w),\; i \leq m \\ B \overset{*}{\Rightarrow} \alpha A \beta,\; \forall B \overset{+}{\Rightarrow} x_{j..m}M(w),\; j < i \\ A \overset{*}{\Rightarrow} \gamma C \rho,\; \forall C \overset{+}{\Rightarrow} x_{i..m}M(w) \end{array}$$

We denote the set of repairs of w *following* x *by* repair(x, w)*, and* A *by* scope(M)*.*

However, the notion of *repair*(x, w) is not sufficient for our purposes, since our aim is to extend the error repair process to consider all possible points of detection proposed by the algorithm for a given point of error, which implies simultaneously considering different valid prefixes and repair zones.

Definition 6. *Let* $e \in \Sigma$ *be a point of error, we define the set of* repairs for e*, as repair*$(e) = \{xM(w) \in$ *repair*$(x, w)/w_1 \in$ *detection*$(e)\}$*, where* detection(e) *denotes the set of points of detection associated to* e*.*

We now need a mechanism to filter out undesirable repair processes, in order to reduce the computational charges. To do that, we should introduce comparison criteria to only select those repairs with minimal cost.

Definition 7. *For each* $a \in \Sigma$ *we assume the existence of positive insert,* $I(a)$*; delete,* $D(a)$*, and replace* $R(a)$ *costs*[1]*. The* cost of a modification[2] $M(w_{1..n})$ *is*

[1] if any edit operation is not applied, we assume its cost to be zero.

[2] we assume that delete and replace operations are exclusive for the same token.

given by $\mathrm{cost}(M(w_{1..n})) = \Sigma_{i=1}^{n}(\Sigma_{j\in J} I(w_i^j) + D(w_i) + R(w_i))$*. In particular,* $\Sigma_{j\in J} I(w_i^j)$ *means that several insertion hypotheses are possible before the token* w_i *is real.*

When several repairs are available on different points of detection, we need a condition to ensure that only those with the same minimal cost are considered, looking for the best repair quality.

Definition 8. *Let* $e \in \Sigma$ *be a point of error, we define the set of* regional repairs for e*, as follows:*

$$regional(e) = \{xM(w) \in repair(e) \Big/ \begin{array}{l} cost(M) \leq cost(M'),\ \forall M' \in repair(x,w) \\ cost(M) = \min_{L\in repair(e)}\{cost(L)\} \end{array} \}$$

It is also necessary to take into account the possibility of cascaded errors, that is, errors precipitated by a previous erroneous repair diagnostics. Previous to dealing with the problem, we need to establish the existing relationship between the regional repairs for a given point of error, and future points of error.

Definition 9. *Let* w_i, w_j *be points of error in an input string* $w_{1..n}$*, such that* $j > i$*. We define the set of* viable repairs for w_i in w_j*, as follows:*

$$viable(w_i, w_j) = \{xM(y) \in regional(w_i)/xM(y)\ldots w_j \ valid\ prefix\ for\ \mathcal{L}(\mathcal{G})\}$$

Intuitively, the repairs in $viable(w_i, w_j)$ are the only ones capable of ensuring the continuity of the parse in $w_{i..j}$ and, therefore, the only possible repairs at the origin of the phenomenon of cascaded errors.

Definition 10. *Let* w_i *be an point of error for the input string* $w_{1..n}$*, we say that a point of error* w_j*,* $j > i$ *is a* point of error precipitated by w_i *iff*

$$\forall xM(y) \in viable(w_i, w_j),\ \exists A \in N \ defining\ w_{j'} \in detection(w_j)$$

such that $A \overset{+}{\Rightarrow} \beta scope(M)\ldots w_j$*.*

Intuitively, a point of error w_j is precipitated by the result of previous repairs on a point of error w_i, when all reductions defining points of detection for w_j summarize some viable repair for w_i in w_j.

4 The Algorithm

We propose that the repair be obtained by searching the PDA itself to find a suitable configuration to allow the parse to continue. At this point, our approach agrees with McKenzie *et al.* in [4], although this method is not asymptotically equivalent to a global repair strategy, and introduces an unsafe technique to speed up the repair algorithm [1]. In relation to this last, McKenzie *et al.* propose a pruning mechanism in order to reduce the number of configurations to be dealt

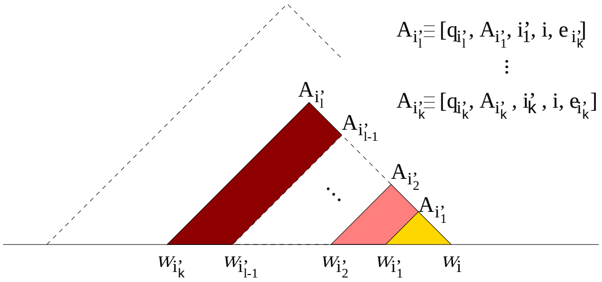
$A_{i'_l} \equiv [q_{i'_l}, A_{i'_1}, i'_1, i, e_{i'_k}]$
⋮
$A_{i'_k} \equiv [q_{i'_k}, A_{i'_k}, i'_k, i, e_{i'_k}]$
$A_{i'_l}$
$A_{i'_{l-1}}$
$A_{i'_2}$
$A_{i'_1}$
$w_{i'_k}$
$w_{i'_{l-1}}$
$w_{i'_2}$
$w_{i'_1}$
w_i

Fig. 1. Error detection

with during the repair process. This mechanism may lead to suboptimal regional repairs or may cause failure to produce any repair even if an error exists.

The problem due to pruning is based on a simple condition that ignores a stack configuration if an earlier one had the same stack top. The motivation is that this newer configuration would not lead to any cheaper repairs than the older one. Our dynamic programming construction eliminates this problem by considering all possible repair paths.

4.1 A Simple Case

We assume that we deal with the first error detected in the input string. The major features of the algorithm involve beginning with a list of error items, with an error counter zero. In order to compute the error counter, we extend the item structure: $[p, X, S_i^w, S_j^w, \text{e}]$, where now e is the error counter accumulated in the recognition of $X \in N \cup \Sigma$.

For each error item, we successively investigate the corresponding list of detection items, one for each parse branch including the error item. One a point of error w_i has been fixed, we can associate to it different points of detection $w_{i'_1}, \ldots w_{i'_k}$, as is shown in Fig. 1. Detection items are located by using the back pointer, that indicates the itemset where we have applied the last PDA action. So, we recursively go back into its ancestors until we find the first descendant of the last node that would have to be reduced if the lookahead was correct[3].

Once the detection items have been fixed for the corresponding error item, on each of the parse branches relying on them we apply all possible transitions beginning at the point of detection. These transitions correspond to four error hypotheses, from a given item:

- For scan transitions the item obtained is the same as for standard parsing.

$$[p, \varepsilon, S_j^w, S_i^w, 0] \overset{\text{scan}_{w_i}}{\longrightarrow} [p, w_i, S_i^w, S_{i+1}^w, 0]$$

[3] this information is directly obtained from the PDA.

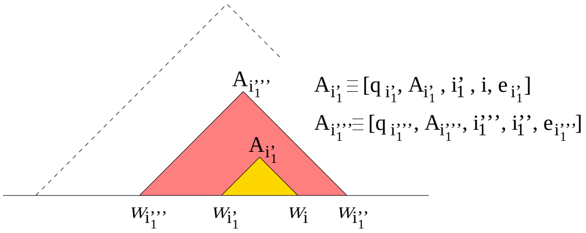
$A_{i'''_1}$
$A_{i'_1} \equiv [q_{i'_1}, A_{i'_1}, i'_1, i, e_{i'_1}]$
$A_{i'''_1} \equiv [q_{i'''_1}, A_{i'''_1}, i'''_1, i''_1, e_{i'''_1}]$
$A_{i'_1}$
$w_{i'''_1}$
$w_{i'_1}$
w_i
$w_{i''_1}$

Fig. 2. Repair scope

- In the case of insertion hypothesis, we initialize the error counter by taking into account the cost of the inserted token, which is included as stack symbol, and we add the new item to the same itemset, preserving the back pointer.

$$[p, \varepsilon, S^w_j, S^w_i, 0] \overset{\text{insert } a}{\longrightarrow} [p, a, S^w_i, S^w_i, I(a)],\ \delta(p, \varepsilon, a) \neq \emptyset$$

- For deletion hypothesis, we initialize the error counter by taking into account the cost of the deleted token and we add the new item to the next itemset, using the same stack symbol. The back pointer is initialized to the current itemset.

$$[p, \varepsilon, S^w_j, S^w_i, 0] \overset{\text{delete } w_i}{\longrightarrow} [p, \varepsilon, S^w_i, S^w_{i+1}, D(w_i)]$$

- Finally, for mutation hypothesis, we initialize the error counter by taking into account the cost of the replaced token and we add the new item to the next itemset. The back pointer is initialized to the current itemset, and the new token resulting from the mutation is included as stack symbol.

$$[p, \varepsilon, S^w_j, S^w_i, 0] \overset{\text{replace } w_i \text{ by } a}{\longrightarrow} [p, a, S^w_i, S^w_{i+1}, R(a)],\ \delta(p, \varepsilon, a) \neq \emptyset$$

We do that until a reduction verifying definition 5 covers both error and detection items accepting a token in the remaining input string, as is shown in Fig. 2, where $[w_{i'''_1}, w_{i''_1}]$ delimits the scope of a repair detected at the point $w_{i'_1} \in$ *detection*(w_i). Once we have applied the previous methodology to each detection item considered, we take only those repairs with regional lowest cost, applying definition 8. At this moment the parse goes back to standard mode.

We use a bottom-up strategy not only to parse the input, but also to compute error counters. This establishes a difference with McKenzie *et al.* [4], that uses a bottom-up parsing architecture with a top-down computation of the error counters. An inheritance strategy to compute the error counters, make the task of propagating these counters on shared parse branches a complex one. In our case, error counters are initialized at each error hypothesis and summarized only at reduce actions time. So, dynamic transitions must include information about the accumulated error counter in the part of the reduce action to be shared. The process is illustrated in Fig. 3 for two reductions, A_i and A_j, over a same rule $A \to X_1 \dots X_m$ sharing the last $X_{k+1} \dots X_m$ syntactic categories. We re-take the part of the error counter accumulated during the first reduction, $e_{i'_{k+1}} + \dots + e_{i'_m}$, for these common categories.

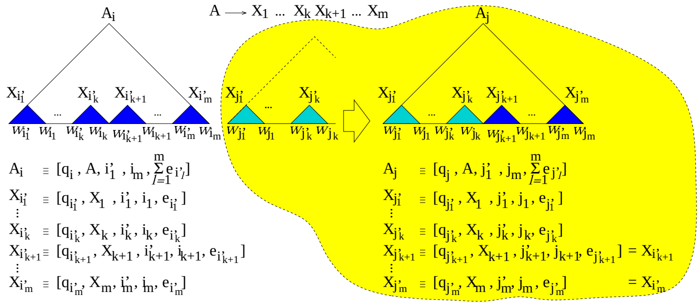
A → X1 ... Xk Xk+1 ... Xm
Ai
Aj
Ai ≡ [qi , A, i'1 , im , Σ e i'l]
Xi'1 ≡ [qi'1 , X1 , i'1 , i1 , ei'1]
Xi'k ≡ [qi'k , Xk , i'k , ik , ei'k]
Xi'k+1 ≡ [qi'k+1 , Xk+1 , i'k+1 , ik+1 , ei'k+1]
Xi'm ≡ [qi'm , Xm , i'm , im , ei'm]
Aj ≡ [qj , A, j'1 , jm , Σ e j'l]
Xj'1 ≡ [qj'1 , X1 , j'1 , j1 , ej'1]
Xj'k ≡ [qj'k , Xk , j'k , jk , ej'k]
Xj'k+1 ≡ [qj'k+1 , Xk+1 , j'k+1 , jk+1 , ej'k+1] = Xi'k+1
Xj'm ≡ [qj'm , Xm , j'm , jm , ej'm] = Xi'm

Fig. 3. Dynamic transitions in repair mode

4.2 The General Case

We now assume that the current repair process is not the first one and, therefore, can modify a previously repaired string. This arises when we realize that we come back to a detection item for which any parse branch includes a previous repair process. This process is illustrated in Fig. 4 for a point of error w_j precipitated by w_i, showing how the variable $A_{j'_1}$ defining w_j summarizes $A_{i'''_1}$, the scope of a previous repair defined by $A_{i'_1}$.

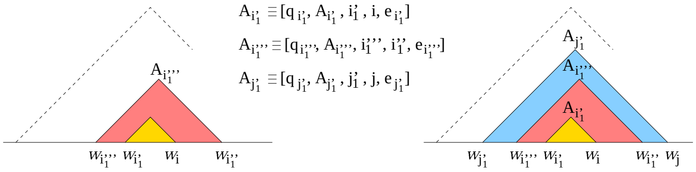
Ai'1 ≡ [q i'1 , Ai'1 , i'1 , i, e i'1]
Ai'''1 ≡ [q i'''1 , Ai'''1 , i'''1 , i''1 , e i'''1]
Aj'1 ≡ [q j'1 , Aj'1 , j'1 , j, e j'1]
Ai'''1
Aj'1
Ai'''1
Ai'1
wi'''1 wi'1 wi wi''1
wj'1 wi'''1 wi'1 wi wi''1 wj

Fig. 4. Dealing with precipitated errors

To deal with precipitated errors, the algorithm re-takes the previous error counters, adding the cost of the new error repair hypothesis to profit from the experience gained from previous repair processes. At this point, regional repairs have two important properties. First, it is independent of the shift-reduce parsing algorithm used. The second property is a consequence of the lemma below.

Lemma 1. *(The Expansion Lemma) Let w_i, w_j be points of error in $w_{1..n} \in \Sigma^*$, such that w_j is precipitated by w_i, then*

$$min\{j'/w_{j'} \in detection(w_j)\} < min\{i'/w_{i'} = y_1,\ xM(y) \in viable(w_i, w_j)\}$$

Proof. Let $w_{i'} \in \Sigma$, such that $w_{i'} = y_1$, $xM(y) \in viable(w_i, w_j)$ be a point of detection for w_i, for which some parsing branch derived from a repair in $regional(w_i)$ has successfully arrived at w_j.

Let w_j be a point of error precipitated by $xM(y) \in \mathit{viable}(w_i, w_j)$. By definition, we can assure that

$$\exists B \in N / B \overset{+}{\Rightarrow} w_{j'} \alpha w_j \overset{+}{\Rightarrow} \beta \text{scope}(M) \ldots w_j \overset{+}{\Rightarrow} \beta x_{l..m} M(y) \ldots w_j, \; w_{i'} = y_1$$

Given that $\mathit{scope}(M)$ is the lowest variable summarizing $w_{i'}$, it immediately follows that $j' < i'$, and we conclude the proof by extending the proof to all repairs in $\mathit{viable}(w_i, w_j)$. □

Corollary 1. *Let w_i, w_j be points of error in $w_{1..n} \in \Sigma^*$, such that w_j is precipitated by w_i, then*

$$\mathit{max}\{\text{scope}(M), \; M \in \mathit{viable}(w_i, w_j)\} \subset \mathit{max}\{\text{scope}(\tilde{M}), \; \tilde{M} \in \mathit{regional}(w_j)\}$$

Proof. It immediately follows from lemma 1. □

This allow us to get an asymptotic behavior close to global repair methods. This property has profound implications for the efficiency, measured by time and space taken, the simplicity and the power of computing regional repairs.

Corollary 2. *Let $w_{1..n}$ be an input string with a point of error in w_i, $i \in [1, n]$, then the time and space bounds for the regional repair algorithm are $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2)$, in the worst case, respectively.*

Proof. It immediately follows from the previous corollary 1. □

5 Conclusions

To improve the quality of repairs we should gather information to the right and to the left of the point of detection as long as this information could possibly be relevant. A criterion that meets our requirements is to expand the repair mode until it is guaranteed to accept the next input symbol, but maintains the chance of reconsidering the process once the system has detected that an incorrect repair assumption has been made.

References

1. Eberhard Bertsch and Mark-Jan Nederhof. On failure of the pruning technique in "Error Repair in Shift-Reduce Parsers". *ACM Transactions on Programming Languages and Systems*, 21(1):1–10, January 1999.
2. J.P. Levy. Automatic correction of syntax-errors in programming languages. *Acta Informatica*, 4:271–292, 1975.
3. J. Mauney and C.N. Fischer. Determining the extend of lookahead in syntactic error repair. *ACM Transactions on Programming Languages and Systems*, 10(3):456–469, 1988.
4. Bruce J. McKenzie, Corey Yeatman, and Lorraine De Vere. Error repair in shift-reduce parsers. *ACM Transactions on Programming Languages and Systems*, 17(4):672–689, July 1995.
5. M. Vilares and B.A. Dion. Efficient incremental parsing for context-free languages. In *Proc. of the* 5th *IEEE International Conference on Computer Languages*, pages 241–252, Toulouse, France, 1994.