A general method for transforming standard parsers into error-repair parsers⋆

Carlos Gómez-Rodríguez[1], Miguel A. Alonso[1], and Manuel Vilares[2]

[1] Departamento de Computación, Universidade da Coruña (Spain)
{cgomezr, alonso}@udc.es
[2] Escuela Superior de Ingeniería Informática, Universidade de Vigo (Spain)
vilares@uvigo.es

Abstract. A desirable property for any system dealing with unrestricted natural language text is robustness, the ability to analyze any input regardless of its grammaticality. In this paper we present a novel, general transformation technique to automatically obtain robust, error-repair parsers from standard non-robust parsers. The resulting error-repair parsing schema is guaranteed to be correct when our method is applied to a correct parsing schema verifying certain conditions that are weak enough to be fulfilled by a wide variety of parsers used in natural language processing.

1 Introduction

In real-life domains, it is common to find natural language sentences that cannot be parsed by grammar-driven parsers, due to insufficient coverage (the input is well-formed, but the grammar cannot recognize it) or ill-formedness of the input (errors in the sentence or errors caused by input methods). A standard parser will fail to return an analysis in these cases. A *robust parser* is one that can provide useful results for such extragrammatical sentences.

The methods that have been proposed to achieve robustness in parsing fall mainly into two broad categories: those that try to parse well-formed fragments of the input when a parse for the complete sentence cannot be found (partial parsers, such as that described in [6]) and those which try to assign a complete parse to the input sentence by relaxing grammatical constraints, such as *error-repair parsers*, which can find a complete parse tree for sentences not covered by the grammar by supposing that ungrammatical strings are corrupted versions of valid strings.

The problem of repairing and recovering from syntax errors during parsing has received much attention in the past (see for example the list of references provided in the annotated bibliography of [5, section 18.2.7]) and recent years (see for example [15, 17, 2, 7, 1, 11]). In this paper, we try to fill the gap between standard and error-repair parsing by proposing a transformation for automatically obtaining error-repair parsers, in the form of *error-repair parsing schemata*, from standard parsers defined as *parsing schemata*.[3]

⋆ Partially supported by Ministerio de Educación y Ciencia and FEDER (HUM2007-66607-C04) and Xunta de Galicia (PGIDIT07SIN005206PR, INCITE08E1R104022ES, INCITE08ENA305025ES, INCITE08PXIB302179PR and Rede Galega de Procesamento da Linguaxe e Recuperación de Información)

[3] *Schemata* is the plural form of the singular noun *schema*.

2 Standard parsing schemata

Parsing schemata [13] provide a formal, simple and uniform way to describe, analyze and compare different parsing algorithms. The notion of a parsing schema comes from considering parsing as a deduction process which generates intermediate results called *items*. An initial set of items is obtaineddirectly from the input sentence, and the parsing process consists of the application of inference rules (*deduction steps*) which produce new items from existing ones. Each item contains a piece of information about the sentence's structure, and a successful parsing process will produce at least one *final item* containing a full parse tree for the sentence or guaranteeing its existence.

When working with a context-free grammar[4] $G = (N, \Sigma, P, S)$, items are sets of trees from a set denoted $Trees(G)$, defined as the set of finitely branching finite trees in which children of a node have a left-to-right ordering, every node is labelled with a symbol from $N \cup \Sigma \cup (\Sigma \times \mathbb{N}) \cup \{\epsilon\}$, and every node u satisfies one of the following conditions:

- u is a leaf,
- u is labelled A, the children of u are labelled $X_1, \ldots, X_n$ and there is a production $A \rightarrow X_1 \ldots X_n \in P$,
- u is labelled A, u has one child labelled ϵ and there is a production $A \rightarrow \epsilon \in P$,
- u is labelled a and u has a single child labelled (a, j) for some j.

The pairs (a, j) will be referred to as *marked terminals*, and when we deal with a string $a_1 \ldots a_n$, we will usually write $\underline{a}_j$ as an abbreviated notation for (a_j, j) in the remainder of this paper. The natural number j is used to indicate the position of the word a in the input.

Valid parses for a string are represented by items containing complete *marked parse trees* for that string. Given a grammar G, a marked parse tree for a string $a_1 \ldots a_n$ is any tree $\tau \in Trees(G)$ such that $root(\tau) = S$ and $yield(\tau) = \underline{a}_1 \ldots \underline{a}_n$, where $root(\tau)$ refers to the root node of τ and $yield(\tau)$ refers to the frontier nodes of τ. An item containing such a tree for some arbitrary string is called a *final item*. An item containing such a tree for a particular string $a_1 \ldots a_n$ is called a *correct final item* for that string.

For each input string, a parsing schema's deduction steps allow us to infer a set of items, called *valid items* for that string. A parsing schema is said to be *sound* if all valid final items it produces for any arbitrary string are correct for that string. A parsing schema is said to be *complete* if all correct final items are valid. A parsing schema which is both sound and complete is said to be *correct*. A correct parsing schema can be used to obtain a working implementation of a parser by using deductive parsing engines as the ones described in [12, 4] to obtain all valid final items.[5]

3 Error-repair parsing schemata

The parsing schemata formalism introduced in the previous section does not suffice to define error-repair parsers that can show a robust behaviour in the presence of errors. In

[4] Although in this paper we will focus on context-free grammars, both standard and error-repair parsing schemata can be defined analogously for other grammatical formalisms.

[5] An example of a correct parsing schema is the Earley parsing schema, which defines the parser described by [3]. A full definition and proof of correctness for this schema can be found at [14].

these parsers, we should obtain items containing “approximate parses” if an exact parse for the sentence does not exist. Approximate parses need not be members of $Trees(G)$, since they may correspond to ungrammatical sentences, but they should be *similar* to a member of $Trees(G)$. Formalizing the notion of “similarity” as a distance function, we can obtain a definition of items allowing approximate parses to be generated.

3.1 Defining error-repair parsing schemata

Given a context-free grammar $G = (N, \Sigma, P, S)$, we shall denote by $Trees'(G)$ the set of finitely branching finite trees in which children of a node have a left-to-right ordering and every node is labelled with a symbol from $N \cup \Sigma \cup (\Sigma \times \mathbb{N}) \cup \{\epsilon\}$. Note that $Trees(G) \subset Trees'(G)$.

Let $d : Trees'(G) \times Trees'(G) \rightarrow \mathbb{N} \cup \{\infty\}$ be a function verifying the usual distance axioms (strict positiveness, symmetry and triangle inequality).

We shall denote by $Trees_e(G)$ the set $\{t \in Trees'(G) \,|\, \exists t' \in Trees(G) : d(t, t') \leq e\}$, i.e., $Trees_e(G)$ is the set of trees that have distance e or less to some valid tree in the grammar. Note that, by the strict positiveness axiom, $Trees_0(G) = Trees(G)$.

Definition 1. *(approximate trees)*
We define the set of *approximate trees* for a grammar G and a tree distance function d as $ApTrees(G) = \{(t, e) \in (Trees'(G) \times \mathbb{N}) \,|\, t \in Trees_e(G)\}$. Therefore, an approximate tree is the pair formed by a tree and its distance to some tree in $Trees(G)$. □

This concept of approximate trees allows us to precisely define the problems that we want to solve with error-repair parsers. Given a grammar G, a distance function d and a sentence $a_1 \dots a_n$, the *approximate recognition problem* is to determine the minimal $e \in \mathbb{N}$ such that there exists an approximate tree $(t, e) \in ApTrees(G)$ where t is a marked parse tree for the sentence. We will call such an approximate tree an *approximate marked parse tree* for $a_1 \dots a_n$.

Similarly, the *approximate parsing* problem consists of finding the minimal $e \in \mathbb{N}$ such that there exists an approximate marked parse tree $(t, e) \in ApTrees(G)$ for the sentence, and finding all approximate marked parse trees for the sentence.

Definition 2. *(approximate item set)*
Given a grammar G and a distance function d, we define an *approximate item set* as a set $\mathcal{I}'$ such that
$\mathcal{I}' \subseteq ((\bigcup_{i=0}^{\infty} \Pi_i) \cup \{\emptyset\})$
where each Π_i is a partition of the set $\{(t, e) \in ApTrees(G) \,|\, e = i\}$. □

Each element of an approximate item set is a set of approximate trees, and will be called an *approximate item*. Note that the concept is defined in such a way that each approximate item contains approximate trees with a single value of the distance e. This concrete value of e is what we will call *parsing distance* of an item ι, or $dist(\iota)$:

Definition 3. *(parsing distance of an item)*
Let $\mathcal{I}' \subseteq ((\bigcup_{i=0}^{\infty} \Pi_i) \cup \{\emptyset\})$ be an approximate item set as defined above, and $\iota \in \mathcal{I}'$. The *parsing distance* associated to the nonempty approximate item ι, $dist(\iota)$, is defined by the (trivially unique) value $i \in \mathbb{N} \,|\, \iota \in \Pi_i$. In the case of the empty approximate item $\emptyset$, we will say that $dist(\emptyset) = \infty$. □

Definition 4. *(error-repair parsing schema)*
Let G be a context-free grammar, d a distance function, and $a_1 \dots a_n \in \Sigma^*$ a string. An *error-repair instantiated parsing system* is a triple $(\mathcal{I}', H, D)$ such that $\mathcal{I}'$ is an approximate item set with distance function d, H is a set of hypotheses such that $\{a_i(\underline{a_i})\} \in H$ for each $a_i, 1 \leq i \leq n$, and D is a set of deduction steps such that $D \subseteq \mathcal{P}_{fin}(H \cup \mathcal{I}') \times \mathcal{I}'$. An *error-repair uninstantiated parsing system* is a triple $(\mathcal{I}', \mathcal{K}, D)$ where $\mathcal{K}$ is a function such that $(\mathcal{I}', \mathcal{K}(a_1 \dots a_n), D)$ is an error-repair instantiated parsing system for each $a_1 \dots a_n \in \Sigma^*$ (in practice, we will always define this function as $\mathcal{K}(a_1 \dots a_n) = \{\{a_i(\underline{a_i})\} \mid 1 \leq i \leq n\} \cup \{\epsilon(\epsilon)\}$). Finally, an *error-repair parsing schema* for a class of grammars $\mathcal{CG}$ and a distance function d is a function that assigns an error-repair uninstantiated parsing system to each grammar $G \in \mathcal{CG}$. □

Definition 5. *(final items)*
The set of *final items* for a string of length n in an approximate item set is defined by

$\mathcal{F}(\mathcal{I}', n) = \{\iota \in \mathcal{I} \mid \exists (t, e) \in \iota : t$ is a marked parse tree for some string $a_1 \dots a_n \in \Sigma^\star\}$.

The set of *correct final items* for a string $a_1 \dots a_n$ in an approximate item set is defined by

$\mathcal{CF}(\mathcal{I}', a_1 \dots a_n) = \{\iota \in \mathcal{I} \mid \exists (t, e) \in \iota : t$ is a marked parse tree for $a_1 \dots a_n\}$.
□

The concepts of *valid items*, *soundness*, *completeness* and *correctness* are analogous to the standard parsing schemata case. Note that the *approximate recognition* and *approximate parsing* problems defined earlier for any string and grammar can be solved by obtaining the set of correct final items for that string whose associated distance is minimal. These items can be deduced by any correct error-repair parsing schema, since they are a subset of correct final items.

3.2 A distance function for repairs based on the edit distance

Let us suppose a generic scenario where we would like to repair errors according to edit distance. The edit distance or Levenshtein distance [8] between two strings is the minimum number of insertions, deletions or substitutions of a single terminal needed to transform eother of the strings into the other one. Given a string $a_1 \dots a_n$ containing errors, we would like our parsers to return an approximate parse based on the exact parse tree of one of the strings in $L(G)$ whose Levenshtein distance to $a_1 \dots a_n$ is minimal.

A suitable distance function d for this case is given by the number of tree transformations that we need to transform one tree into another, if the transformations that we allow are inserting, deleting or changing the label of marked terminal nodes in the frontier. Therefore, $d(t_1, t_2) = e$ if t_2 can be obtained from t_1 by performing e transformations on marked terminal nodes in t_1, and $d(t_1, t_2) = \infty$ otherwise.

4 An error-repair transformation

The error-repair parsing schemata formalism allows us to define a transformation to map correct parsing schemata to correct error-repair parsing schemata that can successfully obtain approximate parses minimizing the Levenshtein distance. This transformation has been formally defined, but for space reasons we cannot include here all the definitions required for a rigorous, formal description. Therefore, we will instead provide an informal explanation of how the technique works, and a sketch of the proof that correctness is preserved.

4.1 From standard parsers to error-repair parsers

Most standard, non-robust parsers work by using grammar rules to build trees and link them together to form larger trees, until a complete parse can be found. Our transformation will be based on generalising parser deduction steps to enable them to link approximate trees and still obtain correct results, and adding some standard steps that introduce error hypotheses into the item set, which will be élegantly integrated into parse trees by the generalized steps.

The particular strategy used by parsers to build and link trees obviously varies between algorithms but, in spite of this, we can usually find two kinds of deduction steps in parsing schemata: those which introduce a new tree into the parse from scratch, and those which link a set of trees to form a larger tree. We will call the former *predictive steps* and the latter *yield union steps*.

Predictive steps can be identified because the yield of the trees in their consequent item does not contain any marked terminal symbol, that is, they generate trees which are not linked to the input string. Examples of predictive steps are the Earley *Initter* and *Predictor* steps. Yield union steps can be identified because the sequence of marked terminals in the yield of the trees of their consequent item (which we call the *marked yield* of these items[6]) is the concatenation of the marked yields of one or more of their antecedents[7], and the trees in the consequent item are formed by linking trees in antecedent items. Examples of yield union steps are Earley *Completer* and *Scanner*, and all the steps in the CYK parsing schema.

If all the steps in a parsing schema are predictive steps or yield union steps, we will call it a *prediction-completion parsing schema*. Most of the parsing schemata which can be found in the literature for widely-used parsers are prediction-completion parsing schemata, which allows us to generalize their steps to deal with an approximate item set in an uniform way. First of all, we must define this approximate item set. In order to do this, we take into account that each item in the item set of a prediction-completion parsing schemata can be represented by a triplet (p, i, j), where p is some entity from a set E (the form of the elements in E is not relevant for the transformation), and i, j are the leftmost and rightmost limits of the marked yields of the trees in the item. More formally, there exists a surjective *representation function* $r : E \times \mathbb{N} \times \mathbb{N} \rightarrow \mathcal{I}$, such that $yield_m(t) = \underline{a}_{i+1} \dots \underline{a}_j$ for every $t \in r(p, i, j)$[8]. We will denote the item $r(p, i, j)$ by $[p, i, j]$, which is the notation commonly used to represent items in the literature. Taking

[6] In the sequel, we will use the notation $yield_m(t)$ to refer to the marked yield of a tree t, and $yield_m(\iota)$ to refer to the common marked yield of the trees in an item ι, which we will call marked yield of the item.

[7] Actually, predictive steps can also be seen as yield union steps where the marked yield of the consequent item is the concatenation of the marked yield of *zero* of their antecedents. From this perspective it is not necessary to define predictive steps, but the concept has been introduced for clarity.

[8] Formally speaking, the necessary condition for the existence of such a representation function is that the trees in each item all have the same marked yield, of the form $yield_m(t) =$

this into account, we can define an approximate item set $\mathcal{I}'$ as the set of approximate items of the form $[p, i, j, x]$, where an approximate tree $(t, x) \in ApTrees(G)$ belongs to $[p, i, j, x]$ iff $yield_m(t) = \underline{a}_{i+1} \dots \underline{a}_j$ and there exists a tree t' in an item of the form $[p, i', j']$ such that $d(t, t') = x$[9]. With this approximate item set defined, we can generalize the steps in the following way:

- Sets of predictive steps taking as antecedents items of the form $[p_1, i_1, j_1]$, $[p_2, i_2, j_2]$, $\dots [p_a, i_a, j_a]$ and producing as consequent an item of the form $[p_c, i_c, i_c]$ are generalized to sets of steps taking antecedents $[p_1, i_1, j_1, x_1]$, $[p_2, i_2, j_2, x_2]$, $\dots [p_a, i_a, j_a, x_a]$ and producing a consequent $[p_c, i_c, i_c, 0]$. This means that we let the distances associated to antecedents take any value, and we always generate consequents with 0 as associated distance (since items generated by predictive steps are not linked in any way to the input, they need not consider error hypotheses).
- Sets of yield union steps taking as antecedents items of the form $[p_1, i_0, i_1]$, $[p_2, i_1, i_2]$, $\dots [p_y, i_{y-1}, i_y]$, $[p'_1, k_1, l_1]$, $[p'_2, k_2, l_2] \dots [p'_a, k_a, l_a]$ and producing as consequent an item of the form $[p_c, i_0, i_y]$ are generalized to sets of steps taking antecedents $[p_1, i_0, i_1, x_1]$, $[p_2, i_1, i_2, x_2]$, $\dots [p_y, i_{y-1}, i_y, x_y]$, $[p'_1, k_1, l_1, x'_1]$, $[p'_2, k_2, l_2, x'_2] \dots [p'_a, k_a, l_a, x'_a]$ and producing as consequent an item of the form $[p_c, i_0, i_y, x_1 + x_2 + \dots + x_y]$. This means that we let the distances associated to antecedents take any value, and the distance associated to the consequent is the sum of the distances associated to the items used for building it. Therefore, we are propagating and adding the errors coming from all the trees used to build the consequent tree.

Now we know how to adapt deduction steps in standard parsing schemata to adequately handle distances between trees, but this is not enough to obtain a correct error-repair parsing schema: the generalized steps will be able to link approximate trees and propagate the errors associated to each of them, but they will not detect any errors in the string by themselves. In order to do this, we must add some steps to the schema to introduce error hypotheses, i.e., elementary approximate trees representing the presence of an error in the input. This can be done in a way totally independent from the starting parser, by adding always the same *error hypothesis steps*, which are the following:

1. $SubstitutionHyp : \frac{[a, i, i+1]}{[b, i, i+1, 1]} \; b \in \Sigma$

2. $DeletionHyp : \frac{[\epsilon, i, i]}{[b, i, i, 1]} \; b \in \Sigma$

3. $InsertionHyp : \frac{[a, i, i+1]}{[\epsilon, i, i+1, 1]}$

4. $CorrectHyp : \frac{[a, i, i+1]}{[a, i, i+1, 0]}$

5. $InsCombiner1 : \frac{\begin{array}{c}[\epsilon, 0, j, e_1]\\ [(a|\epsilon), j, k, e_2]\end{array}}{[(a|\epsilon), 0, k, e_1 + e_2]}$

6. $InsCombiner2 : \frac{\begin{array}{c}[(a|\epsilon), i, j, e_1]\\ [\epsilon, j, k, e_2]\end{array}}{[(a|\epsilon), i, k, e_1 + e_2]}$

7. $DistanceIncreaser : \frac{[p, i, j, e]}{[p, i, i, e+1]} \; p \in E$

$\underline{a}_{i+1} \dots \underline{a}_j$. In practice, this condition should always hold in well-formed prediction-completion parsing schemata.

[9] Note that $\mathcal{I}'$ verifies the definition of an approximate item set if, and only if, $d(t1, t2) = \infty$ for every $t_1 \in [p_1, i_1, j_1], t_2 \in [p_2, i_2, j_2]$ such that $p1 \neq p2$. That is, items associated to different entities should be at infinite distance. This is the case for known item sets (such as the Earley, CYK or Left-Corner item sets) and our distance function.

The first three steps generate the basic error hypotheses: $SubstitutionHyp$ produces trees of the form $b \rightarrow \underline{a}_{i+1}$ for each symbol a_{i+1} in the input string (input symbol) and each $b \in \Sigma$ (expected symbol), which correspond to the hypothesis that the symbol a_{i+1} that we find in the input string is the product of a substitution error, and should appear as b instead in order for the string to be grammatical. The $DeletionHyp$ step generates deletion hypothesis trees of the form $b \rightarrow \epsilon$, corresponding to assuming that the symbol b, which should be the $i+1$th symbol in the input, has been deleted. Finally, the $InsertionHyp$ infers trees of the form $\epsilon \rightarrow \underline{a}_{i+1}$ for each symbol a_{i+1} in the input string, corresponding to the hypothesis that the input symbol a_{i+1} is the product of an insertion error, and therefore should not be taken into account in the parse.

The $CorrectHyp$ step corresponds to the "no error" hypothesis, producing a tree representing the fact that there is no error in a given input symbol a_{i+1}.

The two $Combiner$ steps produce trees of the form $a_2(\epsilon(\underline{a}_1)a_2(\underline{a}_2))$ and $a_1(\underline{a}_1\epsilon(\underline{a}_2))$. If the first symbol in the input is the product of an insertion error, the corresponding hypothesis is combined with the hypothesis immediately to its right. Insertion hypotheses corresponding to symbols other than the first one are combined to the hypothesis immediately to their left.

The necessity of these two combiner steps comes from the fact that standard parsing schemata are not prepared to handle trees rooted at ϵ, as generated by the *InsertionHyp* step. The combiner steps allow trees rooted at ϵ to be attached to neighbouring trees rooted at a terminal. In this way, the steps obtained from generalising those in the standard schema can handle insertion hypotheses. An alternative is not using combiner steps, and instead imposing extra constraints on the schemata to be transformed so that they can handle trees rooted at ϵ. Any prediction-completion parsing schema can be adapted to met these extra constraints, but this adaptation makes the conversion somewhat more complex.

Finally, the $DistanceIncreaser$ step is useless from a practical standpoint, but we have to include it in our error-repair parser if we wish to guarantee its completeness. The reason is that completeness requires the parser to be able to generate every possible correct final item, not just those with minimal associated distance. In practice, only minimal final items are needed to solve the approximate parsing and recognition problems. As this step is never necessary to generate a minimal item, it can be omitted in practical implementations of parsers.

4.2 The transformation

Putting it all together, we can define the *error-repair transformation* of a prediction-completion parsing system $\mathbb{S}$ as the error-repair parsing system $\mathcal{R}(\mathbb{S})$ obtained by applying the following changes to it:

1. Add the $SubstitutionHyp$, $DeletionHyp$, $InsertionHyp$, $InsCombiner1$, $InsCombiner2$, $CorrectHyp$ and $DistanceIncreaser$ steps, as defined above, to the schema.
2. For every predictive step in the schema (steps producing an item with an empty yield), change the step to its generalization obtained (in practice) by setting the distance associated to each antecedent item A_i to an unbound variable e_i, and set the distance for the consequent item to zero. For example, the Earley step

$$EarleyPredictor: \frac{[A \to \alpha.B\beta, i, j]}{[B \to .\gamma, j, j]} \; B \to \gamma \in P$$

produces the step

$$TransformedEarleyPredictor: \frac{[A \to \alpha.B\beta, i, j, e]}{[B \to .\gamma, j, j, 0]} \; B \to \gamma \in P \quad .$$

3. For every yield union step in the schema (steps using items with yield limits (i_0, i_1), (i_1, i_2), . . ., (i_{a-1}, i_a) to produce an item with yield $(i_1 \dots i_a)$:
   - If the step requires an hypothesis $[a, i, i+1]$, then change all appearances of the index $i+1$ to a new unbound index j[10].
   - Set the distance for each antecedent item A_k with yield (i_{k-1}, i_k) to an unbound variable e_k, and set the distance for the consequent to $e_1 + e_2 + \dots + e_a$.
   - Set the distance for the rest of antecedent items, if there is any, to unbound variables d_j.

   For example, the Earley step

   $$EarleyCompleter: \frac{[A \to \alpha.B\beta, i, j] \qquad [B \to \gamma., j, k]}{[A \to \alpha B.\beta, i, k]}$$

   produces the step

   $$TransformedEarleyCompleter: \frac{[A \to \alpha.B\beta, i, j, e_1] \qquad [B \to \gamma., j, k, e_2]}{[A \to \alpha B.\beta, i, k, e_1 + e_2]}$$

   The Earley step

   $$EarleyScanner: \frac{[A \to \alpha.a\beta, i, j] \qquad [a, j, j+1]}{[A \to \alpha a.\beta, i, j+1]}$$

   produces the step:

   $$TransformedEarleyScanner: \frac{[A \to \alpha.a\beta, i, j, e_1] \qquad [a, j, k, e_2]}{[A \to \alpha a.\beta, i, k, e_1 + e_2]}$$

4.3 Correctness of the obtained parsers

The robust transformation function we have just described maps prediction-completion parsing systems to error-repair parsing systems. However, in order for this transformation to be useful, we need it to guarantee that the generated robust parsers will be correct under certain conditions. This is done by the following two theorems:

Let $d: Trees'(G) \times Trees'(G) \to \mathbb{N}$ be a distance function, and let $\mathbb{S} = (\mathcal{I}, \mathcal{K}, D)$ be a prediction-completion parsing system.

Theorem 1. If $(\mathcal{I}, \mathcal{K}, D)$ is sound, every deduction step d in a predictive step set $D_i \subseteq D$ has a nonempty consequent, and for every deduction step d in a yield union step set $D_i \subseteq D$ with antecedents $[p_1, i_0, i_1], [p_2, i_1, i_2], \dots, [p_m, i_{m-1}, i_m], [p'_1, j_1, k_1], [p'_2, j_2, k_2], \dots, [p'_n, j_n, k_n]$ and consequent $[p_c, i_0, i_m]$ there exists a function $C_d : Trees'(G)^n \to Trees'(G)^n$ (tree combination function) such that:

[10] Steps including hypotheses as antecedents are not strictly yield union steps according to the formal definition of yield union step that we have omitted due to lack of space. However, these steps can always be easily transformed to yield union steps by applying this transformation. Note that this change does not alter any of the significant properties of the original (standard) parsing schema, since items $[a, i, j]$ with $j \neq i + 1$ can never appear in the deduction process.

- If $(t_1, \dots, t_m)$ is a tuple of trees in $Trees(G)$ such that $t_w \in [p_w, i_{w-1}, i_w](1 \leq w \leq m)$, then $C_d(t_1, \dots, t_m) \in [p_c, i_0, i_m]$.
- If $(t_1, \dots, t_m)$ is a tuple of trees in $Trees(G)$ such that $t_w \in [p_w, i_{w-1}, i_w](1 \leq w \leq m)$, and $(t'_1, \dots, t'_m)$ is a tuple of contiguous yield trees such that $d(t'_w, t_w) = e_w (1 \leq i \leq m)$, then $d(C_d(t_1, \dots, t_m), C_d(t'_1, \dots, t'_m)) = \Sigma_{w=1}^{m} e_w$, and $C_d(t'_1, \dots, t'_m)$ is a contiguous yield tree with $yield_m(C_d(t'_1, \dots, t'_m)) = yield_m(t'_1) yield_m(t'_2) \dots yield_m(t'_m)$.

Then $\mathcal{R}(\mathcal{I}, \mathcal{K}, D)$ is sound.

Theorem 2. If $(\mathcal{I}, \mathcal{K}, D)$ is complete, then $\mathcal{R}(\mathcal{I}, \mathcal{K}, D)$ is complete.

Note that the condition about the existence of tree combination functions in theorem 1 is usually straightforward to verify. A yield union step set normally combines two partial parse trees in $Trees(G)$ in some way, producing a new partial parse tree in $Trees(G)$ covering a larger portion of the input string. In practice, the existence of a tree combination function just means that we can also combine in the same way trees that are not in $Trees(G)$, and that the obtained tree's minimal distance to a tree in $Trees(G)$ is the sum of those of the original trees. For example, in the case of the Earley $Completer$ step, it is easy to see the function that maps a pair of trees of the form $A(\alpha(...)B\beta)$ and $B(\gamma(...))$ to the combined tree $A(\alpha(...)B(\gamma(...))\beta)$ obtained by adding the children of B in the second tree as children of B in the first tree is a valid combination function.

4.4 Proving Correctness

For space reasons, we cannot show the full proofs for the theorems that ensure that our error-repair transformation preserves correctness. Thus, we will just provide a brief outline on how the proofs are done.

Proof of Theorem 1 (Preservation of the soundness) We define a correct item in an error-repair parsing system for a particular string $a_1 \dots a_n$ as an approximate item $[p, i, j, e]$ containing an approximate tree (t, e) such that t is a tree with $yield_m(t) = \underline{a}_{i+1} \dots \underline{a}_j$; and we prove that the robust transformation of a given schema $\mathbb{S}$ is sound (all valid final items are correct) by proving the stronger claim that all valid items are correct. To prove this, we show that if the antecedents of a deduction step are correct, then the consequent is also correct. If we call D' the set of deduction steps in $\mathcal{R}(\mathbb{S})$, this is proven by writing D' as an union of step sets, and proving it separately for each step set. In the particular case of the steps D'_i coming from yield union step sets D_i in the original schema, the combination function is used to obtain a tree allowing us to prove that the consequent is correct. In the rest of the cases, this tree is obtained directly.

Proof of Theorem 2 (Preservation of the completeness) The proof for this theorem uses the fact that any final item in the approximate item set associated to $\mathcal{R}(\mathbb{S})$ can be written as $[p, i, j, e]$ (formally, there is a surjective *error-repair representation function* r' such that any approximate item in the set can be written as $r'(p, i, j, e)$; we use this function to formally define the approximate item set associated to $\mathcal{R}(\mathbb{S})$). Therefore, proving completeness is equivalent to proving that every correct final item of the form $[p, i, j, e]$ is valid. This is proven by induction on the distance e.

The base case of this induction $(e = 0)$ is proven by mapping items with distance 0 to items from the item set $\mathcal{I}$ corresponding to the original non-error-repair parser, and using the fact that this original parser is complete.

For the induction step, we suppose that the proposition holds for a distance value e, and prove it for $e + 1$. In order to do this, we take an arbitrary correct final item with associated distance $e+1$ and prove that it is valid. This is done by taking an approximate tree $(t, e+1)$ from this item and using it to build an approximate tree (t', d) whose yield only differs from $yield(t)$ in a single substitution, insertion or deletion operation. For each of the three operations, we build an instantiated parsing system where the item containing (t', e) is correct. By the induction hypothesis, this item is also valid in that system, so each case of the induction step is reduced to proving that validity of the item containing (t', e) in the constructed system implies the validity of $(t, e + 1)$ in the original system. This is proven by building suitable mappings between the items of both systems so that deductions are preserved and the item associated to (t', e) is mapped to the one associated to $(t, e+1)$. These mappings are different for each case (substitution, insertion, deletion) of the proof.

4.5 Simplifying the resulting parsers

The error-repair parsers obtained by using our transformation are guaranteed to be correct if the original standard parsers meet some simple conditions, but the extra steps added by the transformation can make the semantics of the obtained parsers somewhat hard to understand. Moreover, the $SubstitutionHyp$ and $DeletionHyp$ steps would negatively affect performance if implemented directly in a deductive engine.

However, once we have the error-repair transformation of a parser, we can apply some standard simplifications to it in order to obtain a simpler, more efficient one which will generate the same items except for the error hypotheses. That is, we can bypass items of the form $[p, i, j, e]$. In order to do this, we remove the steps that generate this kind of items and, for each step requiring $[a, i, j, e]$ as an antecedent, we change this requirement to the set of hypotheses of the form $[b, i', j']$ needed to generate such an item from the error hypothesis steps.

With this simplification, the error-repair transformation for an Earley parser is as follows:

$$TransformedEarleyInitter : \frac{}{[S \rightarrow .\alpha, 0, 0, 0]} \; S \rightarrow \alpha \in P$$

$$TransformedEarleyPredictor : \frac{[A \rightarrow \alpha.B\beta, i, j, e]}{[B \rightarrow .\gamma, j, j, 0]} \; B \rightarrow \gamma \in P$$

$$TransformedEarleyCompleter : \frac{[A \rightarrow \alpha.B\beta, i, j, e_1] \qquad [B \rightarrow \gamma., j, k, e_2]}{[A \rightarrow \alpha B.\beta, i, k, e_1 + e_2]}$$

$$GeneralSubstitutionEarleyScanner : \frac{[A \rightarrow \alpha.a\beta, i, j, e]}{[A \rightarrow \alpha a.\beta, i, k, e + k - j]} \; k \geq j + 1$$

$$GeneralDeletionEarleyScanner : \frac{[A \rightarrow \alpha.a\beta, i, j, e]}{[A \rightarrow \alpha a.\beta, i, k, e + k - j + 1]} \; k \geq j$$

$$GeneralEarleyScanner1 : \frac{[A \to \alpha.a\beta, 0, 0, e] \qquad [a, w-1, w]}{[A \to \alpha a.\beta, 0, k, e+k-1]} \; 0 < w \leq k$$

$$GeneralEarleyScanner2 : \frac{[A \to \alpha.a\beta, i, j, e] \qquad [a, j, j+1]}{[A \to \alpha a.\beta, i, k, e+k-j-1]} \; k \geq j+1$$

where the steps named “scanner” are obtained from the Earley $Scanner$ step after applying the simplification in order to bypass items of the form $[p, i, j, e]$.

Note that, if we choose the alternative transformation that does not use $Combiner$ steps (see section 4.1), the obtained parser would be the one described in [9], which is a *step refinement* ([13]) of the parsing schema described in this section.

5 Empirical results

In order to test the results of our transformation in practice, we have used the system described in [4] to execute the error-repair version of the Earley parser explained above. We have executed the schema in two different modes: obtaining all the valid final items with minimal distance (*global error repair*); and restricting repair steps on errors to regions surrounding the errors in order to obtain a single optimal solution, as explained in [16] (*regional error repair*).

The grammar and sentences used for testing are from the DARPA ATIS3 system. Particularly, we have used the same test sentences that were used by [10]. This test set is suitable for error-repair parsing, since it comes from a real-life application and it contains ungrammatical sentences. In particular, 28 of the 98 sentences in the set are ungrammatical. By running our error-repair parsers, we find that the minimal edit distance to a grammatical sentence is 1 for 24 of them (i.e., these 24 sentences have a possible repair with a single error), 2 for two of them, and 3 for the remaining two.

Minimal Distance	# Sentences	Avg. Length	Avg. Items (Global)	Avg. Items (Regional)	Improvement
0	70	11.04	37,558	37,558	0%
1	24	11.63	194,249	63,751	65.33%
2	2	18.50	739,705	574,534	22.33%
3	2	14.50	1,117,123	965,137	13.61%
>3	none	n/a	n/a	n/a	n/a

Table 1. Performance results for the global and regional error-repair parsers when parsing sentences from the ATIS test set. Each row corresponds to a value of the minimal parsing distance (or error count).

Table 1 shows the average number of items generated by our parsers with respect to the minimal parsing distance of the inputs. As we can see, regional parsing reduces item generation by a factor of three in sentences with a single error. In sentences with more than one error the improvements are smaller. However, we should note that parsing time grows faster than the number of items generated, so these relative improvements in items translate to larger relative improvements in runtime. Moreover, in practical settings we can expect sentences with several errors to be less frequent than sentences with a single error, as in this case. Thus, regional error-repair parsers are a good practical alternative to global ones.

6 Conclusions and future work

We have presented a method to transform context-free grammar parsers (expressed as parsing schemata) into error-repair parsers. Our transformation guarantees that the resulting error-repair parsers are correct as long as the original parsers verify certain conditions. It is easy to see that popular grammar parsers such as CYK, $Earley$ or $Left-Corner$ verify these conditions, so this method can be applied to create a wide range of error-repair parsers. The transformation can be applied automatically, and its results are error-repair parsing schemata that can be executed by means of a deductive engine. Therefore, a system as the one described in [4] can be extended to automatically generate implementations of robust parsers from standard parsing schemata.

The method is general enough to be applied to different grammatical formalisms. Currently, we are applying it to parsers for tree adjoining grammars.

References

1. C. Cerecke. Repairing syntax errors in LR-based parsers. *Australian Computer Science Communications*, 24(1):17–22, 2002.
2. R. Corchuelo, J. A. Pérez, A. Ruiz and M. Toro. Repairing Syntax Errors in LR Parsers. *ACM Transactions on Programming Languages and Systems,* 24(6):698–710, 2002.
3. J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, 1970.
4. C. Gómez-Rodríguez, J. Vilares, and M. A. Alonso. A compiler for parsing schemata. *Software: Practice and Experience*, Forthcoming. DOI 10.1002/spe.904
5. D. Grune and C. J. H. Jacobs. *Parsing Techniques. A Practical Guide — Second edition*. Springer Science+Business Media, 2008.
6. W. Kasper, B. Kiefer, H. U. Krieger, C. J. Rupp, and K. L. Worm. Charting the depths of robust speech parsing. In *Proc. of ACL'99*, pages 405–412, Morristown, NJ, USA, 1999.
7. I.-S. Kim and K.-M. Choe. Error Repair with Validation in LR-based Parsing. *ACM Transactions on Programming Languages and Systems*, 23(4):451–471, 2001.
8. V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.
9. G. Lyon. Syntax-directed least-errors analysis for context-free languages: a practical approach. *Commun. ACM*, 17(1):3–14, 1974.
10. R. C. Moore. Improved left-corner chart parsing for large context-free grammars. In *Proc. of the 6th IWPT, pages 171–182, Trento, Italy*, pages 171–182, 2000.
11. J. C. Perez-Cortes, J. C. Amengual, J. Arlandis, and R. Llobet. Stochastic error-correcting parsing for OCR post-processing. In *ICPR '00: Proceedings of the International Conference on Pattern Recognition*, page 4405, Washington, DC, USA, 2000. IEEE Computer Society.
12. S. M. Shieber, Y. Schabes, and F. C. N. Pereira. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1–2):3–36, July-August 1995.
13. K. Sikkel. *Parsing Schemata — A Framework for Specification and Analysis of Parsing Algorithms*. Springer-Verlag, Berlin/Heidelberg/New York, 1997.
14. K. Sikkel. Parsing schemata and correctness of parsing algorithms. *Theoretical Computer Science,* 199(1–2):87–103, 1998.
15. P. van der Spek, N. Plat and N. Pronk. Syntax Error Repair for a Java-based Parser Generator. *ACM SIGPLAN Notices*, 40(4):47–50, 2005.
16. M. Vilares, V. M. Darriba, and F. J. Ribadas. Regional least-cost error repair. *Lecture Notes in Computer Science*, 2088:293–301, 2001.
17. M. Vilares, V. M. Darriba, J. Vilares and F.J. Ribadas. A formal frame for robust parsing. *Theoretical Computer Science*, 328:171–186, 2004.