Error-repair parsing schemata

Carlos Gómez-Rodríguez*[,a], Miguel A. Alonso[a], Manuel Vilares[b]

[a]*Departamento de Computación, Universidade da Coruña (Spain)*
[b]*Escola Superior de Enxeñería Informática, Universidade de Vigo (Spain)*

Abstract

Robustness, the ability to analyze any input regardless of its grammaticality, is a desirable property for any system dealing with unrestricted natural language text. Error-repair parsing approaches achieve robustness by considering ungrammatical sentences as corrupted versions of valid sentences. In this article we present a deductive formalism, based on Sikkel's parsing schemata, that can be used to define and relate error-repair parsers and study their formal properties, such as correctness. This formalism allows us to define a general transformation technique to automatically obtain robust, error-repair parsers from standard non-robust parsers. If our method is applied to a correct parsing schema verifying certain conditions, the resulting error-repair parsing schema is guaranteed to be correct. The required conditions are weak enough to be fulfilled by a wide variety of popular parsers used in natural language processing, such as CYK, Earley and Left-Corner.

Key words: Parsing, robust parsing, error repair, natural language processing

1. Introduction

When using grammar-driven parsers to process natural language texts in real-life domains, it is common to find sentences that cannot be parsed by the grammar. This may be due to several reasons, including insufficient coverage (the input is well-formed, but the grammar cannot recognize it) and ill-formedness of the input (errors in the

*Corresponding author. Address: Facultade de Informática, Campus de Elviña 5, 15071 A Coruña, Spain. Phone: +34 981 167000 (ext. 1302). Fax: +34 981 167160.
Email addresses: cgomezr@udc.es (Carlos Gómez-Rodríguez), alonso@udc.es (Miguel A. Alonso), vilares@uvigo.es (Manuel Vilares)

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 [1]) and those which try to assign a complete parse to the input sentence by relaxing grammatical constraints. In this article we will focus on error-repair parsers, which fall into the second category. An error-repair parser is a kind of robust parser that can find a complete parse tree for sentences not covered by the grammar, by supposing that ungrammatical strings are corrupted versions of valid strings.

In the field of compiler design for programming languages, the problem of repairing and recovering from syntax errors during parsing has received a great deal of attention in the past (see for example the list of references provided in the annotated bibliography of [2, section 18.2.7]) and also in recent years (see for example [3, 4, 5, 6]). In the field of natural language parsing, some error-repair parsers have also been described, for example, in [7, 8], or more recently in [9, 10].

However, no formalism has been proposed to uniformly describe error-repair parsers, compare them and prove their correctness. In this article we propose such a framework, and we use it to define a transformation for automatically obtaining error-repair parsers, in the form of *error-repair parsing schemata*, from standard parsers defined as *parsing schemata*.

This article may be outlined as follows. In Sect. 2, the framework of parsing schemata for standard parsers is introduced. It is then extended to define error-repair parsing schemata in Sect. 3. A general method to transform standard parsing schemata into error-repair ones is presented in Sect. 4. The formal properties of this transformation are analyzed in Sect. 5, and the proof of correctness is given in Sect. 6. Some techniques to optimize the error-repair parsing schemata resulting from this transformation are presented in Sect. 7, and final conclusions are presented in Sect. 8.

2. Parsing schemata

Parsing schemata [11] 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 obtained directly 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.

Let $G = (N, \Sigma, P, S)$ be a context-free grammar.[1] The set of valid trees for G, denoted $\mathit{Trees}(G)$, is defined by Sikkel [11] as the set of finitely branching finite trees in which

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

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 $X_1, \dots, X_n$ and there is a production $A \rightarrow X_1 \dots X_n \in P$, [2]
- 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 $a_1 \dots a_n$, we will usually write $\underline{a}_j$ as an abbreviated notation for (a_j, j) in the remainder of this article. The natural number j is used to indicate the position of the word a in the input, so that the input sentence $a_1 \dots a_n$ can be viewed as a set of trees of the form $a_j(\underline{a}_j)$ rather than as a string of symbols.

Let *Trees*(G) be the set of trees for some context-free grammar G. An *item set* is any set $\mathcal{I}$ such that $\mathcal{I} \subseteq \Pi \cup \{\emptyset\}$, where Π is a partition of the set *Trees*(G). Each of the elements of an item set is called an *item*. If the item set contains $\emptyset$ as an element, we call this element the *empty item*.

Valid parses for a string in the language defined by a grammar G are represented in an item set by items containing complete *marked parse trees* for that string. Given a grammar G, a marked parse tree for a string $a_1 \dots a_n$ is any tree $\tau \in$ *Trees*(G), whose root is labelled S, and such that *yield*$(\tau) = \underline{a}_1 \dots \underline{a}_n$. 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 $a_1 \dots a_n$ is called a *correct final item* for that string.

Example 1. The Earley item set [12], $\mathcal{I}_{Earley}$, associated to a context-free grammar $G = (N, \Sigma, P, S)$ is defined by:

$$\mathcal{I}_{Earley} = \{[A \rightarrow \alpha \bullet \beta, i, j] \mid A \rightarrow \alpha\beta \in P \wedge 0 \leq i \leq j\}$$

where our notation for items $[A \rightarrow \alpha \bullet \beta, i, j]$ is a shorthand notation for the set of trees rooted at A, such that the direct children of A are the symbols of the string $\alpha\beta$, the combined yields of the subtrees rooted at the symbols in α form a string of marked terminals of the form $\underline{a}_{i+1} \dots \underline{a}_j$, and the nodes labelled with the symbols in β are leaves.

The set of final items in this case is its subset $\mathcal{F}_{Earley} = \{[S \rightarrow \gamma \bullet, 0, n]\}$. ■

Example 2. The CYK item set [13, 14] for a context-free grammar $G = (N, \Sigma, P, S)$ is defined as follows:

$$\mathcal{I}_{CYK} = \{[A, i, j] \mid A \in N \wedge 0 \leq i < j\}$$

where each item $[A, i, j]$ is the set of all the trees in *Trees*(G) rooted at A whose yield is

[2]Throughout the article, we will use uppercase letters $A, B, \dots$ to represent nonterminal symbols and $X, Y, \dots$ for arbitrary symbols, lowercase letters $a, b, \dots$ for terminals, and Greek letters $\alpha, \beta, \dots$ for strings of terminals and nonterminals. We will use parenthetical notation to describe trees. The yield of a tree T, denoted *yield*(T), is the string obtained by concatenating the symbols at its leaf nodes from left to right.

of the form $\underline{a}_{i+1} \dots \underline{a}_j$.[3]

The set of final items is the set $\mathcal{F}_{CYK} = \{[A, 0, n]\}$. ■

Let G be a context-free grammar and $a_1 \dots a_n \in \Sigma^*$ a string. An *instantiated parsing system* is a triple $(\mathcal{I}, \mathcal{H}, D)$ such that $\mathcal{I}$ is an item set, $\{a_i(\underline{a_i})\} \in \mathcal{H}$ for each $a_i, 1 \leq i \leq n$, and $D \subseteq \mathcal{P}_{fin}(\mathcal{H} \cup \mathcal{I}) \times \mathcal{I}$.[4] Elements of $\mathcal{H}$ are called *initial items* or *hypotheses* of the parsing system, and elements of D are called *deduction steps.* For convenience, we denote each deduction step $(\{\eta_1, \dots, \eta_k\}, \xi)$ by $\eta_1, \dots, \eta_k \vdash \xi$. Deduction steps establish an inference relation $\vdash$ between items, so that $Y \vdash \xi$ if $(Y', \xi) \in D$ for some $Y' \subseteq Y$. We will also use $\vdash^*$ as a notation for multiple-step inferences. An *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 instantiated parsing system for each $a_1 \dots a_n \in \Sigma^*$.

An instantiated parsing system is said to be *sound* if all *valid* final items in it (i.e. all final items that can be deduced from its hypotheses by using its deduction steps) are correct for its associated string. A parsing system is said to be *complete* if all correct final items are valid (i.e, if there is a marked parse tree for a particular string, then the system can deduce it). A parsing system which is both sound and complete is said to be *correct.*

A *parsing schema* for a class of grammars $\mathcal{CG}$ is a function that allows us to obtain an uninstantiated parsing system for each grammar $G \in \mathcal{CG}$. A parsing schema is correct if all the parsing systems it generates for different grammars and strings are correct.

A correct parsing schema can be used to obtain a working implementation of a parser by using deductive parsing engines such as the ones described in [15, 16] to obtain all valid final items.

Example 3. A correct parsing schema is the Earley parsing schema, which defines the parser described in [12]. This schema maps each context-free grammar $G \in \mathcal{CFG}$ and string $a_1 \dots a_n \in \Sigma^*$ to an instantiated parsing system $(\mathcal{I}, \mathcal{H}, D)$ where:

$\mathcal{I} = \mathcal{I}_{Earley}$ (as defined before)

$\mathcal{H} = \{[a, i-1, i] \mid a = a_i \wedge 1 \leq i \leq n\}$[5]

$D^{Scanner} = \{[A \to \alpha \bullet x\beta, i, j], [x, j, j+1] \vdash [A \to \alpha x \bullet \beta, i, j+1]\}$

$D^{Completer} = \{[A \to \alpha \bullet B\beta, i, j], [B \to \gamma \bullet, j, k] \vdash [A \to \alpha B \bullet \beta, i, k]\}$

$D^{Predictor} = \{[A \to \alpha \bullet B\beta, i, j] \vdash [B \to \bullet\gamma, j, j]\}$

$D^{Initter} = \{\vdash [S \to \bullet\gamma, 0, 0]\}$

$D = D^{Initter} \cup D^{Scanner} \cup D^{Completer} \cup D^{Predictor}$. ■

Example 4. A parsing schema for the CYK parsing algorithm [13, 14] maps each context-free grammar G in Chomsky normal form (i.e. with all its rules of the form

[3]An item can be the empty item $\emptyset$ if it contains no trees. For example, an item $[A, 0, 2]$ in a CYK parser will be empty if our grammar G does not allow the construction of any trees with root labelled A and a yield of length 2.

[4]We use $\mathcal{P}_{fin}(X)$ to denote the finite power set of X.

[5]This standard set of hypotheses will be used for all the schemata described here, including those for error-repair parsers, so we will not make it explicit in subsequent schemata.

$A \to BC$ or $A \to a$) and string $a_1 \dots a_n \in \Sigma^*$ to the instantiated parsing system $(\mathcal{I}, \mathcal{H}, D)$ such that:

$$\mathcal{I} = \mathcal{I}_{CYK} \text{ (as defined before)}$$

$$D^{CYKUnary} = \{[a, i, i+1] \vdash [A, i, i+1] \mid A \to a \in P\}$$

$$D^{CYKBinary} = \{[B, i, j], [C, j, k] \vdash [A, i, k] \mid A \to BC \in P\}$$

$$D = D^{CYKUnary} \cup D^{CYKBinary}. \qquad \blacksquare$$

The proof of correctness for these two schemata can be found in [17].

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 behavior in the presence of errors: items can only contain members of *Trees*(G), which are trees that conform to the constraints imposed by the grammar, but in order to handle ungrammatical sentences we need to be able to violate these constraints. What we need is to 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). This notion of "similarity" can be formalized as a distance function in order to 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*$'(G) \to \mathbb{N} \cup \{\infty\}$ be a function verifying the axioms of an extended pseudometric ($d(x, x) = 0$ for all x, plus the well-known metric axioms of symmetry and triangle inequality)[6].

We shall denote by *Trees*$_e(G)$ the set $\{t \in Trees'(G) \mid \exists t' \in Trees(G) : d(t, t') \leq e\}$, i.e., *Trees*$_e(G)$ is the set of trees that have distance e or less to some grammatically valid tree. Note that, by construction, *Trees*$(G) \subseteq$ *Trees*$_0(G)$.

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

[6]An extended pseudometric is a generalization of a metric, where the word *extended* refers to the fact that we allow our distance to take the value ∞, and the prefix *pseudo* means that we allow the distance between two distinct trees to be zero, while a metric imposes the additional constraint that an entity can only be at distance 0 from itself.

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 of the form (t, e) for the sentence.

As we can see, while the problem of parsing is a problem of finding trees, the problem of approximate parsing can be seen as a problem of finding approximate trees. In the same way that the problem of parsing can be solved by a deduction system whose items are sets of trees, the problem of approximate parsing can be solved by one whose items are sets of approximate trees.

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 $\{(t, i) \in ApTrees(G)\}$.*

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. Directly defining an approximate item set using any partition of $ApTrees(G)$ would be impractical, since we need our parsers to keep track of the degree of discrepancy of partial parses with respect to the grammar, and that information would be lost if our items were not associated to a single value of e. This concrete value of e is what we will call the *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 $i \in \mathbb{N}$ such that $\iota \in \Pi_i$. In the case of the empty approximate item $\emptyset$, we will say that $dist(\emptyset) = \infty$.*

Having defined approximate item sets that can handle robust parsing by relaxing grammar constraints, error-repair parsers can be described by using parsing schemata that work with these items.

Definition 4. (error-repair parsing system, 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 $(\mathcal{I}', \mathcal{H}, D)$ such that $\mathcal{I}'$ is an approximate item set with distance function d, $\mathcal{H}$ is a set of hypotheses such that $\{a_i(\underline{a_i})\} \in \mathcal{H}$ for each $a_i, 1 \leq i \leq n$, and D is a set of deduction steps such that $D \subseteq \mathcal{P}_{fin}(\mathcal{H} \cup \mathcal{I}') \times \mathcal{I}'$.*

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 $\mathcal{K}(a_1 \dots a_n) = \{\{a_i(\underline{a_i})\} \mid 1 \leq i \leq n\}$).*

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, correct final items)
The set of final items *for strings 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 $a_1 \dots a_n\}$.*

Definition 6. (minimal parsing distance)
The minimal parsing distance *for a string $a_1 \dots a_n$ in an approximate item set $\mathcal{I}'$ is defined by $MinDist(\mathcal{I}', a_1 \dots a_n) = min\{e \in \mathbb{N} \mid \exists \iota \in \mathcal{CF}(\mathcal{I}', a_1 \dots a_n) : dist(\iota) = e\}$.*

Definition 7. (minimal final items)
The set of minimal final items *for a string $a_1 \dots a_n$ in an approximate item set $\mathcal{I}'$ is defined by*

$$\mathcal{MF}(\mathcal{I}', a_1 \dots a_n) = \{\iota \in \mathcal{CF}(\mathcal{I}', a_1 \dots a_n) \mid dist(\iota) = MinDist(\mathcal{I}', a_1 \dots a_n)\}$$

The concepts of *valid items*, *soundness*, *completeness* and *correctness* are totally analogous to the standard parsing schemata case. Note that the *approximate recognition* and *approximate parsing* problems that we defined earlier for any string and grammar can be solved by obtaining the set of minimal final items in an approximate item set. Minimal final 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 edit distance based repair

A correct error-repair parsing schema will obtain the approximate parses whose distance to an exact parse is minimal. Therefore, a suitable distance function should be chosen depending on the kind of errors that are more likely to appear in input sentences.

Let us suppose a generic scenario where we would like to repair errors according to edit distance. The edit distance or Levenshtein distance [18] between two strings is the minimum number of insertions, deletions or substitutions of a single terminal needed to transform either 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 grammatical strings whose Levenshtein distance to $a_1 \dots a_n$ is minimal.

A suitable distance function $\hat{d}$ for this case is given by ignoring the indexes in marked terminals (i.e. two trees differing only in the integer values associated to their marked terminals are considered equal for this definition) and defining the distance as the number of elementary tree transformations that we need to transform one tree into another, if the elementary transformations that we allow are inserting, deleting or changing the label of a marked terminal node in the frontier.

More formally, for each $t \in Trees'(G)$, we define $Insertion(t)$, $Deletion(t)$ and $Substitution(t)$ as the set of trees obtained by inserting a marked terminal node in the frontier, deleting a marked terminal node and changing its associated symbol, respectively. With this, we can define sets of transformations of a given tree t as follows:

$$\mathit{Trans}_0(t) = \{t\}$$

$$\mathit{Trans}_1(t) = \mathit{Insertion}(t) \cup \mathit{Deletion}(t) \cup \mathit{Substitution}(t)$$

$$\mathit{Trans}_i(t) = \{t' \in \mathit{Trees}'(G) \mid \exists u \in \mathit{Trans}_{i-1}(t) : t' \in \mathit{Trans}_1(u)\}, \text{ for } i > 1$$

and our distance function $\hat{d}$ as follows:

$$\hat{d} : \mathit{Trees}'(G) \times \mathit{Trees}'(G) \to \mathbb{N} \cup \{\infty\}$$

$$\hat{d}(t_1, t_2) = \mathit{min}\{i \in \mathbb{N} \mid t_2 \in \mathit{Trans}_i(t_1)\}, \text{ if } \exists i \in \mathbb{N} : t_2 \in \mathit{Trans}_i(t_1)$$

$$\hat{d}(t_1, t_2) = \infty \text{ otherwise}$$

Note that our distance function is symmetrical, since for every $t_1, t_2 \in \mathit{Trees}'(G)$, $t_1 \in \mathit{Trans}_i(t_2)$ if and only if $t_2 \in \mathit{Trans}_i(t_1)$. This is easy to verify if we take into account that $t_1 \in \mathit{Deletion}(t_2) \Leftrightarrow t_2 \in \mathit{Insertion}(t_1)$, $t_1 \in \mathit{Insertion}(t_2) \Leftrightarrow t_2 \in \mathit{Deletion}(t_1)$, and $t_1 \in \mathit{Substitution}(t_2) \Leftrightarrow t_2 \in \mathit{Substitution}(t_1)$. It is trivial to verify that the function $\hat{d}$ also satisfies the other pseudometric axioms.

If we call the string edit distance d_{ed}, then it is easy to see that for any tree t_1 such that $\mathit{yield}(t_1) = \alpha$, and for any string β, there exists a tree t_2 with yield β such that $\hat{d}(t_1, t_2) = d_{ed}(\alpha, \beta)$.

As we only allow transformations dealing with marked terminal nodes, trees that differ in nodes labelled with other symbols will be considered to be at infinite distance. Therefore, when we define a parser using this distance, the parses (t_2) obtained for an ungrammatical input sentence (β) will be identical, except for marked terminals, to the valid parses (t_1) corresponding to the grammatical sentences (α) whose distance to the input is minimal.

3.3. Lyon's global error-repair parser

The formalism of error-repair parsing schemata allows us to represent error-repair parsers in a simple, declarative way, making it easy to explore their formal properties and obtain efficient implementations of them. As an example, we will see how this formalism can be used to describe one of the most influential error-repair parsers in the literature: the one described by Lyon [7].

The schema for Lyon's error-repair parser maps each grammar $G \in \mathcal{CFG}$ to a triple $(\mathcal{I}', \mathcal{K}, D)$, where $\mathcal{K}$ has the standard definition explained in section 3.1, and $\mathcal{I}'$ and D are defined as follows:

$$\mathcal{I}'_{Lyon} = \{[A \to \alpha \bullet \beta, i, j, e] | A \to \alpha\beta \in P \wedge i, j, e \in \mathbb{N} \wedge i \leq j\}$$

where we use $[A \to \alpha \bullet \beta, i, j, e]$ as a shorthand notation for the set of approximate trees (t, e) such that t is a partial parse tree with root A where the direct children of A are labelled with the symbols of the string $\alpha\beta$, and the frontier nodes of the subtrees rooted at the symbols in α form the substring $\underline{a}_{i+1} \dots \underline{a}_j$ of the input string. The distance function d used to define the approximate item set, and therefore conditioning the values of e, is $\hat{d}$ as defined in section 3.2.

The set of deduction steps, D, for Lyon's parser is defined as the union of the following:

$D^{Initter} = \{\vdash [S \to \bullet\gamma, 0, 0, 0]\}$

$D^{Scanner} = \{[A \to \alpha \bullet x\beta, i, j, e], [x, j, j+1] \vdash [A \to \alpha x \bullet \beta, i, j+1, e]\}$

$D^{Completer} = \{[A \to \alpha \bullet B\beta, i, j, e_1], [B \to \gamma\bullet, j, k, e_2] \vdash [A \to \alpha B \bullet \beta, i, k, e1 + e2]\}$

$D^{Predictor} = \{[A \to \alpha \bullet B\beta, i, j, e_1] \vdash [B \to \bullet\gamma, j, j, 0]\}$

$D^{ScanSubstituted} = \{[A \to \alpha \bullet x\beta, i, j, e], [b, j, j+1] \vdash [A \to \alpha x \bullet \beta, i, j+1, e+1]\}$

$D^{ScanDeleted} = \{[A \to \alpha \bullet x\beta, i, j, e] \vdash [A \to \alpha x \bullet \beta, i, j, e+1]\}$

$D^{ScanInserted} = \{[A \to \alpha \bullet \beta, i, j, e], [b, j, j+1] \vdash [A \to \alpha \bullet \beta, i, j+1, e+1]\}$

The *Initter*, *Scanner*, *Completer* and *Predictor* steps are similar to those in Earley's algorithm, with the difference that we have to keep track of the distance associated to the approximate trees in our items.

The *ScanSubstituted*, *ScanDeleted* and *ScanInserted* steps are error-repair steps, and they allow us to read unexpected symbols from the string while incrementing the distance. *ScanSubstituted* allows us to repair a substitution error in the string, *ScanDeleted* repairs a deletion error, and *ScanInserted* an insertion error.

Note that the *ScanInserted* step is defined slightly differently from the one in the original Lyon's algorithm, which is:

$D^{ScanInsertedLyon} = \{[A \to \alpha \bullet x\beta, i, j, e], [b, j, j+1] \vdash [A \to \alpha \bullet x\beta, i, j+1, e+1]\}$

This alternative version of *ScanInserted* cannot be used to repair an insertion error at the end of the input, since a repair is only attempted if we are expecting a terminal a and we find another terminal b instead, but not if we are expecting the end of the string. Lyon avoids this problem by extending the grammars used by the schema by changing the initial symbol S to S' and adding the additional rule $S' \to S\$$, where the character \$ acts as an end-of-sentence marker. However, we choose to keep our more general version of the step, and not to extend the grammar with this additional rule.

The set of final items and the subset of correct final items are:

$\mathcal{F} = \{[S \to \gamma\bullet, 0, n, e]\}$

$\mathcal{CF} = \{\iota = [S \to \gamma\bullet, 0, n, e] \mid \exists(t, e) \in \iota : t \text{ is a marked parse tree for } a_1 \ldots a_n\}$

Once we have defined a parser by means of an error-repair parsing schema, as we have done with Lyon's error-repair parser, we can use the formalism to prove its correctness. However, instead of showing a correctness proof for a particular case, we will describe something more interesting, namely how any standard parsing schema meeting a certain set of conditions can be systematically transformed to an error-repair parser, in such a way that the correctness of the standard parser implies that the error-repair parser obtained by applying the transformation is also correct.

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. We will first provide an informal description of the transformation and how it is applied, and then we will define it formally in Sect. 5, in order to be able to prove its correctness in Sect. 6.

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 elegantly 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)[7] is the concatenation of the marked yields of one or more of their antecedents,[8] 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 [11, 13, 14].

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 obtain error-repair parsers from them.

4.2. The transformation

The *error-repair transformation* of a prediction-completion parsing system $\mathcal{S}$ is the error-repair parsing system $\mathcal{R}(\mathcal{S})$ obtained by applying the following changes to it:

1. Transform the item set into the corresponding approximate item set by introducing a field which will store the corresponding parsing distance.
2. Add the following steps to the schema:

[7]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 the marked yield of the item.

[8]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.

(a) $D'^{SubstitutionHypothesis} = \{[a, i, i+1] \vdash [b, i, i+1, 1] \mid b \in \Sigma\}$
The consequent of this step contains the tree $b \rightarrow \underline{a}_{i+1}$, for each symbol a_{i+1} in the input string (input symbol) and each $b \in \Sigma$ (expected symbol). Generating this item corresponds to the error hypothesis that the symbol a_{i+1} that we find in the input string is the product of a substitution error, and should be b instead.

(b) $D'^{DeletionHypothesis} = \{\vdash [b, i, i, 1] \mid b \in \Sigma\}$
The consequent item contains the tree $b \rightarrow \epsilon$, for each position i in the input string. This corresponds to the error hypothesis that the symbol b, which should be the $i+1$th symbol in the input, has been deleted. The item $[b, i, i, 1]$ allows us to use this symbol during parsing even if it is not there.

(c) $D'^{InsertionHypothesis} = \{[a, i, i+1] \vdash [\epsilon, i, i+1, 1]\}$
The consequent of this step contains the tree $\epsilon \rightarrow \underline{a}_{i+1}$, for each input symbol a_{i+1} in the input string, which corresponds to the hypothesis that the symbol a_{i+1} in the input is the product of an insertion error, and therefore should not be taken into account in the parse.

(d) $D'^{BeginningInsertionCombiner} = \{[\epsilon, 0, j, e_1], [(a|\epsilon), j, k, e_2] \vdash [(a|\epsilon), 0, k, e_1+e_2]\}$

$D'^{OtherInsertionCombiner} = \{[(a|\epsilon), i, j, e_1], [\epsilon, j, k, e_2] \vdash [(a|\epsilon), i, k, e_1+e_2]\}$

These steps produce trees of the form $a_2(\epsilon(\underline{a}_1)a_2(\underline{a}_2))$ and $a_{i+1}(\underline{a}_{i+1}\epsilon(\underline{a}_{i+2}))$, respectively, when used to combine a single insertion hypothesis. Larger trees can be obtained by successive applications. If the first symbol in the input is an *inserted* character, its insertion hypothesis is combined with the hypothesis immediately to its right. Insertion hypotheses corresponding to symbols other than the first one are combined with the hypothesis immediately to their left. This is done so that the items generated by these steps will always contain trees rooted at a terminal symbol, rather than at ϵ: while any correct parsing schema must have steps to handle hypotheses of the form $[a, i, i+1]$, which can be straightforwardly transformed to handle these extended insertion hypotheses; some schemata (such as CYK) do not possess steps to handle subtrees rooted at ϵ, so their conversion to handle epsilon-rooted trees would be more complex.

(e) $D'^{CorrectHypothesis} = \{[a, i, i+1] \vdash [a, i, i+1, 0]\}$
The consequent of this item contains the tree $a \rightarrow \underline{a}_{i+1}$, for each input symbol a_{i+1} in the input string. Therefore, it is equivalent to the hypothesis $[a, i, i+1]$. This item corresponds to the hypothesis that there is no error in the symbol a_{i+1} in the input, hence the distance value 0.

3. 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 with 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

$D^{Predictor} = \{[A \rightarrow \alpha \bullet B\beta, i, j] \vdash [B \rightarrow \bullet\gamma, j, j] \mid B \rightarrow \gamma \in P\}$

produces the step

$D'^{Predictor} = \{[A \rightarrow \alpha \bullet B\beta, i, j, e] \vdash [B \rightarrow \bullet\gamma, j, j, 0] \mid B \rightarrow \gamma \in P\}$.

4. 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 $(i_0 \dots i_a)$):
   - If the step requires a hypothesis $[a, i, i+1]$, then change all appearances of the index $i+1$ to a new unbound index j.[9]
   - 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 e'_j.

Example 5. The Earley step

$D^{Completer} = \{[A \to \alpha \bullet B\beta, i, j], [B \to \gamma\bullet, j, k] \vdash [A \to \alpha B \bullet \beta, i, k]\}$

produces the step

$D'^{Completer} = \{[A \to \alpha \bullet B\beta, i, j, e_1], [B \to \gamma\bullet, j, k, e_2] \vdash [A \to \alpha B \bullet \beta, i, k, e_1 + e_2]\}$.

The Earley step

$D^{Scanner} = \{[A \to \alpha \bullet a\beta, i, j], [a, j, j+1] \vdash [A \to \alpha a \bullet \beta, i, j+1]\}$

produces the step

$D'^{Scanner} = \{[A \to \alpha \bullet a\beta, i, j, e_1], [a, j, k, e_2] \vdash [A \to \alpha a \bullet \beta, i, k, e_1 + e_2]\}$.

The CYK step

$D^{CYKUnary} = \{[a, i, i+1] \vdash [A, i, i+1] \mid A \to a \in P\}$

produces

$D'^{CYKUnary} = \{[a, i, j, e] \vdash [A, i, j, e] \mid A \to a \in P\}$. ■

5. Formal definition of the error-repair transformation

By applying these three simple transformation rules to a prediction-completion parsing schema, we obtain an error-repair parsing schema which shares its underlying semantics. As the transformation rules are totally systematic, they can be applied automatically, so that a system based on parsing schemata, such as the one described in [16, 19], can generate implementations of error-repair parsers from non-error-repair parsing schemata. However, in order for the transformation to be useful we need to ensure that the error-repair parsers obtained from it are correct. In order to do this, we first need to define some concepts that will take us to a formal definition of the transformation that we have informally described in the previous section.

[9]This is done because steps including hypotheses as antecedents are not strictly yield union steps according to the formal definition of yield union step that we will see later in Sect. 5.2. 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.

5.1. Some properties of trees and items

Let $t \in \mathit{Trees}(G)$ be a parse tree. We will say that t is an *anchored tree* if there is at least one marked terminal $\underline{a}_i$ in $\mathit{yield}(t)$. If t does not contain any marked terminal, then we will call it a *non-anchored tree.*

Note that the presence of marked terminals binds anchored trees to particular positions in the input string, while non-anchored trees are not bound to positions.

Definition 8. (yield limits)
We say that an anchored tree t is substring-anchored *if its yield is of the form $\alpha\, \underline{a}_{l+1}\, \underline{a}_{l+2} \, \dots \, \underline{a}_r\, \beta$, where α and β contain no marked terminals, for some $l, r \in \mathbb{N}$ $(l < r)$. The values l and r are called the* leftmost yield limit *and the* rightmost yield limit *of t, respectively, and we will denote them $\mathit{left}(t)$ and $\mathit{right}(t)$.*

Definition 9. (contiguous yield tree, marked yield)
We say that a tree t is a contiguous yield tree *if it is either substring-anchored or non-anchored.*

We define the marked yield *of a contiguous yield tree t, denoted $\mathit{yield}_m(t)$, as:*

- *The empty string ϵ, if t is non-anchored,*
- *The string $\underline{a}_{l+1}\, \underline{a}_{l+2} \, \dots \, \underline{a}_r$, if t is substring-anchored with yield limits $\mathit{left}(t) = l$, $\mathit{right}(t) = r$.*

Definition 10. (types of items according to yield)
Let $\mathcal{I}$ be an item set.

- *We will say that an item $\iota \in \mathcal{I}$ is a* homogeneously anchored item *if there exist l and $r \in \mathbb{N}$ such that, for every tree $t \in \iota$, t is substring-anchored and verifies that $\mathit{left}(t) = l$ and $\mathit{right}(t) = r$. In this case, we will call l the* leftmost yield limit *of the item ι, denoted $\mathit{left}(\iota)$, and r the* rightmost yield limit *of ι, denoted $\mathit{right}(\iota)$.*
- *We will call $\iota \in \mathcal{I}$ a* non-anchored item *if, for every tree $t \in \iota$, t is non-anchored.*
- *We will call any item $\iota \in \mathcal{I}$ which is in neither of these two cases a* heterogeneously anchored item.

We will say that an item set $\mathcal{I}$ is homogeneous *if it contains no heterogenously anchored items.*

Note that all trees contained in items in a homogeneous item set are contiguous yield trees.

Example 6. The Earley, CYK and Left-Corner [11] parsing schemata have, by construction, homogeneous item sets. Earley and Left-Corner[10] items of the form $\iota = [A \to \alpha \bullet \beta, i, j]$ where $i < j$ are homogeneously anchored items, where $\mathit{left}(\iota) = i$ and $\mathit{right}(\iota) = j$. Items where $i = j$ are non-anchored items. In the case of CYK, items of the form $\iota = [A, i, j]$ where $i < j$ are homogeneously anchored, with $\mathit{left}(\iota) = i$ and $\mathit{right}(\iota) = j$. ■

[10]The item set for a Left-Corner parser is a subset of the item set $\mathcal{I}_{Earley}$ of Example 1, of the form $\mathcal{I}_{LC} = \{[A \to X\alpha \bullet \beta, i, j] \mid A \to X\alpha\beta \in P \wedge X \in (N \cup \Sigma) \wedge 0 \leq i \leq j\} \cup \{[A \to \bullet, j, j] \mid A \to \epsilon \in P \wedge j \geq 0\} \cup \{[S \to \alpha \bullet, 0, 0]\}$.

Definition 11. (item representation set, item representation function)
Let $\mathcal{I}$ be a homogeneous item set, $\mathcal{H}$ a set of possible hypotheses.[11] *An* item representation set *for $\mathcal{I}$ is a set $R = \{(q,i,j) \in E \times \mathbb{N} \times \mathbb{N} \mid i \leq j\}$, where E is any set such that $\Sigma \subseteq E$ and there exists a function $r_R : R \to \mathcal{I} \cup \mathcal{H}$ (which we will call an* item representation function*) verifying that it is surjective (every item has at least one inverse image) and, for all $(q,i,j) \in R$,*

- *if $i < j$ and $\iota = r_R(q,i,j)$ is nonempty, ι is a homogeneously anchored item with $left(\iota) = i$ and $right(\iota) = j$.*
- *if $i = j$ and $\iota = r_R(q,i,j)$ is nonempty, ι is a non-anchored item.*
- *if $q \in \Sigma$ and $j = i{+}1$, $\iota = r_R(q,i,j)$ is the hypothesis $[q,i,i{+}1] = \{q((q,i{+}1))\} \in \mathcal{H}$.*
- *if $q \in \Sigma$ and $j \neq i+1$, $\iota = r_R(q,i,j)$ is the empty item $\emptyset$.*

Note that a final item for a string of length n will always be of the form $r_R(q,0,n)$ for some q.

Example 7. In the case of the Earley parser, we consider the representation set $R_{Earley} = \{(q,i,j) \in (D(P) \cup \Sigma) \times \mathbb{N} \times \mathbb{N} \mid i \leq j\}$, where the set of dotted productions $D(P)$ is defined as $\{(A \to \alpha, k) \mid A \to \alpha \in P \wedge k \in \mathbb{N} \wedge 0 \leq k \leq \mathit{length}(\alpha)\}$. This allows us to define the obvious representation function for the Earley item set, $r_{R_{Earley}} : ((A \to \gamma, k), i, j) \to [A \to \alpha \bullet \beta, i, j]$, where α is the substring $\gamma_1 \dots \gamma_k$ of γ and β is the rest of γ; and $r_{R_{Earley}} : (a,i,j) \to [a,i,j]$. ■

5.2. Some properties of deduction steps

Definition 12. (yield union step set)
Let $\mathcal{I}$ be a homogeneous item set, and $R \subseteq E \times \mathbb{N} \times \mathbb{N}$ an item representation set for $\mathcal{I}$, with representation function r_R. If we write $[a,b,c]$ as shorthand for $r_R(a,b,c)$, a yield union step set *is a set of deduction steps of the form*

$$\begin{aligned}\{[q_1,i_0,i_1],[q_2,i_1,i_2],\dots,[q_m,i_{m-1},i_m],[c_1,j_1,k_1],[c_2,j_2,k_2],\dots,[c_n,j_n,k_n] \vdash \\ [q_c,i_0,i_m] \mid \\ i_0 \leq i_1 \leq \dots \leq i_m \in \mathbb{N} \wedge j_1,\dots,j_n,k_1,\dots,k_n \in \mathbb{N} \wedge j_i \leq k_i \wedge \\ P(q_1,q_2,\dots,q_m,c_1,c_2,\dots,c_n,q_c) = 1\}\end{aligned}$$

where P is a boolean function, $P : E^{m+n+1} \to \{0,1\}$.

Therefore, a yield union step set is a step set in which some of the antecedent items have contiguous yields whose union is the consequent's yield. If we represent the antecedent and consequent items as $[q,l,r]$, the only constraints allowed on the left and right positions l and r are that l should always be lesser than or equal to r for all items, and that the (l,r) intervals of some antecedents must be contiguous and their union be the interval corresponding to the consequent. Any constraint is allowed on the entities q and c, as denoted by P.

[11]Note that, in this definition, $\mathcal{H}$ represents the set of all the possible hypotheses of the form $\{a((a,i))\}$ with $a \in \Sigma^\star$ and $i \in \mathbb{N}$, and not only the hypotheses associated to a particular input string.

Example 8. The set of Earley *Completer* steps is a yield union step set with the representation function $r_{R_{Earley}}$, because it can be written as:

$$\{[q_1, i_0, i_1], [q_2, i_1, i_2] \vdash [q_c, i_0, i_2] \mid i_0 \leq i_1 \leq i_2 \in \mathbb{N} \wedge P(q_1, q_2, q_c) = 1\}$$

with $P(x, y, z) = (\exists A, B, \alpha, \beta, \gamma$ such that $x = A \rightarrow \alpha \bullet B\beta, y = B \rightarrow \gamma\bullet, z = A \rightarrow \alpha B \bullet \beta)$. ■

Definition 13. (predictive step set)
Let $\mathcal{I}$ be a homogeneous item set, and $R \subseteq E \times \mathbb{N} \times \mathbb{N}$ an item representation set for $\mathcal{I}$, with representation function r_R. If we write $[a, b, c]$ as shorthand for $r_R(a, b, c)$, a predictive step set *is a set of deduction steps of the form*

$$\{[q_1, j_1, k_1], [q_2, j_2, k_2], \ldots, [q_n, j_n, k_n] \vdash \\ [q_c, f(j_1, k_1, j_2, k_2, \ldots, j_n, k_n), f(j_1, k_1, j_2, k_2, \ldots, j_n, k_n)] \mid \\ j_1, \ldots, j_n, k_1, \ldots, k_n \in \mathbb{N} \wedge j_i \leq k_i \wedge P(q_1, q_2, \ldots, q_n, q_c) = 1\},$$

where P is a boolean function, $P : E^{n+1} \rightarrow \{0, 1\}$, and f is a natural function, $f : \mathbb{N}^{2n} \rightarrow \mathbb{N}$.

Therefore, a predictive step set is a step set in which the consequent is a non-anchored item. If we represent the antecedent and consequent items as $[q, l, r]$, the only constraints allowed on the left and right positions l and r are that l should always be lesser than or equal to r for all items, and that the (l, r) indexes of the consequent must be equal and a function of the (l, r) indexes of the antecedents. Any constraint is allowed on the entities q, as denoted by P.

Example 9. The set of Earley *Predictor* steps is a predictive step set, because it can be written as:

$$\{[q_1, j_1, k_1]\} \vdash [q_c, f(j_1, k_1), f(j_1, k_1)] \mid \wedge j_1, k_1 \in \mathbb{N} \wedge j_1 \leq k_1 \wedge P(q_1, q_c) = 1\}$$

where $f(x, y) = y$, and $P(x, y) = (\exists A, B, \alpha, \beta, \gamma$ such that $x = A \rightarrow \alpha \bullet B\beta, y = B \rightarrow \bullet\gamma)$ with $B \rightarrow \gamma$ a production in the grammar. ■

Definition 14. (prediction-completion parsing schema)
An uninstantiated parsing system $(\mathcal{I}, \mathcal{K}, D)$ is a prediction-completion parsing system *if there exists a representation function r_R such that D can be written as union of sets $D_1 \cup D_2 \cup \ldots \cup D_n$, where each D_i is either a predictive step set or a yield union step set with respect to that representation function.*

A parsing schema $\mathcal{S}$ is said to be a prediction-completion parsing schema *if it maps each grammar G in a class $\mathcal{CG}$ to a prediction-completion parsing system.*

Example 10. It is easy to check that the *Earley*, *CYK* and *Left-Corner* parsing schemata are prediction-completion parsing schemata, as their sets of deduction steps can be rewritten as the union of predictive step sets and yield union step sets. For example, in the case of *Earley*, the standard *Initter* and *Predictor* are predictive step sets, while *Completer* and *Scanner* are yield union step sets. In the case of the *Scanner* step, we can see that it is a yield union step set by rewriting it as $D^{Scanner} = \{[A \rightarrow \alpha \bullet x\beta, i, j], [x, j, k] \vdash [A \rightarrow \alpha x \bullet \beta, i, k]\}$ (see footnote on page 12). ■

5.3. *The error-repair transformation (formal definition)*

Let $\mathcal{S} = (\mathcal{I}, \mathcal{K}, D)$ be a prediction-completion parsing system.

Let $D = D_1 \cup D_2 \cup \ldots \cup D_n$ be an expression of D where each D_i is either a predictive step set or a yield union step set with respect to a representation function r_R associated to a representation set $R \subseteq E \times \mathbb{N} \times \mathbb{N}$. This expression must exist, by definition of prediction-completion parsing system. As before, we will denote $r_R(q, i, j)$ by $[q, i, j]$.

The *error-repair transformation* of $\mathcal{S}$, denoted $\mathcal{R}(\mathcal{S})$, is an error-repair parsing system $(\mathcal{I}', \mathcal{K}, D')$ under the distance function $\hat{d}$, where $\mathcal{I}'$ and D' are defined as follows.

5.3.1. *Items of the error-repair transformation*

$\mathcal{I}' = \mathcal{I}'_1 \cup \mathcal{I}'_2$, with

$$\begin{aligned} \mathcal{I}'_1 = \{ \ \{(t, e) \in \mathit{ApTrees}(G) \mid t \text{ is substring-anchored} \wedge \\ \mathit{left}(t) = i \wedge \mathit{right}(t) = j \wedge \\ \exists i', j' \in \mathbb{N}, t' \in [q, i', j'] \cup \{\epsilon(\epsilon)\} : d(t, t') = e\} \quad | \\ q \in E \cup \{\epsilon\}, \ \ i, j, e \in \mathbb{N} \ \ \} \end{aligned}$$

$$\begin{aligned} \mathcal{I}'_2 = \{ \ \{(t, e) \in \mathit{ApTrees}(G) \mid t \text{ is non-anchored} \wedge \\ \exists i', j' \in \mathbb{N}, t' \in [q, i', j'] \cup \{\epsilon(\epsilon)\} : d(t, t') = e\} \quad | \\ q \in E \cup \{\epsilon\}, \ \ e \in \mathbb{N} \ \ \} \end{aligned}$$

Note that $\mathcal{I}'$ verifies the definition of an approximate item set if, and only if, $d(t_1, t_2) = \infty$ for every $t_1 \in [q_1, i_1, j_1], t_2 \in [q_2, i_2, j_2]$ such that $q_1 \neq q_2$ (this can be easily proved by the triangle inequality, and it can be seen that if this condition does not hold, there will be trees that appear in more than one item in $\mathcal{I}'$, thus violating the definition). Known item sets such as the Earley, CYK or Left-Corner item sets meet this condition when using the distance function $\hat{d}$; since if two items have $q_1 \neq q_2$, their respective trees differ in non-frontier nodes and therefore the distance between them is always ∞.

5.3.2. *Deduction steps of the error-repair transformation*

We define a set $R' = \{(q, i, j, e) \in (E \cup \{\epsilon\}) \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \mid i \leq j\}$ for $\mathcal{I}'$, and call it a *robust representation set* for $\mathcal{I}'$.

We define $r'_R : R' \to (\mathcal{I}' \cup \mathcal{H})$ as the function that maps each tuple (q, i, j, e) to the item:

- $\{(t, e) \in \mathit{ApTrees}(G) \mid t \text{ is substring-anchored } \wedge \mathit{left}(t) = i \wedge \mathit{right}(t) = j \wedge \exists i', j' \in \mathbb{N}, t' \in [q, i', j'] \cup \{\epsilon(\epsilon)\} : d(t, t') = e\} \in \mathcal{I}'_1$, if $i \neq j$.

- $\{(t, e) \in \mathit{ApTrees}(G) \mid t \text{ is non-anchored } \wedge \exists i', j' \in \mathbb{N}, t' \in [q, i', j'] \cup \{\epsilon(\epsilon)\} : d(t, t') = e\} \in \mathcal{I}'_2$, if $i = j$.

We call r'_R a *robust representation function* for $\mathcal{I}'$, and we will denote $r'_R(q, i, j, e)$ by $[\![q, i, j, e]\!]$. Note that the function r'_R is trivially surjective by construction: the images for each of the two cases of its definition are $\mathcal{I}'_1$ and $\mathcal{I}'_2$, respectively, and each hypothesis $[a, i, i+1] \in \mathcal{H}$ is the image of $(a, i, i+1, 0)$.

The set of deduction steps of the error-repair transformation is defined as $D' = D'^{CorrectHyp} \cup D'^{SubstHyp} \cup D'^{DelHyp} \cup D'^{InsHyp} \cup D'^{BegInsComb} \cup D'^{OthInsComb} \cup D'^{DistIncr} \cup D'_1 \cup D'_2 \cup \ldots \cup D'_n$, where[12]

$D'^{CorrectHyp} = \{[a, i, j] \vdash [\![a, i, j, 0]\!]\}$

$D'^{SubstHyp} = \{[a, i, j] \vdash [\![b, i, j, 1]\!] \mid b \in \Sigma\}$

$D'^{DelHyp} = \{\vdash [\![b, i, i, 1]\!] \mid b \in \Sigma\}$

$D'^{InsHyp} = \{[a, i, j] \vdash [\![\epsilon, i, j, 1]\!]\}$

$D'^{BegInsComb} = \{[\![\epsilon, 0, j, e_1]\!], [\![x, j, k, e_2]\!] \vdash [\![x, 0, k, e_1 + e_2]\!] \mid x \in \Sigma \cup \{\epsilon\}\}$

$D'^{OthInsComb} = \{[\![x, i, j, e_1]\!], [\![\epsilon, j, k, e_2]\!] \vdash [\![x, i, k, e_1 + e_2]\!] \mid x \in \Sigma \cup \{\epsilon\}\}$

$D'^{DistIncr} = \{[\![x, i, j, e]\!] \vdash [\![x, i, j, e+1]\!] \mid x \in (E \cup \{\epsilon\})\}$

For each yield union step set D_i of the form

$$\begin{aligned} D_i = \{&[q_1, i_0, i_1], [q_2, i_1, i_2], \ldots, [q_m, i_{m-1}, i_m], [c_1, j_1, k_1], [c_2, j_2, k_2], \ldots, [c_n, j_n, k_n] \vdash \\ &[q_c, i_0, i_m] \mid i_0 \leq i_1 \leq \ldots \leq i_m \in \mathbb{N} \wedge j_1, \ldots, j_n, k_1, \ldots, k_n \in \mathbb{N} \wedge j_i \leq k_i \wedge \\ &P(q_1, q_2, \ldots, q_m, c_1, c_2, \ldots, c_n, q_c) = 1\} \end{aligned}$$

we obtain

$$\begin{aligned} D'_i = \{&[\![q_1, i_0, i_1, e_1]\!], [\![q_2, i_1, i_2, e_2]\!], \ldots, [\![q_m, i_{m-1}, i_m, e_m]\!], \\ &[\![c_1, j_1, k_1, e'_1]\!], [\![c_2, j_2, k_2, e'_2]\!], \ldots, [\![c_n, j_n, k_n, e'_n]\!] \vdash \\ &[\![q_c, i_0, i_m, e_1 + \ldots + e_m]\!] \mid i_0 \leq i_1 \leq \ldots \leq i_m \in \mathbb{N} \wedge j_1, \ldots, j_n, k_1, \ldots, k_n, \\ &e'_1, \ldots, e'_n, e_1, \ldots, e_m \in \mathbb{N} \wedge j_i \leq k_i \wedge P(q_1, q_2, \ldots, q_m, c_1, c_2, \ldots, c_n, q_c) = 1\} \end{aligned}$$

For each predictive step set D_i of the form

$$\begin{aligned} \{&[q_1, j_1, k_1], [q_2, j_2, k_2], \ldots, [q_n, j_n, k_n] \vdash \\ &[q_c, f(j_1, k_1, j_2, k_2, \ldots, j_n, k_n), f(j_1, k_1, j_2, k_2, \ldots, j_n, k_n)] \mid \\ &j_1, \ldots, j_n, k_1, \ldots, k_n \in \mathbb{N} \wedge j_i \leq k_i \wedge P(q_1, q_2, \ldots, q_n, q_c) = 1\}, \end{aligned}$$

we obtain

$$\begin{aligned} D'_i = \{&[\![q_1, j_1, k_1, e_1]\!], [\![q_2, j_2, k_2, e_2]\!], \ldots, [\![q_n, j_n, k_n, e_n]\!] \vdash \\ &[\![q_c, f(j_1, k_1, j_2, k_2, \ldots, j_n, k_n), f(j_1, k_1, j_2, k_2, \ldots, j_n, k_n), 0]\!] \mid \\ &j_1, \ldots, j_n, k_1, \ldots, k_n, e_1, \ldots, e_n \in \mathbb{N} \wedge j_i \leq k_i \wedge P(q_1, q_2, \ldots, q_n, q_c) = 1\} \end{aligned}$$

Note that we have included a step in the transformation, $D'^{DistIncr}$, that is used to

[12]The names of deduction steps have been shortened with respect to those given in Sect. 4.2 for reasons of space.

increase the parsing distance of an item. This step was not mentioned in the informal description of Section 4.2. The reason is that this step is not necessary in practice, since in a practical implementation of an error-repair parser we are not interested in strict completeness (finding all possible approximate parses) as we only need the minimal-distance parses. However, if we are able to prove that a parser that includes $D'^{DistIncr}$ is correct, it immediately follows that a version of the same parser that removes that step generates all minimal final items, as any sequence of deductions made with an item $[\![x, i, j, e+1]\!]$ obtained from $D'^{DistIncr}$ can also be made directly with its antecedent $[\![x, i, j, e]\!]$, avoiding the use of this step.

Example 11. Consider the Earley parsing schema, with the *Scanner* step rewritten in order to be a yield union step, as explained in Example 10. Its robust transformation is given by $(\mathcal{I}', \mathcal{K}, D')$, where:

$\mathcal{I}' = \mathcal{I}'_{11} \cup \mathcal{I}'_{12} \cup \mathcal{I}'_{13} \cup \mathcal{I}'_{21} \cup \mathcal{I}'_{22}$

$\mathcal{I}'_{11} = [\![A \rightarrow \alpha \bullet \beta, i, j, e]\!]$ with $i \neq j$, representing the set of substring-anchored approximate trees (t', e) such that $\mathit{left}(t') = i$, $\mathit{right}(t') = j$, and $d(t', t) = e$ for some t in an Earley item of the form $[A \rightarrow \alpha \bullet \beta, i', j']$ for some $i', j' \in \mathbb{N}$

$\mathcal{I}'_{12} = [\![a, i, j, e]\!]$ with $i \neq j$, representing the set of substring-anchored approximate trees (t', e) such that $\mathit{left}(t') = i$, $\mathit{right}(t') = j$, and $d(t', t) = e$ for some t in a hypothesis of the form $[a, i', j']$ for some $i', j' \in \mathbb{N}$

$\mathcal{I}'_{13} = [\![\epsilon, i, j, e]\!]$ with $i \neq j$, representing the set of substring-anchored approximate trees (t', e) such that $\mathit{left}(t') = i$, $\mathit{right}(t') = j$, and $d(t', t) = e$ for $t = \epsilon(\epsilon)$

$\mathcal{I}'_{21} = [\![A \rightarrow \alpha \bullet \beta, i, i, e]\!]$, representing the set of non-anchored approximate trees (t', e) such that $d(t', t) = e$ for some t in an Earley item of the form $[A \rightarrow \alpha \bullet \beta, i', j']$ for some i', j'

$\mathcal{I}'_{22} = [\![\epsilon, i, i, e]\!]$, representing the set of non-anchored approximate trees (t', e) such that $d(t', t) = e$ for $t = \epsilon(\epsilon)$

$r'_R(x, i, j, e) = [\![x, i, j, e]\!]$, for all $x \in D(P) \cup \Sigma \cup \{\epsilon\}$

$D' = D'^{CorrectHyp} \cup D'^{SubstHyp} \cup D'^{DelHyp} \cup D'^{InsHyp} \cup D'^{BegInsComb} \cup D'^{OthInsComb} \cup$

$D'^{DistIncr} \cup'^{DistIncr2} \cup D'^{DistIncr3} \cup D'^{Initter} \cup D'^{Scanner} \cup D'^{Completer} \cup D'^{Predictor}$

$$D'^{CorrectHyp} = \{[a,i,j] \vdash [\![a,i,j,0]\!]\}$$
$$D'^{SubstHyp} = \{[a,i,j] \vdash [\![b,i,j,1]\!] \mid b \in \Sigma\}$$
$$D'^{DelHyp} = \{\vdash [\![b,i,i,1]\!] \mid b \in \Sigma\}$$
$$D'^{InsHyp} = \{[a,i,j] \vdash [\![\epsilon,i,j,1]\!]\}$$
$$D'^{BegInsComb} = \{[\![\epsilon,0,j,e_1]\!], [\![x,j,k,e_2]\!] \vdash [\![x,0,k,e_1+e_2]\!] \mid x \in \Sigma \cup \{\epsilon\}\}$$
$$D'^{OthInsComb} = \{[\![x,i,j,e_1]\!], [\![\epsilon,j,k,e_2]\!] \vdash [\![x,i,k,e_1+e_2]\!] \mid x \in \Sigma \cup \{\epsilon\}\}$$
$$D'^{DistIncr} = \{[\![x,i,j,e]\!] \vdash [\![x,i,j,e+1]\!]\}$$
$$D'^{DistIncr2} = \{[\![A \to \alpha \bullet \beta,i,j,e]\!] \vdash [\![A \to \alpha \bullet \beta,i,j,e+1]\!]\}$$
$$D'^{DistIncr3} = \{[\![\epsilon,i,j,e]\!] \vdash [\![\epsilon,i,j,e+1]\!]\}$$
$$D'^{Initter} = \{\vdash [\![S \to \bullet\gamma,0,0,0]\!]\}$$
$$D'^{Scanner} = \{[\![A \to \alpha \bullet x\beta,i,j,e_1]\!], [\![x,j,k,e_2]\!] \vdash [\![A \to \alpha x \bullet \beta,i,k,e_1+e_2]\!]\}$$
$$D'^{Completer} = \{[\![A \to \alpha \bullet B\beta,i,j,e_1]\!], [\![B \to \gamma\bullet,j,k,e_2]\!] \vdash [\![A \to \alpha B \bullet \beta,i,k,e_1+e_2]\!]\}$$
$$D'^{Predictor} = \{[\![A \to \alpha \bullet B\beta,i,j,e_1]\!] \vdash [\![B \to \bullet\gamma,j,j,0]\!]\} \quad \blacksquare$$

6. Proof of correctness of the error-repair transformation

The robust transformation function $\mathcal{R}$ 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 robust parsers generated will be correct under certain conditions. This will be done in the following two theorems.

Let $\mathcal{S} = (\mathcal{I}, \mathcal{K}, D)$ be a prediction-completion parsing system with representation function $r_R(q,i,j) = [q,i,j]$, and with $D = D_1 \cup D_2 \cup \ldots \cup D_n$ an expression of D where each D_i is either a predictive step set or a yield union step set.

Theorem 1. (preservation of the soundness of the transformation)
If $(\mathcal{I}, \mathcal{K}, D)$ is sound, every deduction step δ in a predictive step set $D_i \subseteq D$ has a nonempty consequent, and for every deduction step δ in a yield union step set $D_i \subseteq D$ of the form

$$\begin{aligned} D_i = \{\ & [q_1,i_0,i_1],[q_2,i_1,i_2],\ldots,[q_m,i_{m-1},i_m],[c_1,j_1,k_1],[c_2,j_2,k_2],\ldots,[c_n,j_n,k_n] \vdash \\ & [q_c,i_0,i_m]\ /\ i_0 \le i_1 \le \ldots \le i_m \in \mathbb{N} \wedge j_1,\ldots,j_n,k_1,\ldots,k_n \in \mathbb{N} \wedge j_i \le k_i \wedge \\ & P(q_1,q_2,\ldots,q_m,c_1,c_2,\ldots,c_n,q_c) = 1\} \end{aligned}$$

there exists a function $C_\delta : Trees'(G)^m \to Trees'(G)$ (tree combination function) such that:

- *If $(t_1,\ldots,t_m)$ is a tuple of trees in $Trees(G)$ such that $t_w \in [q_w,i_{w-1},i_w]$ $(1 \le w \le m)$, then $C_\delta(t_1,\ldots,t_m) \in [q_c,i_0,i_m]$.*

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

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

Theorem 2. (preservation of the completeness of the transformation)
If $(\mathcal{I}, \mathcal{K}, D)$ *is sound and complete, then* $\mathcal{R}(\mathcal{I}, \mathcal{K}, D)$ *is complete.*

Note that the condition regarding 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 bigger portion of the input string. In practice, the existence of a tree combination function simply 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 (i.e. the combined tree contains the errors or discrepancies from all the antecedent trees). For example, in the case of the Earley *Completer* step, it is easy to see that 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. Combination functions for the remaining yield union steps in *CYK*, *Earley* and *Left-Corner* parsers are equally obvious.

6.1. Proof of Theorem 1

Let $\mathcal{S} = (\mathcal{I}, \mathcal{K}, D)$ be a prediction-completion parsing system verifying the conditions of Theorem 1, and $\mathcal{R}(\mathcal{S}) = (\mathcal{I}', \mathcal{K}, D')$ the error-repair transformation of $\mathcal{S}$.

We define a *correct item* in the error-repair parsing system $\mathcal{R}(\mathcal{S})$ for a particular input string $a_1 \dots a_n$ as an approximate item $r'_R(q, i, j, e) = [\![q, i, j, e]\!]$ containing an approximate tree (t, e) such that t is a contiguous yield tree with $yield_m(t) = \underline{a}_{i+1} \dots \underline{a}_j$ (we call such an approximate tree a *correct approximate tree* for that item and string). Note that a final item containing such an approximate tree verifies the definition of a correct final item that we gave earlier.

We will prove that $\mathcal{R}(\mathcal{S})$ is sound (all valid final items are correct) by proving the stronger claim that *all valid items are correct.*

To prove this, we take into account that a valid item is either a hypothesis or the consequent of a deduction step with valid antecedents. Therefore, in order to prove that valid items are correct, it suffices to show that

(i) hypotheses are correct, and that
(ii) if the antecedents of a deduction step are correct, then the consequent is correct.

Proving (i) is trivial, since each hypothesis $[a, i-1, i]$ obtained from the function $\mathcal{K}$ contains a single tree with yield $\underline{a}_i$.

To prove (ii), we will show that it holds for all the deduction step sets in D'. Let $D = D_1 \cup D_2 \cup \dots \cup D_n$ be an expression of D where each D_i is either a predictive step set or a yield union step set (this expression must exist, since $\mathcal{S}$ is a prediction-completion parsing system). Then the set of deduction steps D', used in the error-repair parsing system $\mathcal{R}(\mathcal{S})$, can be written as $D' = D'^{CorrectHyp} \cup D'^{SubstHyp} \cup D'^{DelHyp} \cup D'^{InsHyp} \cup D'^{BegInsComb} \cup D'^{OthInsComb} \cup D'^{DistIncr} \cup D'_1 \cup D'_2 \cup \ldots \cup D'_n$, as defined above. We will show that (ii) holds for each of the deduction step sets D_i, by proving it separately for each step set:

- For the deduction step sets D'_i, by considering two possible cases:
  (1) D'_i comes from a yield union step set D_i.
  (2) D'_i comes from a predictive step set D_i.
- For the fixed deduction step sets $D'^{CorrectHyp}$, $D'^{SubstHyp}$, etc., by considering each set separately.

6.1.1. Proof for case (1)

Let us consider the first case, where D'_i comes from a yield union step set D_i. Then, by definition of the error-repair transformation, D_i can be written as

$$D_i = \{[q_1, i_0, i_1], [q_2, i_1, i_2], \ldots, [q_m, i_{m-1}, i_m], [c_1, j_1, k_1], [c_2, j_2, k_2], \ldots, [c_n, j_n, k_n] \vdash [q_c, i_0, i_m] \mid i_0 \leq i_1 \leq \ldots \leq i_m \in \mathbb{N} \wedge j_1, \ldots, j_n, k_1, \ldots, k_n \in \mathbb{N} \wedge j_i \leq k_i \wedge P(q_1, q_2, \ldots, q_m, c_1, c_2, \ldots, c_n, q_c) = 1\}$$

and D'_i can be written as

$$D'_i = \{[\![q_1, i_0, i_1, e_1]\!], [\![q_2, i_1, i_2, e_2]\!], \ldots, [\![q_m, i_{m-1}, i_m, e_m]\!], [\![c_1, j_1, k_1, e'_1]\!], [\![c_2, j_2, k_2, e'_2]\!], \ldots, [\![c_n, j_n, k_n, e'_n]\!] \vdash [\![q_c, i_0, i_m, e_1 + \ldots + e_m]\!] \mid i_0 \leq i_1 \leq \ldots \leq i_m \in \mathbb{N} \wedge j_1, \ldots, j_n, k_1, \ldots, k_n, e'_1, \ldots, e'_n, e_1, \ldots, e_m \in \mathbb{N} \wedge j_i \leq k_i \wedge P(q_1, q_2, \ldots, q_m, c_1, c_2, \ldots, c_n, q_c) = 1\}.$$

Let $\delta' \in D'_i$ be a particular deduction step in this set. We will prove that, if the antecedents of δ' are correct, then the consequent is also correct.

Let $\delta \in D_i$ be the deduction step in D_i with the same values of $q_1, \ldots, q_m, i_0, \ldots, i_m$, $c_1, \ldots, c_n, j_1, \ldots, j_n$, $k_1, \ldots, k_n$ as δ'. Let C_δ be a combination function for this step δ.

If the antecedents of δ' are correct, then there exist m approximate trees $(t'_w, e_w) \in [\![q_w, i_{w-1}, i_w, e_w]\!] (1 \leq w \leq m)$. By definition of r'_R, we know that for each t'_w there exists a tree $t_w \in [q_w, i'_w, i''_w]$ such that $\hat{d}(t'_w, t_w) = e_w$. Taking into account that indexes associated to marked terminals do not affect our distance $\hat{d}$, it can be shown that we can assume, without loss of generality, that $t_w \in [q_w, i_{w-1}, i_w]$.

By the first condition that C_δ must verify, we know that $C_\delta(t_1, \ldots, t_n) \in [q_c, i_0, i_m]$.

By the second condition, we know that $\hat{d}(C_\delta(t'_1, \ldots, t'_n), C_\delta(t_1, \ldots, t_n)) = \Sigma_{w=1}^m e_w$.

These two facts imply that $(C_\delta(t'_1, \ldots, t'_n), \Sigma_{w=1}^m e_w)$ is a member of an item $[\![q_c, k_1, k_2, \Sigma_{w=1}^m e_w]\!] \in \mathcal{I}'$ for some $k_1, k_2 \in \mathbb{N}$.

By hypothesis, the antecedents of δ' are correct, so we know that $\mathit{yield}(t'_w) = \underline{a}_{i_{w-1}+1} \ldots \underline{a}_{i_w}$. Therefore, by the second condition of a combination function, $C_\delta(t'_1, \ldots, t'_n)$ is a contiguous yield tree with yield $\underline{a}_{i_0} \ldots \underline{a}_{i_m}$. Hence, we know that $k_1 = i_0$, $k_2 = i_m$, and $(C_\delta(t'_1, \ldots, t'_n), \Sigma_{w=1}^m e_w)$ is a correct approximate tree for the consequent item of δ', $[\![q_c, i_0, i_m, \Sigma_{w=1}^m e_w]\!]$. This proves that the consequent of δ' is correct.

6.1.2. Proof for case (2)

Let us consider the second case, where D'_i comes from a predictive step set D_i. In this case, the consequent of any deduction step $\delta' \in D'_i$ is of the form $[\![q_c, v, v, 0]\!]$ for some v. By construction of r'_R, this means that the consequent is the set of non-anchored approximate trees $(t, 0)$ with $t \in [q_c, k_1, k_2]$ for any $k_1, k_2 \in \mathbb{N}$.

Let $\delta \in D_i$ be the deduction step in D_i with the same values of $q_1, \ldots, q_n$, $j_1, \ldots, j_n$, $k_1, \ldots, k_n$ as δ'. The consequent of this step is $[q_c, v, v] \in \mathcal{I}$. By definition of representation function, $[q_c, v, v]$ must be a non-anchored item. Therefore, any tree $t \in [q_c, v, v]$ is non-anchored. By hypothesis, since $[q_c, v, v]$ is a consequent of a deduction step from a predictive step set $D_i \subseteq D$, we know that $[q_c, v, v]$ is nonempty, so there exists at least one non-anchored tree $t \in [q_c, v, v]$. The tree $(t, 0)$ is a correct approximate tree in $[\![q_c, v, v, 0]\!]$. Therefore, the consequent of δ' is correct.

6.1.3. Proof for fixed deduction step sets

We consider each deduction step set separately:

- A $D'^{CorrectHyp}$ step is of the form $[a, i, j] \vdash [\![a, i, j, 0]\!]$. The antecedent of this step can only be correct in $\mathcal{S}$ if $j = i+1$, since otherwise it equals the empty item. If the antecedent is correct, then there exists a hypothesis $[a, j-1, j]$, containing a tree $a((a, j)) \in \mathit{Trees}(G)$. In this case, since $j = i+1$, the consequent is $[\![a, j-1, j, 0]\!]$.

  By definition of r'_R, the consequent item $[\![a, j-1, j, 0]\!]$ is the set of substring-anchored approximate trees $(t, 0) \in \mathit{ApTrees}(G)$ such that $\mathit{left}(t) = j-1$, $\mathit{right}(t) = j$, and $\hat{d}(t, u) = 0$ for some $u \in [a, k_1, k_2] (k_1, k_2 \in \mathbb{N})$. One such tree is $(a((a, j)), 0) \in \mathit{ApTrees}(G)$, which is trivially a correct tree for this item. Therefore, the consequent item of $D'^{CorrectHyp}$ is correct.

- The consequent item of a step in $D'^{SubstHyp}$, $[\![b, j-1, j, 1]\!]$, is the set of substring-anchored approximate trees $(t, 1) \in \mathit{ApTrees}(G)$ such that $\mathit{left}(t) = j-1$, $\mathit{right}(t) = j$, and $\hat{d}(t, u) = 1$ for some $u \in [b, k_1, k_2]$. One such tree is $(b((a, j)), 1) \in \mathit{ApTrees}(G)$, where $b((a, j))$ is at distance 1 from the tree $b((b, j)) \in [b, j-1, j]$ by a substitution operation. This is a correct tree for the consequent, therefore the consequent of $D'^{SubstHyp}$ is correct. Note that the antecedent is not used in the proof, so the transformation would still be sound with a step $\vdash [\![b, j-1, j, 1]\!]$. We only use the antecedent to restrict the range of j.

- In the case of D'^{DelHyp}, a correct tree for the consequent is $(b(\epsilon)$,1), where $b(\epsilon)$ is at distance 1 from any $b((b, j)) \in [b, j-1, j]$.

- In the case of D'^{InsHyp}, a correct tree for the consequent is $(\epsilon((a, j)), 1)$, which is at distance 1 from $\epsilon(\epsilon)$.

- A correct tree for the consequent of steps in $D'^{BegInsComb}$ is obtained by appending a correct tree in the antecedent $[\![\epsilon, 0, j, e_1]\!]$ as the leftmost child of a correct tree in the antecedent $[\![x, j, k, e_2]\!]$.

- A correct tree for the consequent of steps in $D'^{OthInsComb}$ is obtained by appending a correct tree in the antecedent $[\![\epsilon, j, k, e_2]\!]$ as the rightmost child of a correct tree in the antecedent $[\![x, i, j, e_1]\!]$.

- A correct tree for the consequent of steps in $D'^{DistIncr}$ is $(t, e+1)$, for any approximate tree (t, e) in the antecedent $[\![x, i, j, e]\!]$.

6.1.4. End of the proof of Theorem 1

As a result, we have proved that, under the theorem's hypotheses, (ii) holds for every deduction step. This implies that all valid items are correct and, therefore, that $\mathcal{R}(\mathcal{S})$ is sound, as we wanted to prove.

6.2. Proof of Theorem 2

Let $\mathcal{S} = (\mathcal{I}, \mathcal{K}, D)$ be a sound and complete prediction-completion parsing system, and $\mathcal{R}(\mathcal{S}) = (\mathcal{I}', \mathcal{K}, D')$ the error-repair transformation of $\mathcal{S}$. We will prove that $\mathcal{R}(\mathcal{S})$ is complete. Proving completeness for this deduction system is proving that, given an input string $a_1 \dots a_n$, all correct final items are valid. Therefore, given a string $a_1 \dots a_n$, we have to prove that every item containing an approximate tree (t, e) such that t is a marked parse tree for $a_1 \dots a_n$ can be inferred from the hypotheses.

Since the robust representation function for $\mathcal{R}(\mathcal{S})$, r'_R, is surjective, we know that every final item in this deduction system can be written as $[\![q, i, j, e]\!]$. Therefore, proving completeness is equivalent to proving the following proposition:

Proposition 1. *Given any string $a_1 \dots a_n$, every correct final item of the form $[\![q, i, j, e]\!]$ is valid in the instantiated parsing system $\mathcal{R}(\mathcal{S})(a_1 \dots a_n) = (\mathcal{I}', \mathcal{K}(a_1 \dots a_n), D')$.*

We will prove this proposition by induction on the distance e.

6.2.1. Base case (e=0)

Items in the item set $\mathcal{I}'$ where the distance e is 0 can be mapped to items from the item set $\mathcal{I}$ (corresponding to the original non-error-repair parser) by the function $f : \{[\![q, i, j, 0]\!] \in \mathcal{I}'\} \rightarrow \mathcal{I}$ that maps $\iota = [\![q, i, j, 0]\!]$ to $f(\iota) = [q, i, j]$. This mapping is trivially bijective, and it is easy to see that deductions are preserved: the deduction $\iota_1 \iota_2 \vdash \iota_c$ can be made by a step from D'_i if and only if the deduction $f(\iota_1) f(\iota_2) \vdash f(\iota_c)$ can be made by a step from D_i. Moreover, an item $f(\iota)$ contains a tree t if and only if ι contains the approximate tree $(t, 0)$, so $f(\iota)$ is a final item in the standard parser if and only if ι is a final item in the error-repair parser. Since any correct final item of the form $[\![q, i, j, 0]\!]$ in the error-repair parser is $f^{-1}(\kappa)$ for some correct final item $\kappa = [q, i, j]$ in the standard parser, and we know by hypothesis that the standard parser is complete, it follows that all final items with distance 0 are valid in our error-repair parser.

6.2.2. Induction step

Supposing that the proposition holds for a distance value e, we must prove that it also holds for $e + 1$.

Let $[\![q, 0, n, e+1]\!]$ be a correct final item for the string $a_1 \dots a_n$. We will prove that this item is valid in the deduction system $(\mathcal{I}', \mathcal{K}(a_1 \dots a_n), D')$.

As this item is correct for the string $a_1 \dots a_n$, we know that it contains an approximate tree $(t, e+1)$ where t is a tree rooted at S with $yield(t) = \underline{a}_1 \dots \underline{a}_n$. By definition of approximate tree, we know that there exists a tree $u \in \mathit{Trees}(G)$ such that $\hat{d}(t,u) = e+1$ or, equivalently, $t \in \mathit{Trans}_{e+1}(u)$.[13]

By definition of $\mathit{Trans}_{e+1}(u)$, this implies that there is another tree t' such that $t' \in \mathit{Trans}_e(u)$ and $t \in \mathit{Trans}_1(t')$, and this implies that there exists an approximate tree (t', e) such that $\hat{d}(t,t') = 1$.

Since $\hat{d}(t,t') = 1$, and $\mathit{yield}(t) = \underline{a}_1 \dots \underline{a}_n$, we know that $t \in \mathit{Substitution}(t') \cup \mathit{Insertion}(t') \cup \mathit{Deletion}(t')$, and therefore $\mathit{yield}(t')$ must be one of the following:

(1) $\underline{a}_1 \dots \underline{a}_{j-1}\ (b,j)\ \underline{a}_{j+1} \dots \underline{a}_n$, if $t \in \mathit{Substitution}(t')$ [14]
(2) $\underline{a}_1 \dots \underline{a}_{j-1}\ (a_{j+1}, j) \dots (a_n, n-1)$, if $t \in \mathit{Insertion}(t')$
(3) $\underline{a}_1 \dots \underline{a}_{j-1}\ \underline{b}_j\ (a_j, j+1)\ (a_{j+1}, j+2) \dots (a_n, n+1)$, if $t \in \mathit{Deletion}(t')$

Induction step, case (1) (substitution error).

Suppose that $\mathit{yield}(t')$ is of the form $\underline{a}_1 \dots \underline{a}_{j-1}\ (b,j)\ \underline{a}_{j+1} \dots \underline{a}_n$. Consider the deduction system $(\mathcal{I}', \mathcal{K}(a_1 \dots a_{j-1}\ b\ a_{j+1} \dots a_n), D')$ obtained by applying our uninstantiated parsing system to the string $a_1 \dots a_{j-1}\ b\ a_{j+1} \dots a_n$. Consider the item in $\mathcal{I}'$ containing the approximate tree (t', e): this item must be of the form $[\![q, 0, n, e]\!]$, since $\hat{d}(t,t') = 1$ and $(t, e+1) \in [\![q, 0, n, e+1]\!]$ (under the distance function $\hat{d}$, if trees in two items $[\![q_1, i_1, j_1, e_1]\!]$ and $[\![q_2, i_2, j_2, e_2]\!]$ are at finite distance, then q_1 must equal q_2).

This item $[\![q, 0, n, e]\!]$ is a correct final item in this system, since t' is a marked parse tree for the input string $a_1 \dots a_{j-1}\ b\ a_{j+1} \dots a_n$. By the induction hypothesis, this item is also valid in this system. If we prove that the validity of this item in the system $(\mathcal{I}', \mathcal{K}(a_1 \dots a_{j-1}\ b\ a_{j+1} \dots a_n), D')$ implies that the item $[\![q, 0, n, e+1]\!]$ is valid in the system $(\mathcal{I}', \mathcal{K}(a_1 \dots a_n), D')$, the induction step will be proved for the substitution case.

Therefore, we have reduced this case of the proof to proving the following lemma:

Lemma 1. *Let $\mathcal{R}(\mathcal{S}) = (\mathcal{I}', \mathcal{K}, \mathcal{D}')$ be the uninstantiated parsing system obtained by applying the error-repair transformation to a sound and complete parsing system $\mathcal{S}$.*

Given a nonempty string $a_1 \dots a_n$, and a string $a_1 \dots a_{j-1}\ b\ a_{j+1} \dots a_n (1 \le j \le n)$ obtained by substituting the jth terminal in the first string.

If $[\![q, 0, n, e]\!]$ is a valid item in the instantiated parsing system $\mathcal{R}(\mathcal{S})(a_1 \dots a_{j-1}\ b\ a_{j+1} \dots a_n) = (\mathcal{I}', \mathcal{K}(a_1 \dots a_{j-1}\ b\ a_{j+1} \dots a_n), D')$, then $[\![q, 0, n, e+1]\!]$ is valid in the instantiated parsing system $\mathcal{R}(\mathcal{S})(a_1 \dots a_n) = (\mathcal{I}', \mathcal{K}(a_1 \dots a_n), D')$.

In order to prove this lemma, we define a function $f_1 : \mathcal{I}' \to \mathcal{I}'$ as follows:

$f_1([\![q, i, k, e]\!]) = [\![q, i, k, e]\!]$ if $i > j-1$ or $k < j$

$f_1([\![q, i, k, e]\!]) = [\![q, i, k, e+1]\!]$ if $i \le j-1$ and $j \le k$

We will prove that if $\iota_1, \iota_2, \dots \iota_a \vdash \iota_c$ in the instantiated parsing system

[13] Note that, strictly speaking, the definition of approximate tree only guarantees us that $\hat{d}(t,u) \le e+1$, rather than strict equality. However, this is not relevant for the proof: if $d(t,u) < e+1$, we would have that $[\![q, 0, n, d(t,u)]\!]$ is a correct final item, and thus valid by induction hypothesis, and we conclude that $[q, 0, n, e+1]$ is valid by applying $D^{DistIncr}$ steps.

[14] As our definition of $\hat{d}$ ignores indexes associated to marked terminals, we can safely assume that the marked terminal inserted in the frontier has the index j. In the other cases, we follow the same principle to reindex the marked terminals.

$(\mathcal{I}', \mathcal{K}(a_1 \dots a_{j-1}\ b\ a_{j+1} \dots a_n), D')$, then $\mathcal{K}(a_1 \dots a_n) \cup \{f_1(\iota_1), f_1(\iota_2), \dots f_1(\iota_a)\} \vdash^* f_1(\iota_c)$ in the instantiated parsing system $(\mathcal{I}', \mathcal{K}(a_1 \dots a_n), D')$.
We say that $\iota_1, \iota_2, \dots \iota_a \vdash \iota_c$ in some instantiated parsing system if ι_c can be obtained from $\iota_1, \iota_2, \dots \iota_a$ by application of a single deduction step. Therefore, we will prove the implication by considering all the possible deduction steps with which we can perform such a deduction:

- $D'^{CorrectHyp}$

  If $\iota_1, \iota_2, \dots \iota_a \vdash \iota_c$ by a $D'^{CorrectHyp}$ step, then $a = 1, \iota_1 = [\![a, x-1, x, 0]\!]$ and $\iota_c = [\![a, x-1, x, 0]\!]$. If we compute $f_1(\iota_1)$ and $f_1(\iota_c)$ depending on the values of the indexes i, j, we obtain that:

  if $x \neq j$, $f_1(\iota_1) = [\![a, x-1, x, 0]\!]$ and $f_1(\iota_c) = [\![a, x-1, x, 0]\!]$

  if $x = j$, $f_1(\iota_1) = [\![a, x-1, x, 1]\!]$ and $f_1(\iota_c) = [\![a, x-1, x, 1]\!]$

  In both cases we have that $\mathcal{K}(a_1 \dots a_n) \cup \{f_1(\iota_1)\} \vdash^* f_1(\iota_c)$, because $f_1(\iota_1) = f_1(\iota_c)$.

- $D'^{SubstHyp}$

  By reasoning analogously to the previous case, we obtain:

  if $x \neq j$, $f_1(\iota_1) = [\![a, x-1, x, 0]\!]$ and $f_1(\iota_c) = [\![b, x-1, x, 1]\!]$

  if $x = j$, $f_1(\iota_1) = [\![a, x-1, x, 1]\!]$ and $f_1(\iota_c) = [\![b, x-1, x, 2]\!]$

  In the first case, we have that we can infer $f_1(\iota_c)$ from $f_1(\iota_1)$ by a $D'^{SubstHyp}$ step. In the second case, we can infer $f_1(\iota_c)$ from $\mathcal{K}(a_1 \dots a_n)$: if we take the hypothesis $[\![a_x, x-1, x, 0]\!] = [a_x, x-1, x] \in \mathcal{K}(a_1 \dots a_n)$, we can infer $\iota_t = [\![b, x-1, x, 1]\!]$ from it by using a $D'^{SubstHyp}$ step, and then infer $f(\iota_c) = [\![b, x-1, x, 2]\!]$ from ι_t by using a $D'^{DistIncr}$ step.

- D'^{DelHyp}

  In this case, we always have that $\iota_c = [\![b, x, x, 1]\!]$ and $f_1(\iota_c) = [\![b, x, x, 1]\!]$, and therefore $f_1(\iota_c)$ can be inferred directly from the empty set by a D'^{DelHyp} step.

- D'^{InsHyp}

  In this case, we have:

  if $x \neq j$, $f_1(\iota_1) = [\![a, x-1, x, 0]\!]$ and $f_1(\iota_c) = [\![\epsilon, x-1, x, 1]\!]$

  if $x = j$, $f_1(\iota_1) = [\![a, x-1, x, 1]\!]$ and $f_1(\iota_c) = [\![\epsilon, x-1, x, 2]\!]$

  In the first case, we can infer $f_1(\iota_c)$ from $f_1(\iota_1)$ by a D'^{InsHyp} step. In the second case, we can infer $f_1(\iota_c)$ from $\mathcal{K}(a_1 \dots a_n)$: if we take the hypothesis $[\![a_x, x-1, x, 0]\!] = [a_x, x-1, x] \in \mathcal{K}(a_1 \dots a_n)$, we can infer $\iota_t = [\![\epsilon, x-1, x, 1]\!]$ from it by using a D'^{InsHyp} step, and then infer $f(\iota_c) = [\![\epsilon, x-1, x, 2]\!]$ from ι_t by using a $D'^{DistIncr}$ step.

- $D'^{BegInsComb}$

  In the case of $D'^{BegInsComb}$, we have:

  1. if $0 < j \leq i_1$, $f_1(\iota_1) = [\![\epsilon, 0, i_1, e_1 + 1]\!]$, $f_1(\iota_2) = [\![x, i_1, i_2, e_2]\!]$ and $f_1(\iota_c) = [\![x, 0, i_2, e_1 + e_2 + 1]\!]$.

2. if $i_1 < j \leq i_2$, $f_1(\iota_1) = [\![\epsilon, 0, i_1, e_1]\!]$, $f_1(\iota_2) = [\![x, i_1, i_2, e_2 + 1]\!]$, and $f_1(\iota_c) = [\![x, 0, i_2, e_1 + e_2 + 1]\!]$.
3. otherwise, $f_1(\iota_1) = [\![\epsilon, 0, i_1, e_1]\!]$, $f_1(\iota_2) = [\![x, i_1, i_2, e_2]\!]$ and $f_1(\iota_c) = [\![x, 0, i_2, e_1 + e_2]\!]$.

In any of the three cases, $f_1(\iota_c)$ can be inferred from $f_1(\iota_1)$ and $f_1(\iota_2)$ by a $D'^{BegInsComb}$ step.

- $D'^{OthInsComb}$

  Analogous to the previous case.

- $D'^{DistIncr}$

  Reasoning as in the previous cases, we obtain that either $\iota_1 = [\![x, i, j, e]\!]$ and $\iota_c = [\![x, i, j, e + 1]\!]$, or $\iota_1 = [\![x, i, j, e + 1]\!]$ and $\iota_c = [\![x, i, j, e + 2]\!]$. In both cases, the resulting deduction can be performed by a $D'^{DistIncr}$ step.

- D'_i coming from a predictive step set D_i

  Let us consider the case of a step D'_i which comes from a predictive step set D_i. Then D'_i can be written as

  $D'_i = \{[\![q_1, j_1, k_1, e_1]\!], [\![q_2, j_2, k_2, e_2]\!], \ldots, [\![q_n, j_n, k_n, e_n]\!] \vdash$
  $[\![q_c, f(j_1, k_1, j_2, k_2, \ldots, j_n, k_n), f(j_1, k_1, j_2, k_2, \ldots, j_n, k_n), 0]\!]$
  $\mid j_1, \ldots, j_n, k_1, \ldots, k_n, e_1, \ldots, e_n \in \mathbb{N} \wedge j_i \leq k_i \wedge P(q_1, q_2, \ldots, q_n, q_c) = 1\}$

  In this case, we have that

  $f_1(\iota_1) = [\![q_1, j_1, k_1, e_1 + b_1]\!]$

  $f_1(\iota_2) = [\![q_2, j_2, k_2, e_2 + b_2]\!]$

  $\vdots$

  $f_1(\iota_n) = [\![q_n, j_n, k_n, e_n + b_n]\!]$

  where b_i can be either 0 or 1, and $f_1(\iota_c) = [\![q_c, f(j_1, k_1, j_2, k_2, \ldots, j_n, k_n), f(j_1, k_1, j_2, k_2, \ldots, j_n, k_n), 0]\!]$.

  Clearly, $f_1(\iota_c)$ can be inferred from $f_1(\iota_1) \ldots f_1(\iota_n)$ by a D'_i step.

- D'_i coming from a yield union step set D_i

  In the case of a step D'_i coming from a yield union step set D_i in the non-error-repair schema, we can write D'_i as

  $D'_i = \{[\![q_1, i_0, i_1, e_1]\!], [\![q_2, i_1, i_2, e_2]\!], \ldots, [\![q_m, i_{m-1}, i_m, e_m]\!],$
  $[\![c_1, j_1, k_1, e'_1]\!], [\![c_2, j_2, k_2, e'_2]\!], \ldots, [\![c_n, j_n, k_n, e'_n]\!]\} \vdash$
  $[\![q_c, i_0, i_m, e_1 + \ldots + e_m]\!] \mid i_0 \leq i_1 \leq \ldots \leq i_m \in \mathbb{N} \wedge j_1, \ldots, j_n, k_1, \ldots, k_n,$
  $e'_1, \ldots, e'_n, e_1, \ldots, e_m \in \mathbb{N} \wedge j_i \leq k_i \wedge P(q_1, q_2, \ldots, q_m, c_1, c_2, \ldots, c_n, q_c) = 1\}$

  In this case, we have

  $f(\iota_1) = [\![q_1, i_0, i_1, e_1 + b_j(i_0, i_1)]\!]$

  $f(\iota_2) = [\![q_2, i_1, i_2, e_2 + b_j(i_1, i_2)]\!]$

$\vdots$

$f(\iota_m) = [\![q_m, i_{m-1}, i_m, e_m + b_j(i_{m-1}, i_m)]\!]$

$f(\iota_{m+1}) = [\![c_1, j_1, k_1, e'_1 + b_j(j_1, k_1)]\!]$

$\vdots$

$f(\iota_{m+n}) = [\![c_n, j_n, k_n, e'_n + b_j(j_n, k_n)]\!]$,

where $b_j(n_1, n_2)$ is the function returning 1 if $n_1 < j \leq n_2$ and 0 otherwise.

For the consequent, we have that $f(\iota_c) = [\![q_c, i_0, i_m, e_1 + \ldots + e_m + b_j(i_0, i_m)]\!]$.

We have that $b_j(i_0, i_m) = b_j(i_0, i_1) + \ldots + b_j(i_{m-1}, i_m)$, since position j can belong at most to one of the intervals $(i_{w-1}, i_w]$. If it does belong to one of the intervals, it also belongs to $(i_0, i_m]$, so both members of the expression equal one. On the other hand, if it does not belong to any of the intervals $(i_{w-1}, i_w]$, nor can it belong to $(i_0, i_m]$, so both members equal zero.

Therefore, $f(\iota_c)$ can be deduced from $f(\iota_1) \ldots f(\iota_{m+n})$ directly by applying the D'_i step.

With this we have proved that, for any deduction $\iota_1, \iota_2, \ldots \iota_a \vdash \iota_c$ made in the instantiated parsing system $(\mathcal{I}', \mathcal{K}(a_1 \ldots a_{j-1}\ b\ a_{j+1} \ldots a_n), D')$, we have $\mathcal{K}(a_1 \ldots a_n) \cup \{f_1(\iota_1), f_1(\iota_2), \ldots f_1(\iota_a)\} \vdash^* f_1(\iota_c)$ in the instantiated parsing system $(\mathcal{I}', \mathcal{K}(a_1 \ldots a_n), D')$.

This implies that, if $\mathcal{K}(a_1 \ldots a_{j-1}\ b\ a_{j+1} \ldots a_n) \cup \{\iota_1, \iota_2, \ldots \iota_a\} \vdash^* \iota_c$ in $(\mathcal{I}', \mathcal{K}(a_1 \ldots a_{j-1}\ b\ a_{j+1} \ldots a_n), D')$, then $\mathcal{K}(a_1 \ldots a_n) \cup \{f_1(\iota_1), f_1(\iota_2), \ldots f_1(\iota_a)\} \vdash^* f_1(\iota_c)$ in $(\mathcal{I}', \mathcal{K}(a_1 \ldots a_n), D')$. In the particular case where $a = 0$ and $\iota_c = [\![q, 0, n, e]\!]$, we have that $f_1(\iota_c) = [\![q, 0, n, e+1]\!]$ is valid, and therefore this proposition for that particular case is equivalent to Lemma 1. Thus, we have proved the substitution case of the induction step.

Induction step, case (2) (insertion error).

In this case, we have that *yield*$(t') = \underline{a}_1 \ldots \underline{a}_{j-1}(a_{j+1}, j) \ldots (a_n, n-1)$. Following a similar reasoning to that in the previous case, we can reduce this to proving the following lemma.

Lemma 2. *Let $\mathcal{R}(\mathcal{S}) = (\mathcal{I}', \mathcal{K}, \mathcal{D}')$ be the uninstantiated parsing system obtained by applying the error-repair transformation to a sound and complete parsing system $\mathcal{S}$.*

Given a nonempty string $a_1 \ldots a_n$, and a string $a_1 \ldots a_{j-1}a_{j+1} \ldots a_n$ $(1 \leq j \leq n)$ obtained by deleting the jth terminal in the first string.

If $[\![q, 0, n-1, e]\!]$ is a valid item in the instantiated parsing system $(\mathcal{I}', \mathcal{K}(a_1 \ldots a_{j-1}a_{j+1} \ldots a_n), D')$, then $[\![q, 0, n, e+1]\!]$ is valid in the instantiated parsing system $(\mathcal{I}', \mathcal{K}(a_1 \ldots a_n), D')$.

The proof, which we shall not detail, is also analogous to that of the previous case. In this case, the function that we use to map items and deductions in

$(\mathcal{I}', \mathcal{K}(a_1 \ldots a_{j-1}a_{j+1} \ldots a_n), D')$ to those in $(\mathcal{I}', \mathcal{K}(a_1 \ldots a_n), D')$ is the function f_2 defined by:

$f_2([\![q, i, k, e]\!]) = [\![q, i, k, e]\!]$ if $j > k$

$f_2([\![q, i, k, e]\!]) = [\![q, i, k+1, e+1]\!]$ if $j > i$ and $j \leq k$

$f_2([\![q, i, k, e]\!]) = [\![q, i+1, k+1, e]\!]$ if $j \leq i$

Induction step, case (3) (deletion error).

Reasoning as in the previous cases, we can reduce this case to the following lemma.

Lemma 3. *Let $\mathcal{R}(\mathcal{S}) = (\mathcal{I}', \mathcal{K}, \mathcal{D}')$ be the uninstantiated parsing system obtained by applying the error-repair transformation to a sound and complete parsing system $\mathcal{S}$.*

Given a string $a_1 \ldots a_n$, and a string $a_1 \ldots a_{j-1}\ b\ a_j a_{j+1} \ldots a_n$ $(1 \leq j \leq n)$ obtained by inserting a terminal b in position j of the first string.

If $[\![q, 0, n+1, e]\!]$ is a valid item in the instantiated parsing system $(\mathcal{I}', \mathcal{K}(a_1 \ldots a_{j-1}\ b\ a_j a_{j+1} \ldots a_n), D')$, then $[\![q, 0, n, e+1]\!]$ is valid in the instantiated parsing system $(\mathcal{I}', \mathcal{K}(a_1 \ldots a_n), D')$.

This lemma can be proved by using the same principles as in the previous ones, and the following function f_3:

$f_3([\![q, i, k, e]\!]) = [\![q, i, k, e]\!]$ if $j > k$

$f_3([\![q, i, k, e]\!]) = [\![q, i, k-1, e+1]\!]$ if $j > i$ and $j \leq k$

$f_3([\![q, i, k, e]\!]) = [\![q, i-1, k-1, e]\!]$ if $j \leq i$

6.2.3. End of the proof of Theorem 2

This concludes the proof of the induction step for Proposition 1 and, therefore, it is proved that our error-repair transformation preserves completeness (Theorem 2).

7. Optimization techniques

The error-repair transformation that we have defined allows us to obtain error-repair parsers from non-error-repair ones; and we have formally shown that the error-repair parsers obtained by the transformation are always correct if the starting parser satisfies certain conditions, which are easy to verify for widely known parsers such as CYK, Earley or Left-Corner.

However, as we can see in the example obtained by transforming the Earley parser, the extra steps generated by our transformation make the semantics of the resulting parser somewhat hard to understand, and the *SubstHyp* and *DelHyp* steps would negatively affect performance if implemented directly in a deductive engine. Once we have used our transformation to obtain a correct error-repair parser, we can apply some simplifications to it in order to obtain a simpler, more efficient one which will generate the same items except for the modified hypotheses. That is, we can bypass items of the form $[a, i, j, e]$. In order to do this:

- We remove the steps that generate items of this kind.
- For each step requiring an item of the form $[a, i, j, e]$ as an antecedent, we change this requirement to the set of hypotheses of the form $[b, i1, i2]$ needed to generate such an item from the error hypothesis steps.

Example 12. Given the $D'^{Scanner}$ step obtained by transforming an Earley *Scanner* step

$D'^{Scanner} = \{[A \to \alpha \bullet a\beta, i, j, e_1], [a, j, k, e_2] \vdash [A \to \alpha a \bullet \beta, i, k, e_1 + e_2]\}$

we can make the following observations:

- The item $[a, j, k, e_2]$ can only be generated from error hypothesis steps if $e_2 = k - j$, $e_2 = k - j - 1$ or $e_2 = k - j + 1$. It is trivial to see that the hypothesis steps added by the transformation always preserve this property. Therefore, we can separately consider each of these three cases.
- The item $[a, j, k, k - j]$ is valid if and only if $k > j$. This item can be obtained by combining a substitution hypothesis $[b, j, j+1, 1]$ with $k-j-1$ insertion hypotheses $[\epsilon, j+1, j+2, 1]$, $\ldots$, $[\epsilon, j+(k-j-1), j+(k-j), 1]$ via *OtherInsertionCombiner* steps.
- The item $[a, j, k, k-j+1]$ is valid if and only if $k \geq j$. This item can be obtained by combining a deletion hypothesis $[b, j, j, 1]$ with $k - j$ insertion hypotheses $[\epsilon, j, j + 1, 1]$, $\ldots$, $[\epsilon, j + (k - j - 1), j + (k - j), 1]$ via *OtherInsertionCombiner* steps.
- The item $[a, j, k, k - j - 1]$ is valid if and only if one of the following holds:
  1. $j = 0$ (therefore our item is $[a, 0, k, k - 1]$, and thus $k > 0$), and we have the hypothesis $[a, w - 1, w]$ for $w \leq k$. In this case, the item $[a, 0, k, k - 1]$ can be obtained by applying the *Combiner* steps to a correct hypothesis and $k - 1$ insertion hypotheses: $[\epsilon, 0, 1, 1]$, $[\epsilon, 1, 2, 1]$, $\ldots$, $[a, w - 1, w, 0]$, $[\epsilon, w, w + 1, 1]$, $\ldots$, $[\epsilon, k - 1, k, 1]$.
  2. $j > 0$ and we have the hypothesis $[a, j, j+1]$. In this case, the item $[a, j, k, k-j-1]$ (obviously, k must be $\geq j+1$) can be obtained by applying the *Combiner* steps to a correct hypothesis and $k - 1$ insertion hypotheses: $[a, j, j + 1, 0]$, $[\epsilon, j + 1, j + 2, 1]$, $\ldots$, $[\epsilon, k - 1, k, 1]$.

Therefore, we can change the $D'^{Scanner}$ step to the following set of steps:

- For $e_2 = k - j$:

  $D'^{GeneralSubsScan} = \{[A \to \alpha \bullet a\beta, i, j, e] \vdash [A \to \alpha a \bullet \beta, i, k, e + k - j] \ / \ k \geq j + 1\}$
- For $e_2 = k - j + 1$:

  $D'^{GeneralDeleScan} = \{[A \to \alpha \bullet a\beta, i, j, e] \vdash [A \to \alpha a \bullet \beta, i, k, e + k - j + 1] \ / \ k \geq j\}$
- For $e_2 = k - j - 1$ and $j = 0$:

  $D'^{GeneralScan1} = \{[A \to \alpha \bullet a\beta, 0, 0, e][a, w - 1, w] \vdash [A \to \alpha a \bullet \beta, 0, k, e + k - 1] \ / \ 0 < w \leq k\}$

- For $e_2 = k - j - 1$ and $j > 0$:

  $D'^{GeneralScan2} = \{[A \rightarrow \alpha \bullet a\beta, i, j, e], [a, j, j+1] \vdash [A \rightarrow \alpha a \bullet \beta, i, k, e + k + j - 1] \;/\; k \geq j+1\}$

Note that *GeneralSubsScan* is equivalent to Lyon's *ScanSubstituted* in the particular case that $k = j + 1$. Similarly, *GeneralDeleScan* is equivalent to Lyon's *ScanDeleted* when $k = j$, and the *GeneralScan*s are equivalent to Lyon's *Scanner* when $k = 1$ and $k = j + 1$ respectively.

Insertions are repaired for greater values of k: for example, if $k = j + 3$ in *GeneralSubsScan*, we are supposing that we scan over a substituted symbol and two inserted symbols. The order of these is irrelevant, since the same consequent item would be obtained in any of the possible cases.

In the case of the last two steps, we are scanning over a correct symbol and $k-(j+1)$ inserted symbols. In this case order matters, so we get two different steps: *GeneralScan*1 is used to scan any symbols inserted before the first symbol, to scan the first symbol, and to scan any symbols inserted between the first and the second symbol of the string. *GeneralScan*2 is used to scan any symbol in the input string and the symbols inserted between it and the next one. ■

Additionally, as mentioned above, the $D'^{DistIncr}$ can be removed from the transformation in practice. This step is needed if we are interested in completeness with respect to the full set of correct final items, but, since it increases the distance measure without modifying any tree, it is unnecessary if we are only interested in minimal-distance parses, as is usually the case in practice. A similar reasoning can be applied to constrain $D'^{GeneralDeleScan}$ to the case where $k = j$.

Example 13. With these simplifications, the parser obtained from transforming the *Earley* parsing schemata has the following deduction steps:

$D'^{Initter} = \{\vdash [S \rightarrow \bullet\gamma, 0, 0, 0] \;/\; S \rightarrow \gamma \in P\}$

$D'^{Completer} = \{[A \rightarrow \alpha \bullet B\beta, i, j, e_1], [B \rightarrow \gamma\bullet, j, k, e_2] \vdash [A \rightarrow \alpha B \bullet \beta, i, k, e_1 + e_2]\}$

$D'^{Predictor} = \{[A \rightarrow \alpha \bullet B\beta, i, j, e] \vdash [B \rightarrow \bullet\gamma, j, j, 0] \;/\; B \rightarrow \gamma \in P\}$

$D'^{GeneralSubsScan} = \{[A \rightarrow \alpha \bullet a\beta, i, j, e] \vdash [A \rightarrow \alpha a \bullet \beta, i, k, e + k - j] \;/\; k \geq j + 1\}$

$D'^{GeneralDeleScan} = \{[A \rightarrow \alpha \bullet a\beta, i, j, e] \vdash [A \rightarrow \alpha a \bullet \beta, i, j, e + 1]\}$

$D'^{GeneralScan1} = \{[A \rightarrow \alpha \bullet a\beta, 0, 0, e][a, w-1, w] \vdash [A \rightarrow \alpha a \bullet \beta, 0, k, e + k - 1] \;/\; 0 < w \leq k\}$

$D'^{GeneralScan2} = \{[A \rightarrow \alpha \bullet a\beta, i, j, e], [a, j, j+1] \vdash [A \rightarrow \alpha a \bullet \beta, i, k, e + k + j - 1] \;/\; k \geq j + 1\}$

This algorithm is a variant of Lyon's parser that generates the same set of valid items, although inference sequences are contracted because a single *GeneralScan* step can deal with several inserted characters. ■

Example 14. If we apply the same ideas to a CYK bottom-up parser, we obtain an error-repair parser with the following deduction steps:

$D'^{Binary} = \{[B, i, j, e_1], [C, j, k, e_2] \vdash [A, i, k, e_1 + e_2] \ / \ A \rightarrow BC \in P\}$

$D'^{SubsUnary} = \{\vdash [A, j, k, k - j] \ / \ A \rightarrow a \in P \wedge k \geq j + 1\}$

$D'^{DeleUnary} = \{\vdash [A, j, j, 1] \ / \ A \rightarrow a \in P\}$

$D'^{GenUnary1} = \{[a, w - 1, w] \vdash [A, 0, k, k - 1] \ / \ A \rightarrow a \in P \wedge 0 < w \leq k\}$

$D'^{GenUnary2} = \{[a, j, j + 1] \vdash [A, j, k, k - j - 1] \ / \ A \rightarrow a \in P \wedge k \geq j + 1\}$ ■

8. Conclusions

In this article, we have presented a deductive formalism, based on Sikkel's parsing schemata, that can be used to describe, analyze and compare robust parsers based on the error-repair paradigm.

By using this formalism, we have defined a transformation that can be applied to standard parsers in order to obtain robust, error-repair parsers. We have formally proved that the parsing algorithms obtained are correct if the original algorithm satisfies certain conditions. These conditions are weak enough to hold for well-known parsing schemata such as those for Earley, CYK or Left-Corner parsers.

The transformation is completely systematic, enabling it to be applied automatically by a parsing schemata compiler (as the one described in [16, 19]). This means that, by providing such a system with a description of a standard parsing schema, we can automatically obtain a working implementation of an error-repair parser.

In this sense, note that parsing schemata are abstract descriptions of the semantics of parsing algorithms, and the same parsing schema can often be implemented in different ways. If we execute the schemata in this article with a simple deductive engine as described in [15], what we obtain are *global* error-repair parsers: algorithms that find all the minimal final items, but require us to suppose that errors may be located at any position in the input. This causes these parsers to execute many instances of error-repair steps, leading to inefficiency. However, when implementing the schemata, we can modify the deductive engine to implement heuristic searches that greatly increase efficiency at the cost of not always obtaining *all* the solutions. This leads to *regional* and *local* error-repair strategies [9], which execute error-repair steps only when needed and have only a small performance penalty when compared to non-error-repair parsers. As these strategies can be obtained from generic modifications of a deductive parsing engine, our transformation allows a parsing schemata compiler [16, 19] to generate global, regional or local error-repair parsers from the same standard parsing schema. Empirical performance results comparing the performance of global and regional implementations of error-repair parsing schemata obtained by compilation, using ungrammatical sentences taken from natural language corpora, can be found in [20] and [21, section 6.5].

This makes our transformation a useful tool for prototyping and testing different robust parsers for practical applications.

Although the focus of this article has been on context-free grammar parsers, the ideas behind the transformation are generic enough to be applied to other constituency-based formalisms, such as tree adjoining grammars.

Acknowledgements

The work reported in this article has been supported in part by the Spanish *Ministerio de Educación y Ciencia* and FEDER (grants HUM2007-66607-C04-02 and HUM2007-66607-C04-03) and the *Xunta de Galicia* (grants PGIDIT07SIN005206PR, INCITE08-PXIB302179PR, INCITE08E1R104022ES, INCITE08ENA166098ES, Rede Galega de Procesamento da Linguaxe e Recuperación de Información, Rede Galega de Lingüística de Corpus.)

References

[1] W. Kasper, B. Kiefer, H. U. Krieger, C. J. Rupp, K. L. Worm, Charting the depths of robust speech parsing, in: Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics, Association for Computational Linguistics, Morristown, NJ, USA, 1999, pp. 405–412. doi:http://dx.doi.org/10.3115/1034678.1034741.

[2] D. Grune, C. J. Jacobs, Parsing Techniques. A Practical Guide — Second edition, Springer Science+Business Media, 2008.

[3] P. van der Spek, N. Plat, C. Pronk, Syntax error repair for a Java-based parser generator, ACM SIGPLAN Notices 40 (4) (2005) 47–50.

[4] R. Corchuelo, J. A. Pérez, A. Ruiz, M. Toro, Repairing syntax errors in LR parsers, ACM Transactions on Programming Langauges and Systems 24 (6) (2002) 698–710.

[5] I.-S. Kim, K.-M. Choe, Error repair with validation in LR-based parsing, ACM Transactions on Programming Languages and Systems 23 (4) (2001) 451–471.

[6] C. Cerecke, Repairing syntax errors in LR-based parsers, Australian Computer Science Communications 24 (1) (2002) 17–22.

[7] G. Lyon, Syntax-directed least-errors analysis for context-free languages: a practical approach, Commun. ACM 17 (1) (1974) 3–14. doi:http://doi.acm.org/10.1145/360767.360771.

[8] C. Mellish, Some chart-based techniques for parsing ill-formed input, in: Meeting of the Association for Computational Linguistics, 1989, pp. 102–109.

[9] M. Vilares, V. M. Darriba, J. Vilares, F. J. Ribadas, A formal frame for robust parsing, Theoretical Computer Science 328 (2004) 171–186.

[10] J. C. Perez-Cortes, J. C. Amengual, J. Arlandis, R. Llobet, Stochastic error-correcting parsing for OCR post-processing, in: ICPR '00: Proceedings of the International Conference on Pattern Recognition, IEEE Computer Society, Washington, DC, USA, 2000, p. 4405.

[11] K. Sikkel, Parsing Schemata — A Framework for Specification and Analysis of Parsing Algorithms, Texts in Theoretical Computer Science — An EATCS Series, Springer-Verlag, Berlin/Heidelberg/New York, 1997.

[12] J. Earley, An efficient context-free parsing algorithm, Communications of the ACM 13 (2) (1970) 94–102.

[13] D. H. Younger, Recognition and parsing of context-free languages in time n^3, Information and Control 10 (2) (1967) 189–208.

[14] T. Kasami, An efficient recognition and syntax algorithm for context-free languages, Scientific Report AFCRL-65-758, Air Force Cambridge Research Lab., Bedford, Massachussetts (1965).

[15] S. M. Shieber, Y. Schabes, F. Pereira, Principles and implementation of deductive parsing, Journal of Logic Programming 24 (1–2) (1995) 3–36.

[16] C. Gómez-Rodríguez, J. Vilares, M. A. Alonso, A compiler for parsing schemata, Software: Practice and Experience 39 (5) (2009) 441–470. doi:10.1002/spe.904.

[17] K. Sikkel, Parsing schemata and correctness of parsing algorithms, Theoretical Computer Science 199 (1–2) (1998) 87–103.

[18] V. I. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals, Soviet Physics Doklady 10 (8) (1966) 707–710.

[19] COMPAS (COMpiler for PArsing Schemata), `http://www.grupolys.org/software/COMPAS/` (2008).

[20] C. Gómez-Rodríguez, M. A. Alonso, M. Vilares, A general method for transforming standard parsers into error-repair parsers, in: A. Gelbukh (Ed.), Computational Linguistics and Intelligent Text Processing, volume 5449 of Lecture Notes in Computer Science, Springer-Verlag, Berlin-Heidelberg-New York, 2009, pp. 207–219.

[21] C. Gómez-Rodríguez, Parsing schemata for practical text analysis, Ph.D. thesis, Universidade da Coruña, available on demand (`cgomezr@udc.es`) (2009).