Fault-tolerant Information Extraction from OCR Medical Images

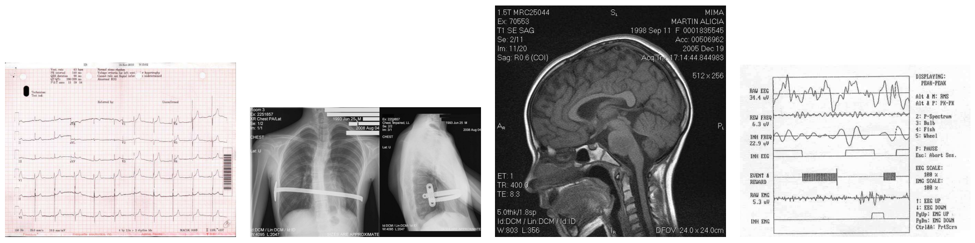
Jinyi Lu[1], Jialing Pei[1], Kenny Q. Zhu[1], Weiguo Gao[2], Jia Wei[2], and Meizhuo Zhang[2]

[1] Shanghai Jiao Tong University
[2] AstraZeneca China

Abstract. Optical character recognition (OCR) [5] [7] has been widely used in digitizing scanned documents into text. Good results have been achieved in applying OCR to textual documents in many languages, such as novels and reports where the content is virtually all words. For semi-structured medical images that combine graphics and words, extraction of useful textual information is difficult due to the discontinuity of text flows in these documents. In this paper, we propose a spatial, fault-tolerant description language that allows users to specify a rough format of the image along with some typing information and constraints. The evaluation engine of the language then parses the semi-structured data according to the description and associates values into different variables in the description while conforming to the type and spatial constraints. This populated parse tree can then be used to extract any desired data values. Moreover, the system guides the user in determining where to make incremental corrections to the parsing results and learns from these corrections to make better parses in the future. Our experimental results show that this system outperforms existing academic and commercial systems by substantial margins in terms of extraction accuracy.

1 Introduction

Due to ever evolving hardware and software, many medical images are directly printed and stored in hard copy formats. Examples are shown in Figure 1. These images often contain a mix of graphics and text, which include technical settings of the hardware used, test measurements or simple diagnoses. Recently, there has been a growing demand for digitizing such medical information from paper media sources. Apart from scanning the graphics into a digital format, extracting the semi-structured textual information is also an important part of building electronic medical records for patients.
Optical character recognition (OCR) is a traditional technique used to turn images of printed text into machine encoded text. It is well researched and performs well on plain text documents such as novels and reports, for a variety of languages.
For a semi-structured medical image, such as Figure 2, we would like to extract the attribute-value pairs (e.g., *Vent. rate = 63 bpm*) and possibly other values such as date (e.g., *18-Nov-2010*) and time (e.g., *9:13:02*)



(a) ECG (b) X-RAY (c) MRI (d) EEG

Fig. 1: Examples of Medical Images

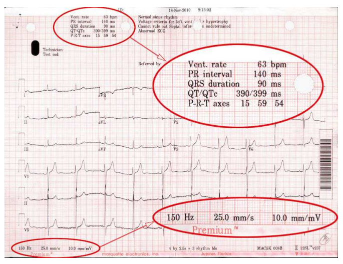


Fig. 2: An ECG Image with Interesting Text Area

```
<p class="ocr_par" title="box 263 33
    444 119">
  <span class="ocr_l" title="box 264
      33 336 45">
    <span class="ocrx_w" title="box
        264 33 299 45">Vcnt.</
      span>
    <span class="ocrx_w" title="box
        308 34 336 45">rule</span
      >
  </span>
  <span class='ocr_l'>
    <span class="ocrx_w" title="box
        264 51 283 64">PR</span>
    <span class="ocrx_w" title="box
        291 51 346 64">Interval</
      span>
    <span class="ocrx_w" title="box
        389 52 411 64">140</span>
    <span class="ocrx_w" title="box
        420 55 439 64">ms</span>
  </span>
  ...
  </span>
</p>
<p class="ocr_p" dir="ltr">
  <span class="ocr_l">
    <span class="ocrx_w" title="box
        396 33 411 45">53</span>
    <span class="ocrx_w" title="box
        420 33 449 48">bpm</span>
  </span>
</p>
```

Fig. 3: Simplified OCR Results in XML for an ECG with Layout Information

since those values endow us with lots of information about the patient. Existing OCR software cannot extract such structured information in a straightforward fashion, but instead it produces rather convoluted results from the whole image, similar to those in Figure 3, which was produced by Tesseract, which is one of the most popular open source multilingual recognizers.

Errors in the OCR text [1] [9] will greatly affect the effectiveness of other related tasks. Much work has been done to improve the performance of the OCR [3]. However, there are still a number of significant challenges involved in extracting the information from medical images or OCR results in XML form.

First, medical images differ from pure text documents in that they contain layout information. Although most current OCR engines attempt to reproduce the physical layout of the text units (along with X-Y coordinates) and store them in a special format such as XML , such spatial information is approximate and sometimes inaccurate, which is why neighboring text blocks in Figure 2, such as "Vent. Rate" and "63 bpm" were not automatically combined into the same XML block, but were rather far apart (shown in two different "classes") in Figure 3 made by OCR softwares. Even for images produced by the same ECG printer, the XML results can still be very different as the spatial layout is sensitive to

many factors, such as accidental spots on the prints, color and contrast, or the angle of the scanner camera.
Second, there are many types of medical images, resulting from a variety of medical tests. Even the same type of test can produce vastly different images due to the use of different equipments. Writing individual extraction wrappers for the OCR outputs of all these formats is tedious and inefficient, not to mention that there are significant programming barriers for writing these wrappers, especially for the medical professionals who are the end users of these extraction results.
Finally, most off-the-shelf OCR programs are pre-trained with specific recognition models, which may not be suitable for the extraction of medical images. Furthermore, changes in imaging equipment technology over time may produce different formats, layout, or terminology, rendering existing OCR models obsolete. Re-training the models requires a large amount of labeled data, which may not be readily available. Incremental training as more labeled data arrives is currently not supported by any OCR product.
In order to solve above problems, We design a system which makes three main contributions:

1. Based on some previous work on spatial data description language [4] [8],we design a new declarative spatial data description language, which borrows its syntax from PADS [2], an ad hoc data processing language, for describing semi-structured data in medical images. We call this language OCR description language, or ODL . ODL descriptions, which are created by some users or specific graphical user interfaces, contains both spatial and data constraints in medical images(Section 2.2).
2. We propose a fuzzy parser which takes OCR results (typically in XML form) and the corresponding description in ODL as input and outputs a data structure with values extracted for each variables in the description . Our fuzzy parser could tolerate the noises and errors in the OCR results as well as inaccuracies of input description (Section 2.3);
3. We propose an incremental correction framework, which makes use of the previous parsers as well as user corrections as labels; such a framework considerably increases the productivity of the extraction from the large number of medical images. (Section 2.4).

2 Approach

Figure 4 shows the framework of our fault-tolerant information extraction system. The input of the system is one or more medical images of the same format (but with some slight variations), possibly after preprocessing. Those images will then be processed by the OCR Engine and will be turned into files in XML form with spatial annotations. Descriptions about those images in ODL will also be created by users or specific graphical user interfaces.
Next, we propose a fuzzy parser to match all the elements on the provided description with the text from the XML data. The parser takes in

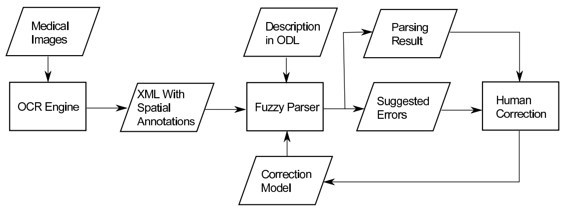


Fig. 4: System Framework

XML data and corresponding ODL description as inputs and produces the parsing results. The parsing results are in the form of parsing trees in which every leaf nodes contains both the description and the corresponding textual information. The word "fuzzy" means that the parser we created can not only extract useful information from those descriptions and XML files, but can tolerate the errors of our inputs as well during the extracting process.

After the fuzzy parsing process, a parsing tree is generated, in which some extracted textual information may contains some OCR errors. In order to correct these errors, we design an incremental correction process. After multiple images are parsed, our system will find some common OCR errors discovered in these images. Our system will provide the user with those common OCR errors, and expect the user to give possible corrections. These corrections would then be used in our correction model and hence improve our parser.

In next few subsections, we will present a running example, using an ECG image, with which we will explain the syntax and semantics of ODL, as well as the interactive human correction mechanism in this framework.

2.1 Running Example

Our running example is the ECG image shown in Figure 2. In this example, we are interested in extracting the time, the measurements, and some basic diagnoses, all highlighted in the bright red ovals. Fragments of the XML results coming out of the OCR engine are shown in Figure 3. In addition, spatial coordinates for different OCR units, like paragraphs, lines and words, are recorded as values for the "title" attribute in XML.

Our discussion on the ODL language will use the abstract syntax instead of concrete syntax. The abstract syntax of running example is shown in Figure 6. In the description, *Osource* is the root description for the entire data of interest. Many types are used in the defination of *Osource* object to represent different forms of the textual information, for example, *Ostruct* is a type that represents the information with multiple fileds, and *Ounion* is a type that represents the enumerable information.

Those descriptions is then used as input for a fuzzy parser that produces a parsing representation, as shown in Figure 5. If the system detects a popular OCR error, it may prompt the user for a manual correction.

```
time                          inter
  ⟨day, F, "18"⟩                ⟨"Abnormal ECG", F⟩
  ⟨"−", F⟩
  month                       para
    ⟨str, F, "Jan"⟩             ⟨p1, E, "15o"⟩
  ⟨"−", F⟩                      ⟨"Hz", F⟩
  ⟨year, F, "2012"⟩             ⟨p2, F, "25.0"⟩
                                ⟨"mm/s", F⟩
triple                          ⟨p3, E, "1o.o"⟩
  ⟨"Vent.rate", F⟩              ⟨"mm/mV", F⟩
  ⟨skip1, F⟩
  ⟨x, F, "65"⟩
  ⟨"bpm", F⟩
  ⟨skip2, F⟩
```

Fig. 5: Results of Our System

```
{
    {
        day(int, 1, 31),
        "−",
        {
            num(int, 1, 12) |
            {
                "Jan" | "Feb" | "
                    Mar" |
                "Apr" | "May" | "
                    Jun" |
                "Jul" | "Aug" | "
                    Sept" |
                "Oct" | "Nov" | "
                    Dec"
            }
        }
        "−",
        year(int)
    }(<_,_,_,0.3l>),

    {
        "Vent. rate",
        hskip \s,
        x(int, 60, 100),
        "bpm",
    }(<0.1w,_,0.5w,_>) as triple,

    {
        "Normal ECG" |
        "Abnormal ECG"
    }(<triple.x1,triple.y0,_,_>),

    vskip \n list

    {
        p1(int),
        "Hz",
        p2(float, 3, 1),
        "mm/s",
        p3(float, 3, 1),
        "mm/mV"
    }(<_,_,_,_>)
}
```

Fig. 6: ODL Description in Abstract Syntax

2.2 Syntax

The abstract syntax of ODL is formally described in Figure 7. The first six types in Figure 7 are basic types. ODL can be used to describe both semi-structured data and spatial information for the physical layout. Next we will discuss the syntax and type system in more detail.

Values Basically, all expressions in ODL can be parsed into values. Values in ODL can take the form of extracted textual information or the spatial parameters. We can see that a value can take on many different shapes in Figure 7. For example, *()* can be regarded as a value. *int*, *float* and *string* can also be regarded as values. *len* and *coor* are two special vlaue types in ODL. They indicate the values of spatial parameters. *len* contains two parameters: the length and the unit for the spatial description. *coor* indicate values of those coordinates, and it contains four *len* values representing the coordinates of the top left corner and the bottom right corner of the region of interest (see it in Figure 7). The last definition for value $\{v_1, ..., v_n\}$ indicates that a struct which contains many values can also be defined as a value.

Primitive Expression Primitive expressions include *constant* and *name* , which are the first two definitions for expressoin. Constants represent fixed-valued strings or numerical values to be recognized from the image. In Figure 6, the key name "Vent. rate" and the unit "bpm" for the medical test are constant expressions
On the other hand, if a data field has variable values, but we know the data type or the constraints of the data field, (e.g., a numeric type), we can use a *name.* In Figure 6, the value for "Vent. rate" is a variable by the patient and their different medical tests, we use a name x to represent that. In addition, we know that the value for "Vent. rate" should be an integer between 60 and 100, so we combine *name* and *constraints* to describe the whole field.

$$
\begin{array}{rcll}
\text{int} & ::= & \wedge -?\backslash d + \$ & \\
\text{float} & ::= & \wedge\,(-?\backslash d+)(.\backslash d+)?\$ & \\
\text{num} & ::= & int|float & \\
\text{len} & ::= & num \;\; (pixel|cm) & \\
\text{coor} & ::= & \langle len_1, len_2, len_3, len_4\rangle & \\
\text{bop} & ::= & + \mid - \mid * \mid / & \\
 & \mid & = \mid != & \\
 & \mid & < \mid > \mid <= \mid >= & \\
\text{v} & ::= & () & \\
 & & \mid int & \\
 & & \mid float & \\
 & & \mid string & \\
 & & \mid len & \\
 & & \mid coor & \\
 & & \mid \{v_1, \ldots, v_n\} & \\
\text{e} & ::= & c & \text{(constant)} \\
 & & \mid x & \text{(name)} \\
 & & \mid \mathrm{e}_0(e_1, e_2, \ldots, e_n) & \text{(constraints)} \\
 & & \mid hskip \;\; e & \text{(horizontal skip)} \\
 & & \mid vskip \;\; e & \text{(vertical skip)} \\
 & & \mid \{e_1|e_2|\ldots|e_n\} & \text{(union)} \\
 & & \mid \{e_1, \ldots, e_n\} & \text{(struct)} \\
 & & \mid e \;\; list & \text{(list)} \\
 & & \mid e \;\; as \;\; x & \text{(blinding)} \\
 & & \mid e_1 \;\; bop \;\; e_2 & \text{(binary operation)}
\end{array}
$$

Fig. 7: Syntax

$$
\begin{array}{rcl}
\text{t} & ::= & unit \\
 & & \mid int \\
 & & \mid float \\
 & & \mid len \\
 & & \mid \langle len, \;\; len, \;\; len, \;\; len\rangle \\
 & & \mid \{t_1 + t_2 + \ldots + t_n\} \\
 & & \mid \{l_1 : t_1, \;\; \ldots, \;\; l_n : t_n\} \\
 & & \mid t \;\; list
\end{array}
$$

Fig. 8: ODL Data Types

Spatial Expression Spatial description expressions include *hskip e* and *vskip e*. These are essential constructs for ODL to describe spaces in the images. For ODL, an image document can be considered the combination of one or more text areas, or text boxes. In Figure 2, there are four separate text boxes on the image. In ODL, the text boxes are described by using the coordinates of the corners of the box. This coordinates information also serves to define constraints. Text boxes can be specified if we want to describe the layout on large scale. *hskip e* and *vskip e* can be used for describing approximate distances horizontally or vertically. Since spatial information is usually described together with the measurement unit, like pixel or centimeter, the spatial description expression in ODL can only be generated with "len" expressions. In the example, the coordinates of the upper left corner and the bottom right corner for four main parts of the image are outlined in the constraints and the symbol "_" means default, which means the coordinates of the upper left corner and the bottom right corner of the whole document. *hskip* $\backslash s$ indicates that there are some spaces between the key "Vent. rate" and the value x. And *vskip* $\backslash s$ indicates that there are some vertical spaces between information about interpretation and parameters.

Composition Composition expressions are compound expressions constructed from other expressions. These include *union*, *struct*, *list*, *constraints*, *binding* and *bop*.

Struct is used to describe a text box by using both data description expressions and spatial description expressions. In *struct*, all sub-expressions are described in a sequence, and the default sequence is from left to right and from top to bottom. In the struct description for “tuples”, key, value and unit are listed from left to right. *Union* means there are many potential data or potential spatial operations to consider. The union description for “month” is the enumeration of all abbreviations of the names of the months. *List* is used to indicate that a sequence of similar data or the same spatial operations should be applied multiple times. In the example, *list* is used to indicate that there are many blank lines between descriptions for interpretation and parameters. *Constraints* can be set based on both data description expressions and spatial description expressions. In ODL, constraints can be put on data or a text box. Other expressions like *binding* and *bop* are designed to simplify the description and make ODL more expressive. The function of *binding* is to give a name x to the expression e so that we can reuse the parsing result of e in the latter description.

$$\frac{\Gamma(x) = t}{\Gamma \vdash x : t} \quad \text{(T-VARIABLE)}$$

$$\frac{\Gamma \vdash e_1 : int \quad \Gamma \vdash e_2 : int \quad bop \in \{+, -, *, /, \%\}}{\Gamma \vdash e_1 \ \ bop \ \ e_2 : int} \quad \text{(T-INT ARITH)}$$

$$\frac{\Gamma \vdash e_1 : int \quad \Gamma \vdash e_2 : int \quad bop \in \{=, != , <, >, <=, >=\}}{\Gamma \vdash e_1 \ \ bop \ \ e_2 : bool} \quad \text{(T-INT REL)}$$

$$\frac{\Gamma \vdash e_1 : float \quad \Gamma \vdash e_2 : float \quad bop \in \{+, -, *, /, \%\}}{\Gamma \vdash e_1 \ \ bop \ \ e_2 : float} \quad \text{(T-FLOAT ARITH)}$$

$$\frac{\Gamma \vdash e_1 : float \quad \Gamma \vdash e_2 : float \quad bop \in \{=, != , <, >, <=, >=\}}{\Gamma \vdash e_1 \ \ bop \ \ e_2 : float} \quad \text{(T-FLOAT REL)}$$

$$\frac{\Gamma \vdash e_0 : t_0}{\Gamma \vdash e_0(e_1, e_2, ..., e_n) : t_0} \quad \text{(T-CONSTRAINT)}$$

$$\frac{\Gamma \vdash e : len}{\Gamma \vdash hskip \ \ e : unit} \quad \text{(T-HSKIP)}$$

$$\frac{\Gamma \vdash e : len}{\Gamma \vdash vskip \ \ e : unit} \quad \text{(T-VSKIP)}$$

$$\frac{for \ \ each \ \ i \ \ \Gamma \vdash e_i : t_i}{\Gamma \vdash \{e_1|...|e_n\} : t_1 + ... + t_n} \quad \text{(T-UNION)}$$

$$\frac{for \ \ each \ \ i \ \ \Gamma \vdash e_i : t_i}{\Gamma \vdash \{e_1, ..., e_n\} : t_1 * ... * t_n} \quad \text{(T-STRUCT)}$$

$$\frac{\Gamma \vdash e : t}{\Gamma \vdash e \ \ list : t \ \ list} \quad \text{(T-LIST)}$$

Fig. 9: Typing Rules

Type System In ODL, we can assign a type to every value and expression. We set up some rules determining types for values and expressions, which make up a complete type system.The type system of ODL is shown in Figure 8 and Figure 9. Figure 8 shows the basic principles for detemining types and Figure 9 extends it by giving some judgement forms which can be seen as inductive typing rules. *T-VARIABLE* indicates that the type of *name* expression is based on the type of name in the typing context. *T-INT ARITH*, *T-INT REL*, *T-FLOAT ARITH* and *T-FLOAT REL* indicate that the two expressions of the *bop* are of the same type, int or float, and the final type of the *bop* expression is based on the binary operation. *T-CONSTRAINT* indicates that the types of the expressions in the constraints have nothing to do with the final type of the *constraints* expression, which is the same as the original expression e_0. *T-HSKIP* and *T-VSKIP* are rules for two spatial expressions. The expression e in *horizontal skip* and *vertical skip* should have spatial parameters and be of the *len* type. The type of these two expressions should be *unit*. The last three of all the typing rules are for the composition expressions. *T-STRUCT* indicates that the type of *struct* is the combination of the types of all sub-expressions. *T-UNION* indicates that the type of *union* can be any type of the sub-expressions. *T-LIST* indicates that the type of *list* is a sequence of the same type.

2.3 Semantics

An in-memory representation is generated for each expression recursively, which eventually builds into a complete parsing tree given an input XML. Due to the fuzzy matching policy and the approximate spatial information in the description, our parser can generate a number of candidate parse trees. The final results will be selected based on an optimization strategy.

Input Data and Parse Result The input of our generated parser is the XML data generated by OCR and corresponding ODL descriptions. We define the result of parsing as a *parse_tree*, which records the ODL expression and corresponding parsing result.

$$\begin{aligned}
\text{text\_box} ::= \;& \{c = \text{coor}, t = v\} \\
\mid \;& \{\} \\
\text{parse\_tree} ::= \;& ((e, text\_box), [parse\_tree_1, \ldots, parse\_tree_n]) \\
\mid \;& () \\
\text{input\_data} ::= \;& \{text\_box \mid \; \forall text\_box \;\; in \;\; XML\}
\end{aligned}$$

Fig. 10: input_data and the parse_tree

Correction Model Besides the input data and the description in ODL, the fuzzy parser also needs a correction model to correct the errors in the input data. The generation of the correction model will be discussed in Section 2.4. Correction model M is a set of correction strategies S. Each correction strategy S is a one-to-many mapping relationship between the original substring in the OCR result and candidate result substrings after correction. Each candidate is attached with a score, which indicates that the probability of this candidate correction result is the right answer for the original substring. If m stands for the original substring and A is the set of the correct candidates and probabilities. , a correction strategy S can be represented as

$$S = \{(m, n, p)| \ \ \forall (n, p) \in A\} \tag{1}$$

These candidates will be further filtered by using the scoring policies discussed below.

Scoring Policy In the fuzzy parser, two scoring policies are designed in order to solve fuzzy matching problems in content-based data description and spatial information description respectively.

Scoring Policy for Data Description Due to the limitations of OCR techniques, data derived from images is not 100% correct. To tolerate the errors and noises in the OCR results, a scoring policy is proposed to take consideration of the constraints and OCR results.
The calculation of error score $es(e, t)$ derived from the edit distance. It uses an expression and a text box content in XML as inputs. Specifically, we use the edit distance result as the error score for the *constant*.

$$es(c, t) = ed(c, t) \tag{2}$$

For the *name* of the integer and floating point type, edit distances are calculated between the potential data and each number which satisfies the range, length or precision information requirements. The smallest edit distance is chosen as the error score. If Z is the set of all integers and R' is the set of all floating points that in length l and precision p, the error score is calculated as:

$$es(x(int, a, b), t) = \min\{ed(i, t)| \ \ \forall i \in Z, i \in [a, b]\} \tag{3}$$

$$es(x(float, l, p, a, b), t) = \min\{ed(i, t)| \ \ \forall i \in R', i \in [a, b]\} \tag{4}$$

Using error score computation method, we can further compute $esm(e, t, M)$ which is the error score with the correction model. We first define $cor(t, S)$, which takes a text box content t and a correction strategy S as inputs. This function will give us a set of candidate corrected text box contents and corresponding probabilities by replacing the substring of t to new candidates according to S. In $cor(t, S)$, $rep(t, m, n) = t'$ means that t' is the result of replacing all occurances of m in the string t with n.

$$cor(t, S) = \{(t', p)| \ \ \forall (m, n, p) \in S, rep(t, m, n) = t'\} \tag{5}$$

Then the error score with correction model $esm(e, t, M)$ will find the minimum product of the error score $es(e, t')$ and the probability p.

$$esm(e, t, M) = \min\{es(e, t') * p| \ \ \forall S \in M, \forall (t', p) \in cor(t, S)\} \tag{6}$$

The total score for the description is calculated by adding all the error scores together. By adding all the error scores together, we can balance the tolerance for errors in a sub expression and the overall sequence.

Scoring Policy for Spatial Description In ODL, spatial description is used to describe the rough spatial relationship based on human estimates. In order to minimize human efforts in estimating the spatial distances, a spatial score $ss(coor, cc)$ is used, which measures the differences between the provided spatial description and data. It takes the coordinates of the text box ($coor$) and the position of the current cursor (cc) as inputs. In the semantics, we record the position of the parsing cursor, which is the current location of the parsing process. For all potential input data, the spatial score is calculated according to the distance between the data coordinates and the cursor.

$$ss(cc, coor) = ||cc - coor|| \tag{7}$$

Evaluation Rules In those previous sections, we introduce some methods of processing the input data in order to tolerate some errors in the input data. In this section, we describe the evaluation rule for generating the parsing result. The result of parsing is a list of all candidate parsing trees. Two scoring policies are used to relieve the computational cost by filtering low probability results. Those specific rules for parsing are shown in Figure 14 in the form of inductive judgement forms. Those functions defined in Figure 13 would be used in the those inductive rules. The *Cons* function is one of the key functions of parsing. It takes the current position of the cursor c, a text box tb and the expression e as the input and return true or false for whether the text box is a suitable candidate for the expression. The function will evaluate the content of the text box based on the scoring policy for data description and the spatial position with the cursor according to the scoring policy for spatial description. A threshold t is set for the sum of the two scores.

$$Cons(tb, c, e) = \begin{cases} \text{T} & esm(e, tb.t, M) + k * ss(c, tb.c) < t \\ \text{F} & \text{otherwise} \end{cases} \tag{8}$$

The *Find* function is designed for finding all potential results that satisfy the scoring policies. The parsing process involves mapping the semi-structured data with the expression. *Filter* is designed to filter the results based on the various constraints. Due to the spatial feature of the semi-structured data, a variable named c is used in the environment to record the position of the cursor. In the judgement form that we designed to represent the parsing process, an ODL expression, provided with the environment E and input data D, will be evaluated into a list of parse trees and remaining data *D'*.

$$\text{Judgment Form}: \quad E, D; e \Downarrow (D'; parse\_tree)list$$

Evaluation rules E-X, E-C1 and E-C2 are used for evaluating variable expression and constant expression respectively. For each expression every given text box will be checked and will be considered as a potential result if it satisfies the constraints. For example, given *constant* “Vent. rate”, current cursor at “(132, 0)” and some potential text boxes “(c=⟨264,33, 336,45⟩, t=“Vcnt. rule”)” and “(c=⟨264,51,439,64⟩, t=“PR

Interval")", the error score for the first text box is much lower compared to the second one. According to E-C1, the first text box will be considered as a potential result for the expression and the second text box will not be a potential result, according to E-C2. For *name* and *constraints*, filtering based on the constraints will take place after generating all the results for the *name*.
Evaluation rule E-Coor, E-Hskip and E-Vskip are used for evaluating spatial operations. E-Coor indicates that the data searching area will be limited when providing coordinate information.
For union, E-Union indicates that each component of the union description will be considered.
Struct and *list* share similar evaluation processes according to E-Sturct and E-List. These rules say that data will be binding to them in sequence.

$$Cons : text\_box * coor * expression \rightarrow bool \qquad Begin : input\_data \rightarrow coor$$
$$Find : input\_data * coor * value \rightarrow input\_data \qquad End : input\_data \rightarrow coor$$

$$Filter : (input\_data * parse\_tree)list * expression \rightarrow (input\_data * parse\_tree)list$$

$$Hskip : input\_data * coor * (input\_data * parse\_tree)list \rightarrow input\_data$$
$$Vskip : input\_data * coor * (input\_data * parse\_tree)list \rightarrow input\_data$$

Fig. 11: Functions in Semantics

2.4 Human Correction

In this section, we describe the human correction process in our system. Based on a parsing tree returned to the user, our system will suggest that users should correct some of errors manually in order to improve the accuracy of the fuzzy parser.

Incremental Learning Model Our correction model is incrementally changed according to the corrections that humans make. The design of the correction model adheres to the scoring policy in Section 2.3.

Initial Model Before the results been corrected by human, the initial model is generated using the candidate results of the OCR engine. To calculate the probabilities of the correction candidates in the initial model, we count the number of occurrences of each correction.

Training From Human Correction After generating the parsing results using the initial model, we have made full use of the OCR engine. To correct the remaining errors, human input is needed. The incremental learning model is also suitable for learning from human correction. For example, if a human corrects the error result "1o.o" to "10.0", we can

$$\frac{}{E, \{\}; x \Downarrow []} \quad \text{(E-Empty)}$$

$$\frac{p \in D \quad E(c) = coor \quad Cons(p, coor, const) = true \quad E, (D - p); x \Downarrow r}{E, D; const \Downarrow ((D - p); (const, p), []) :: r} \quad \text{(E-C1)}$$

$$\frac{p \in D \quad E(c) = coor \quad Cons(p, coor, const) = false \quad E, (D - p); x \Downarrow r}{E, D; const \Downarrow (D; (const, \{\}), []) :: r} \quad \text{(E-C2)}$$

$$\frac{p \in D \quad E(c) = coor \quad E, (D - p); x \Downarrow r}{E, D; x \Downarrow ((D - p); (x, p), []) :: r} \quad \text{(E-X)}$$

$$\frac{E, D; e_0 \Downarrow r_0 \qquad \forall i : r_{i+1} = Filter(r_i, e_i)}{E, D; e_0(e_1, e_2, ..., e_n) \Downarrow r_{n+1}} \quad \text{(E-Constraint)}$$

$$\frac{Find(D, coor, nil) = D' \quad E[c \mapsto Begin(D')], D'; e \Downarrow (D''; t)list}{E, D; e(coor) \Downarrow (D - D' + D''; t)list} \quad \text{(E-Coor)}$$

$$\frac{E, D; e \Downarrow r \quad E(c) = coor \quad Hskip(D, coor, r) = D'}{E, D; hskip \;\; e \Downarrow (D'; []) :: []} \quad \text{(E-Hskip)}$$

$$\frac{E, D; e \Downarrow r \quad E(c) = coor \quad Vskip(D, coor, r) = D'}{E, D; vskip \;\; e \Downarrow (D'; []) :: []} \quad \text{(E-Vskip)}$$

$$\frac{E, D; e_1 \Downarrow r_1 \quad E, D; \{e_2|..|e_n\} \Downarrow r_2}{E, D; \{e_1|e_2|...|e_n\} \Downarrow r_1 + r_2} \quad \text{(E-Union)}$$

$$\frac{E, D; e_1 \Downarrow (E', D'; parse\_tree)list \qquad \forall i : E'_i[c \mapsto End(D - D'_i)], D'_i; \{e_2, ..., e_n\} \Downarrow r_i}{E, D; \{e_1, e_2, ...e_n\} \Downarrow \sum_{i=1}^{m}(D'; parse\_tree)list * r_i} \quad \text{(E-Sturct)}$$

$$\frac{E, D; head \;\; e \;\; list \Downarrow (E', D'; parse\_tree)list \qquad \forall i : E'_i[c \mapsto End(D - D'_i)], D'_i; tail \;\; e \;\; list \Downarrow r_i}{E, D; e \;\; list \Downarrow \sum_{i=1}^{m}(D'; parse\_tree)list * r_i} \quad \text{(E-List)}$$

Fig. 12: Semantics

learn from it that for "o"s in the OCR results, it's possible that they should be corrected as "0"s. So the correction strategy for "o" is modified and "0" is the new correct candidate.

Manual Correction Policy In this section, we describe the policies for recommending errors to be manual corrected. When making use of human correction we find that some errors will have a greater impact on accuracy if they are corrected. The reason is some similar errors occur frequently. Which errors are recommended to a user for correction will affect the accuracy and the number of corrections that the user has to made.

Most Frequent Error Description Elements An efficient way to perform manual correction is to recommend the description elements that contain the most frequent errors. For a set of images in similar formats and the corresponding ODL descriptions, we find out which elements in the description are more likely to be parsed with errors. For those elements, similar errors are more likely to happen since the descriptions for them are the same. In this way, our recommendations can be more accurate than a random recommendation.

3 Experimental Results

3.1 Dataset and Preprocessing

The dataset which we use are from real ECG reports, which are recorded at different times and different hospitals. Those reports can be divided into several different categories. We chose four typical kinds of reports which have lots of images and contain many useful information such as attributes, languages so that we could extract more data from them. The statistics about our dataset are shown in Table 1;
As the examples shown, these ECG images are in different colors and have many noises like grid lines. Because these variations and noises may affect the performance of the OCR engine, we preprocess the images to derive a clean version.

3.2 Extraction Accuracy

Next, we compare our method with three other existing methods. The first and most naive method for information extraction from medical images is to write a simple parser for the XML results of the OCR engine. We consider this approach to be the baseline for evaluation. In this parser, we didn't include any fuzzy matching strategies, but instead extracted all results using exact matches.
The second competing method involves marking all zones of interest on images and getting all the OCR results in them. To adjust the small changes of zone areas between images, a marker zone is set so that all other zones of interest can be adjusted according to it as a reference point. An example image after being marked with the zones of interest and the marker zone is shown in Figure 13 (Zones of interest are in blue and the marker zone is in red).

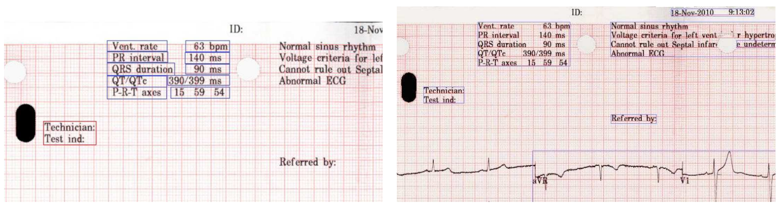


Fig. 13: Other approaches(1.Image Marked With Zones 2.Page Layout Analysis Result)

The third approach is to use the page layout analysis technique. The page layout analysis technique is used to determine where the text resides on a page [6]. By using the page layout analysis technique, the hierarchy of physical components can be generated which we can use to match them to the predefined hierarchy of logical components. An example result of our page layout analysis is shown in Figure 13.

The results of the comparison experiments are shown in Table 2. We only calculated the accuracies for extracting the results of variables since we already know the exact values of constant expressions. Based on our experiment, our method of fuzzy matching outperforms all other methods on all 4 types of ECGs.

Format	1	2	3	4
Number of Images	124	113	102	97
Number of Attributes per Image	17	16	18	15

Table 1: Statistics for The Dataset

Format	1	2	3	4
Exact Match	58.8%	56.3%	61.1%	53.4%
Zonal OCR	81.2%	79.8%	81.7%	80.6%
Page Layout	79.7%	80.2%	81.2%	81.1%
Our Fuzzy Match	**85.5%**	**83.8%**	**84.9%**	**84.0%**

Table 2: Accuracy For Different Methods

3.3 Incremental Manual Correction

We compare two main policies for recommending errors for manual correction, which involve random recommendation and most frequent error description element recommendation. The relationship between the amount of corrections and the accuracy of different types of ECGs are shown in Figure 14. For random correction, we randomly suggest that some errors be corrected each time. The accuracy of random correction is calculated by averaging the results 100 times. For the most frequent error description element recommendation, corrections for most frequently made errors will be suggested first.

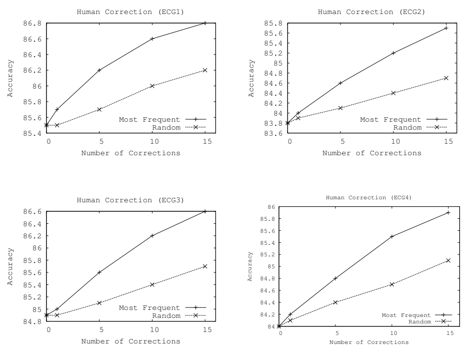


Fig. 14: Comparison of Different Correction Recommendation

As shown in Figure 14, the more corrections we make, the better accuracy we can get. The improvement to the accuracy rate is better when using the most frequent recommendation, compared with random recommendation because using the most frequent recommendation is more efficient in making use of human judgment.

4 Conclusion

We have proposed a new declarative data description language for describing and extracting information from various medical images. Our new language makes use of both data information and spatial information to automatically generate parsers for information extraction from images. Although there are errors in the OCR results, scoring strategies in the parsers make them capable of tolerating errors. We further improve the accuracy of the extracted information with the proposed incremental correction framework. Our evaluation results validated that our proposed methods outperform the other three existing solutions on our real life dataset. Furthermore, our evaluation of the manual correction framework indicates that by using a suitable error correction recommendation system, we can make efficient use of the manual correction and correct more errors with limited manual correction information. In sum, our system can be used to extract information from a large quantity of medical images with simple descriptions and to find an efficient way to make use of human power.

References

1. Darwish, K., Magdy, W.: Error correction vs. query garbling for arabic ocr document retrieval. ACM Transactions on Information Systems (TOIS) 26(1), 5 (2007)
2. Fisher, K., Gruber, R.: PADS: A domain specific language for processing ad hoc data. pp. 295–304 (Jun 2005)
3. Kolak, O., Byrne, W., Resnik, P.: A generative probabilistic ocr model for nlp applications. In: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1. pp. 55–62. Association for Computational Linguistics (2003)
4. Lamport, L., LaTEX, A.: Document Preparation System. Addison-Wesley Reading, MA (1986)
5. Mori, S., Suen, C.Y., Yamamoto, K.: Historical review of ocr research and development. Proceedings of the IEEE 80(7), 1029–1058 (1992)
6. O'Gorman, L.: The document spectrum for page layout analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence 15(11), 1162–1173 (1993)
7. Smith, R.: An overview of the tesseract ocr engine. In: International Conference on Document Analysis and Recognition. vol. 7, pp. 629–633 (2007)
8. Taft, E., Chernicoff, S., Rose, C.: Post-Script Language Reference. Addison-Wesley (1999)
9. Taghva, K., Borsack, J., Condit, A.: Evaluation of model-based retrieval effectiveness with ocr text. ACM Transactions on Information Systems (TOIS) 14(1), 64–93 (1996)