

METHOD FOR INFORMATION EXTRACTION FAILURE DIAGNOSIS AND AUTO-RECOVERY

Field of the Invention

5 This invention relates to a system and a method to extract information using regular expression and refine extraction pattern upon failure to extract information.

Prior Art

 All data analyses begin with data collection and many of which require extraction.
10 Input documents have to be extracted because they might have been created to be consumed by other means (e.g. HTML documents are meant for human readers). Thus they are not in the format of which can be consumed directly by the data analysis process. Very often these input documents' format evolves over time and extraction procedures need to be adapted.

15

 Traditionally, the task of developing and maintaining the extraction falls onto the development team. The debugging and fixing of broken extraction is a laborious process.

 In the paper "Regular expression learning for information extraction" from Li et al,
20 [Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing, pages 21–30, Honolulu, October 2008], their idea is to use regular expression and machine learning technique to find the optimum regular expression to extract the given set of input documents. This approach requires an initial regular expression to be given as the scope of the search space. From which the machine-learning algorithm
25 searches for the goals by applying transformation operation of the regular expression. However, their approach demands that the initial regular expression must already be extracting the required information, which might produce some false positive results.

 In US 8380650, they proposed an invention of information extraction rule making
30 support system. The main objective of their invention is to develop an efficient system, which allows information extraction rules to be derived. The rule making process is based on 1) the training input documents with extraction place indicated, or 2) a set of extraction rules. However, the generated extraction rules might overlap with and subsume with one

another. Further, the extraction rule relation sub system creates a relation network among the extraction rules. Such system favours extraction rules with higher number of results or higher numbers of descendants in the network.

5 Hence, those skilled in the art are striving to provide a system and method that is capable of refining the process of information extraction efficiently.

Summary of the Invention

10 The above and other problems are solved and an advance in the art is made by system and method for extracting information and refining the training data in accordance with this invention. A first advantage of the system and method in accordance with this invention is that the system and method provides a deterministic result. A second advantage of system and method in accordance with this invention is that the system and method does not require any prior training data to function and hence reduces operational
15 cost. A third advantage of the system and method in accordance with this invention is that the system and method diagnoses and recovers from extraction failures, instead of refining/optimising an existing working extraction system. A fourth advantage of the system and method in accordance with this invention is that the system and method considers the user requirements as input and searches for an optimal refactoring procedure
20 to fix the failed regular expression. A fifth advantage of the system and method in accordance with this invention is that the system and method is flexible enough to be integrated in any one of the followings: 1) Direction integration with existing analytics on the source level, 2) binary package library API or 3) a web service interface.

25 In accordance with an aspect of the invention, a system and method for extracting information and refining rules for extracting information is provided in the following manner. The method comprises receiving data, retrieving a regular expression from an extraction pattern database, annotating the regular expression, applying a first phase of error detection extension having a first set of rules by subjecting the data to the annotated
30 regular expression and the first set of rules, generating a match result from the application of the first phase of error detection extension, applying a second phase of error detection extension having a second set of rules by subjecting the data to the annotated regular expression and the second set of rules, generating a mismatch result from the application

of the second phase of error detection extension in response to a mismatch in the data; applying a refinement process to the data in response to generating the mismatch report from the second phase of extraction of information from the data.

5 In accordance with an embodiment of the invention, the step of annotating the regular expression is provided in the following manner. The method determines functions in the regular expression starting from the outermost function to the innermost function, adds parenthesis to each determine function and assigns annotation from left to right in ascending order.

10 In accordance with an embodiment of the invention, the step of applying a first phase of error detection extension comprises: i) determining number of characters in the data; ii) determining number of annotations in the annotated regular expression; iii) selecting a first character; iv) determining a rule among the first set rules based on annotated regular expression and the selected character; v) executing determined rules; vi) generating a regular expression partial derivative and append trace; vii) repeating steps (iv)-(vi) for each character in the data; and viii) generating a match result. The first set of rules are provided in figure 5, where $\bar{i}$ is the trace associated to the source location, r is the regular expression, l is a literal of said data, w is a string of literals of said data; rule (Eps) takes a trace i , an input expression ϵ , an input regular expression literal l , and returns an empty set when there is a mismatch between the input expression ϵ and the regular expression literal l ; rule (Lab1) takes a trace i , an input regular expression literal l , a matching literal expression $i:l$, and append the matching source location i to the trace $\bar{i}$; rule (Lab2) takes a trace i , an input literal l , another literal, literal prime l' and returns an empty set when there is a mismatch between l and l' ; rule (Seq1) takes a trace i , an input regular expression literal l , a first of a sequence of expression r_1 , and return with $\overline{(r'_1, \bar{j})}$, which denotes a set of possible partial derivatives and trace by taking away l from r_1 , and further takes a second of the sequence of expression, r_2 , and return with $\overline{(r'_2, \bar{j})}$, which denotes a set of possible partial derivatives and trace by taking away l from r_2 , said rule (Seq1) is applicable where input expression ϵ is an element of r_1 ; rule (Seq2) takes a trace i , an input literal l , a first of a sequence of expression r_1 , and return with $\overline{(r'_1, \bar{j})}$, which denotes a set of possible partial derivatives and trace by taking away l from r_1 , said rule

(Seq2) is applicable where empty input sequence ϵ is not an element of r_1 ; rule (Choice) takes a trace i , an input literal l , and a first of the two regular expressions r_1 , and returns with $\overline{(r'_1, \bar{j})}$, which denotes a set of possible partial derivatives and trace by taking away l from r_1 and further takes a trace i , an input literal l , and a second of the two regular expressions r_2 , and returns with $\overline{(r'_2, \bar{j}')}$, which denotes a set of possible partial derivatives and trace by taking away l from r_2 ; rule (Star) takes a trace i , a regular expression r , an input literal l , and returns with $\overline{(r', \bar{l}')}$, which denotes a set of possible partial derivatives and trace by taking away l from r' ; rule (WordEps) takes a trace i , a regular expression r , an empty input expression ϵ , and returns $\{(r, \bar{l}')\}$; and rule (WordCon) is separated into two parts, the first part takes a trace i , a regular expression r , a first literal from the string of literal l , and returns $\overline{(r', \bar{l}')}$, the second part takes an appended trace from the first part $\bar{l}'_j$, an intermediate partial derivative regular expression from the first part r'_j , the rest of the string of literal w , and returns with $\overline{(r''_j, \bar{l}''_j)}$, which is the final result for this rule by aggregating across all the j in $\{1 \dots n\}$.

In accordance with an embodiment of the invention, the step of applying the second phase of error detection extension comprises i) determining a rule among the second set rules based on annotated regular expression and the selected character; ii) executing determined rules; iii) generating a regular expression partial derivative and append matching error; iv) repeating steps (i)-(iii) for each character in the data; and v) generating a mismatch result. The second set of rules are provided in figure 7 where the second set of rules is an extension of the first set of rules to track literal mismatch error $E_l^{\bar{l}}$ and the epsilon mismatch error $E_\epsilon^{\bar{l}}$, rule (Eps) raises an epsilon mismatch error, rule (Lab2) raises a literal mismatch error, rule (Lab1) consumes an input literal with the matching literal expression and append the matching source location i to the trace $\bar{l}$, rules (Seq1) and (Seq2) handle sequence expression $r_1 r_2$, rule (Seq1) is applicable where $\epsilon \in r_1$, otherwise rule (Seq2) is applied, rule (Choice) handles choice expression, and rule (Star) handles the Kleene's star expression.

In accordance with an embodiment of the invention, the step of applying the refinement process to said data comprises: i) determining a rule among the third set rules based on annotated regular expression and the selected character; ii) executing determined

rules; iii) generating a regular expression partial derivative and refinement operation; iv) repeating steps (i)-(iii) for each character in the data; and v) applying refinement operations to the regular expression, and vi) updating the refined regular expression in the extraction pattern database.

5

In accordance with an embodiment of the invention, the refinement operation comprises state-extension where a literal is added to the regular expression and the last two literals in the regular expression are finalised, transition-extension where a new transition is added to the regular expression, state-finalisation where the last second literal
10 of the regular expression is finalised, and no-change where no changes are made.

In accordance with an embodiment of the invention, the third set rules are provided in figures 14 and 15 where judgement $R \vdash_A^L r/l \Rightarrow \overline{(R, r, \theta, A)}$ extends from the first set of rules, L denotes the levels of precedence, range over strong (S) and weak (W), A denotes
15 an alternative indication to the source location, θ denotes the refinement environment for each source location, i , and a corresponding refinement operation, op , $i:l +_L^o$ refers to state-extension, $i:l +_L^{\rightarrow}$ refers to transition-extension, $+^\epsilon$ refers to state-finalisation, and id_L refers to no-change, rule (EpsW) proposes a fix to epsilon-literal mismatch by adding a state-extension, in case where the current source location i is not in the domain of the
20 refinement requirement, rule (EpsS) ensures that $(i:r) \in R$ and r can consume the input literal l , rules (LitS1) and (LitW1) handle cases where the input literal matches with the regular expression l where a no-change refinement operation is added, rules (LitS2) and (LitS1) update the user requirement environment R with $(i:r')$ where r' is the partial derivative of r and $(i:r) \in R$, rules (SeqS1), (SeqS2), (SeqW1) and (SeqW2) handle cases
25 where the regular expression to be refined is a concatenation, rules (ChoiceS) and (ChoiceW) handle cases where the regular expression to be refined is a choice, rules (StarS) and (StarW) handle cases where the regular expression to be refined is a kleen's star, and rules (WordCon) and (WordEps) extend the refinement operation to support a sequence of literal input.

30

In accordance with an embodiment of the invention, the step of applying refinement operations to the regular expression comprises applying the refinement operation obtained to the input word w , dividing result from the application of the

refinement operation obtained to the input word into two categories, namely $\bar{\theta}$ and $\bar{\theta}'$ where $\bar{\theta}$ contains refinement environments that makes resulting partial derivative r' final, where $\bar{\theta}'$ contains refinement environments that makes the resulting partial derivative r' non-final, determine the refinement environment from $\bar{\theta} \cup \bar{\theta}'$ with the fewest changes to the original regular expression, and applying the refinement environment with the fewest changes to the original regular expression.

Brief Description of the Drawings

The above and other features and advantages in accordance with this invention are described in the following detailed description and are shown in the following drawings:

Figure 1 illustrating a block diagram showing the flow of data in accordance with an embodiment of this invention;

Figure 2 illustrating a processing system that executes the processes in accordance with an embodiment of this invention;

Figure 3 illustrating a flow diagram of a process for extracting information from data received and refining extraction pattern in accordance with an embodiment of this invention;

Figure 4 illustrating annotation of regular expressions with source locations in accordance with an embodiment of this invention;

Figure 5 illustrating a first phase of a matching algorithm in accordance with an embodiment of this invention;

Figure 6 illustrating an example of the implementation of the first phase of the matching algorithm in accordance with an embodiment of this invention;

Figure 7 illustrating a second phase of a matching algorithm in accordance with an embodiment of this invention;

Figure 8 illustrating an example of the implementation of the second phase of the matching algorithm in accordance with an embodiment of this invention;

Figure 9 illustrating a flow diagram of a process for extracting information using regular expression in accordance with an embodiment of this invention;

Figure 10 illustrating a flow diagram of a process for the first phase of the matching algorithm in accordance with an embodiment of this invention;

Figure 11 illustrating a flow diagram of a process for the second phase of the matching algorithm in accordance with an embodiment of this invention;

Figure 12 illustrating a refinement requirement algorithm in accordance with an embodiment of this invention;

Figure 13 illustrating a sub-matching environment in accordance with an embodiment of this invention;

5 Figures 14 and 15 a refinement algorithm in accordance with an embodiment of this invention;

Figure 16 illustrating the application of refinement environment to a regular expression in accordance with an embodiment of this invention;

10 Figure 17 illustrating the refinement algorithm in pseudo-ML code in accordance with an embodiment of this invention;

Figure 18 illustrating a total order among refinement environments in accordance with an embodiment of this invention;

Figures 19-24 illustrating an example of the implementation of the refinement algorithm in accordance with an embodiment of this invention; and

15 Figure 25 illustrating a flow diagram of a process for refining extraction pattern in accordance with an embodiment of this invention.

Detailed Description

20 This invention relates to a system and a method to extract information using regular expression and refine extraction pattern upon failure to extract information.

25 All data analyses begin with data collection and many of which require extraction. Input documents have to be extracted because they might have been created to be consumed by other means such as HTML documents meant for human readers. Thus they are not in the format of which can be consumed directly by the data analysis process. Very often there input documents' format evolve over time. The extraction procedures need to be adapted.

30 Traditionally, the task of developing and maintaining the extraction process falls onto the development team. The debugging and fixing of the broken extraction is a laborious process. Hence, it is envisioned that a system and a method in accordance with embodiments of this invention may be used for debugging and fixing the extraction

process and thus allowing the development team to handle other more higher value work. Furthermore, the turnaround time of the break-and-fix cycle will be shortened.

The system and method in accordance with embodiments of this invention is based on an elegant process. The system and method are built on a simple and widely used domain specific language, namely, regular expression. By expressing the extraction procedure in terms of regular expressions, we can apply term rewriting techniques such as derivatives and partial derivatives to reason about every single steps of the extraction procedure. In the event of failure, the error location can be easily captured. Together with the user requirement as input, the system is able to make smart decision to reshape the faulty expression based on the user requirement to match the input. From the users' or developers' point of view, all they need to do is to develop the first working version of the extraction procedure and the user requirements and let the system to handle the future changes.

It is noted that the invention is not based on machine learning techniques which demand a huge set of manually labelled training data. Instead, the focus of the invention is on fixing the existing extraction pattern based on user requirement rather than inferring a new extraction pattern based on the training data.

The programming languages that may be used for writing the programme in accordance with embodiments of this invention include Scala, Java and Haskell. Nevertheless, one skilled in the art will recognise that programming languages may be used without departing from the invention. The processes in accordance with embodiments of this invention are executable on the various platforms, namely, Windows, Linux and Mac. Nevertheless, one skilled in the art will recognise that the processes in accordance with embodiments of this invention may also be executable of other platforms without departing from this invention.

Figure 1 illustrates a system for extracting and diagnostic of data in accordance with embodiments of this invention. As shown in figure 1, data 110 retrieved from the World Wide Web (WWW) is being transmitted to the extraction pattern diagnostic subsystem 120 to perform a pattern extraction procedure based on the extraction pattern

rules stored on the extraction pattern database 140. If there is a successful extraction or a match, the findings will be reported. If there is no successful extraction or match, all the failed matches will be reported. Further, if there is no successful extraction or match, the refinement subsystem 130 will be invoked to refine the extraction pattern stored on the extraction pattern database. Further details of the pattern extraction procedure and the refinement procedure will be described below.

Figure 2 illustrates an exemplary processing system 200 that represents the processing systems or subsystems that execute instructions to perform the processes described below in accordance with this invention. One skilled in the art will recognize that the instructions may be stored and/or performed as hardware, firmware, or software without departing from this invention. One skilled in the art will recognize that the exact configuration of each processing system may be different and the exact configuration executing processes in accordance with this invention may vary and processing system 200 shown in figure 2 is provided by way of example only.

Processing system 200 includes Central Processing Unit (CPU) 205. CPU 205 is a processor, microprocessor, or any combination of processors and microprocessors that execute instructions to perform the processes in accordance with the present invention. CPU 205 connects to memory bus 210 and Input/Output (I/O) bus 215. Memory bus 210 connects CPU 205 to memories 220 and 225 to transmit data and instructions between the memories and CPU 205. I/O bus 215 connects CPU 205 to peripheral devices to transmit and receive data between CPU 205 and the peripheral devices. One skilled in the art will recognize that I/O bus 215 and memory bus 210 may be combined into one bus or subdivided into many other busses and the exact configuration is left to those skilled in the art.

A non-volatile memory 220, such as a Read Only Memory (ROM), is connected to memory bus 210. Non-volatile memory 220 stores instructions and data needed to operate various sub-systems of processing system 200 and to boot the system at start-up. One skilled in the art will recognize that any number of types of memory may be used to perform this function.

A volatile memory 225, such as Random Access Memory (RAM), is also connected to memory bus 210. Volatile memory 225 stores the instructions and data needed by CPU 205 to perform software instructions for processes such as the processes for providing a system in accordance with this invention. One skilled in the art will recognize that any number of types of memory may be used to provide volatile memory and the exact type used is left as a design choice to those skilled in the art.

I/O device 230, keyboard 235, display 240, memory 245, network device 250 and any number of other peripheral devices connect to I/O bus 215 to exchange data with CPU 205 for use in applications being executed by CPU 205. I/O device 230 is any device that transmits and/or receives data from CPU 205. Keyboard 235 is a specific type of I/O that receives user input and transmits the input to CPU 205. Display 240 receives display data from CPU 205 and display images on a screen for a user to see. Memory 245 is a device that transmits and receives data to and from CPU 205 for storing data to a media. Network device 250 connects CPU 205 to a network for transmission of data to and from other processing systems.

A system and method for extraction and refinement of extraction patterns processes in accordance with embodiments of the invention will now be described.

Figure 3 illustrates a flow diagram of process 300 performed by the processor in processing system 200 in accordance with an embodiment of this invention. Process 300 is a process for processing the extraction procedure. Process 300 begins with step 305 by receiving data. For purposes of this discussion, data is information received from the WWW. Nevertheless, one skilled in the art will recognise that other types of data can also be used without departing from the invention.

In step 310, an extraction process based on regular expression is being performed on the data received. The regular expression is retrieved from the extraction patterns database 140. Further details of the extraction process executable by the extraction pattern diagnostic subsystem 120 based on regular expressions will be described below with reference to figure 9.

In step 315, if the extraction process is successful, process 300 proceeds to step 325. Otherwise, process 300 proceeds to step 320 to refine the extraction patterns.

In step 320, process 300 refines the regular expression for the data 110 received and stores the revised regular expressions on the extraction pattern database and repeats from step 310. Further details of the refinement process executable by the refinement subsystem 130 based on regular expressions will be described below with reference to figure 25.

In step 325, process 300 generates a report and display on the screen for user. Process 300 ends after step 325.

Further details of the extraction process and refinement of regular expressions will now be described below.

Regular expression

The syntax of regular expression is defined with the following expression.

$$r ::= \epsilon \mid l \mid rr \mid r + r \mid r^*$$

$$w ::= \epsilon \mid lw$$

Where ϵ denotes the language of empty sequence, l denotes a literal, rr denotes concatenation of two regular expressions, $r + r$ denotes a choice of two regular expressions, r^* denotes the Kleene star expression. A word w is either an empty sequence ϵ or non-empty word lw .

Given a word w and a regular expression r , we determine whether w is an element of r , $w \in r$. One of the algorithms proposed by Antimirov is to check via partial derivatives construction. The algorithm proposed by Antimirov is as follows.

Regular Expression Partial Derivatives: $\cdot^p \cdot :: r \rightarrow l \rightarrow \{r\}$

$$\epsilon / ^p l = \{\}$$

$$l_1 / ^p l_2 = \begin{cases} \{\epsilon\} & \text{if } l_1 == l_2 \\ \{\} & \text{otherwise} \end{cases}$$

5

$$(r_1 + r_2) / ^p l = r_1 / ^p l \cup r_2 / ^p l$$

$$(r_1 r_2) / ^p l = \begin{cases} \{(rr_2) | r \in r_1 / ^p l\} \cup r_2 / ^p l & \text{if } \epsilon \in L(r_1) \\ \{(rr_2) | r \in r_1 / ^p l\} & \text{otherwise} \end{cases}$$

$$r^* / ^p l = \{(r' r^*) \mid r' \in r / ^p l\}$$

10

Based on the algorithm proposed by Antimirov, we present the partial derivative operation in terms of a function $\cdot / ^p \cdot :: r \rightarrow l \rightarrow \{r\}$, which takes a regular expression r and a literal l and returns a set of possible residuals of r with respect to literal l . For example, $(aa)^* / ^p a = \{r(aa)^* \mid r \in aa / ^p a\} = \{a(aa)^*\}$

15

Further, using the algorithm proposed by Antimirov, regular expression partial derivatives operation has a few properties as follows

Lemma 1 - Let $r' \in r / ^p l$ and $w \in r'$, then $lw \in r$

20

Lemma 2 - Let $lw \in r$, then there exists $r' \in r / ^p l$ such that $w \in r'$

With the above properties, we can implement a solution to the matching problem using partial derivatives.

$match :: r \rightarrow w \rightarrow Bool$

25

$match\ r\ \epsilon = poseps\ r$

$match\ r\ lw = match'\ r / ^p l\ w$

$match' :: \{r\} \rightarrow w \rightarrow Bool$

$match'\ \{r_1, \dots, r_n\}\ w = (match\ r_1\ w) \text{ or } \dots \text{ or } (match\ r_n\ w)$

$poseps :: r \rightarrow Bool$

30

$poseps\ \epsilon = True$

$poseps\ r^* = True$

$poseps\ l = False$

$poseps\ r1\ r2 = (poseps\ r1) \text{ and } (poseps\ r2)$

$$poseps\ r1+r2 = (poseps\ r1) \text{ or } (poseps\ r2)$$

Given the input regular expression, the match function (and its set variant match') are recursively applied to the word until it becomes the empty sequence.

5

In the empty sequence case, we just need to test whether the regular expression r possesses ϵ .

Extraction process

10

The extraction process is essentially a matching process using regular expression. In particular, we extend the partial derivatives based regular expression matching algorithm to handle error detection.

15

In Figure 4, we annotate the regular expressions structurally with a series of running integers as the source locations. For purposes of this discussion, we use r to denote a regular expression with source locations annotated. In the context of source location annotation process, we often use “function” to refer to regular expression construction by applying regular expression operators such as concatenation, choice and Kleene’s star to smaller regular expressions. The regular expressions are annotated based on the following steps:

20

1. Determine the function from outer to inner. Certain rules applies when determining the function and they are as follows:

25

- a. In cases where two different functions are available, Kleene’s star r^* has the first priority, concatenation rr has the second priority, choice $r+r$ has the third priority.

- b. In cases where 2 identical functions are available, the left function is selected.

30

2. Assign annotation from left to right.

Some examples of the steps of annotation will now be described as follows.

1. Assuming $r = a(b+c)$

Step 1

The first function is $(a(b+c))$ and the second function is, $r+r, (b+c)$. After
5 parenthesis is added for each determined function, $r = (a(b+c))$.

Step 2

1 is assigned to $(a(b+c))$ due to the first function being the left most

2 is assigned to a due to a preceding immediately after the first function

10 3 is assigned to $(b+c)$ due to the second function preceding immediately after a

4 is assigned to b due to the b preceding immediately after the second function

5 is assigned to c due to the c preceding immediately after b

Hence, the end result is $1:2:a3:(4:b + 5:c)$. With parenthesis added for easy
15 reference, $r = 1:(2:a3:(4:b + 5:c))$.

2. Assuming $r = ab+c$

Step 1

20 The first function is $(ab+c)$ and the second function is, $rr, (ab)$. After parenthesis is
added for each determined function, $r = ((ab)+c)$.

Step 2

1 is assigned to $(ab+c)$ due to the first function being the left most

25 2 is assigned to (ab) due to the second function preceding immediately after the
first function

3 is assigned to a due to a preceding immediately after the second function

4 is assigned to b due to b preceding immediately after a

5 is assigned to c due to c preceding immediately after b

30 Hence, the end result is $1:2:3:a4:b+5:c$. With parenthesis added for easy reference,
 $r = 1:(2:(3:a4:b) + 5:c))$.

3. Assuming $r = a(b+c)d$

Step 1

5 The first function is $(a(b+c)d)$. Due to rule (a), the second function is, rr , which in this case is either $a(b+c)$ or $(b+c)d$. However, rule (b) applies and we select $a(b+c)$ as the second function. The third function is, $r+r$, $(b+c)$. After parenthesis is added for each determined function, $r = ((a(b+c))d)$.

Step 2

10 1 is assigned to $(a(b+c)d)$ due to the first function being the left most
2 is assigned to $(a(b+c))$ due to the second function preceding immediately after the first function
3 is assigned to a due to a preceding immediately after the second function
4 is assigned to $(b+c)$ due to the third function preceding immediately after the a
15 5 is assigned to b due to b preceding immediately after the third function
6 is assigned to c due to c preceding immediately after b
7 is assigned to d due to d preceding immediately after c

Hence, the end result is $1:2:3:a4:(5:b+6:c)7:d$. With parenthesis added for easy
20 reference, $r = 1:(2:(3:a4:(5:b+6:c))7:d)$

4. Assuming $r = a(b+c)*d$

Step 1

25 The first function is $(a(b+c)*d)$. Due to rule (a), the second function is, $r*$, $a(b+c)*$. Due to rule (a), the third function is, rr , $a(b+c)$. The fourth function is, $r+r$, $(b+c)$. After parenthesis is added for each determined function, $r = (((a(b+c))*d)$.

Step 2

30 1 is assigned to $(a(b+c)*d)$ due to the first function being the left most
2 is assigned to $(a(b+c)*)$ due to the second function preceding immediately after the first function

3 is assigned to $(a(b+c))$ due to the third function preceding immediately after the second function

4 is assigned to a due to a preceding immediately after the third function

5 is assigned to $(b+c)$ due to the fourth function preceding immediately after a

5 6 is assigned to b due to b preceding immediately after the fourth function

7 is assigned to c due to c preceding immediately after b

8 is assigned to d due to d preceding immediately after c

Hence, the end result is $1:2:3:4:a5:(6:b+7:c)*8:d$. With parenthesis added for easy
10 reference, $r = 1:(2:(3:(4:a5:(6:b+7:c))))*8:d$

For the ease of understanding, we stage the error detection extension into two phases. Figure 5 illustrates the first phase. In this phase, we provide a simple extension to the regular expression matching algorithm using partial derivative. We extend the partial
15 derivative operation to keep track of the trace of the source locations of the literals expression we match, denoted by $\bar{l}$. For purposes of this discussion, we use the deduction system. For instance, the first deduction system is denoted as

$$\bar{l} \vdash r/l \Rightarrow \overline{(r', l')}\tag{1}$$

20

where $\bar{l}$ is the trace associated to the source location, r is the regular expression, l is a literal (i.e. data). Equation (1) can be seen as a function with the following type signature in ML (Meta-language) style.

$$\cdot \vdash \cdot / \cdot \Rightarrow :: \{i\} \rightarrow r \rightarrow l \rightarrow \{(r, \{i\})\}\tag{2}$$

25

Equation (2) reads as follows. The regular expression partial derivative operation takes a trace $\{i\}$, a regular expression r and a literal l as inputs, returns a set of partial derivatives accompanied with the respective traces, $\{(r, \{i\})\}$. The first deduction system,
30 equation (1), applies a set of rules, namely, Eps, Lab1, Lab2, Seq1, Seq2, Choice, and Star, as illustrated in figure 5. The rules are described as follows.

For purposes of this discussion, a bar on top of i (i.e. $\bar{i}$) has the same meaning as a set of i (i.e. $\{i\}$).

Rule (Eps) deals with empty sequence, ϵ . Rule (Eps) reads as follows. Rule (Eps) takes a trace i , an input regular expression ϵ , a literal l as input, and returns an empty set since there is a mismatch between the input expression ϵ and the literal l .

Rule (Lab2) deals with literal, l , where l is not equal to l' . Rule (Lab2) reads as follows. Rule (Lab2) takes a trace i , an input literal regular expression l , another literal, literal prime l' and returns the empty set since there is a mismatch between l and l' . literal l is the first literal of the regular expression while literal prime l' is the first literal from the input word.

Rule (Lab1) deals with input literal, l , at source location i is equal to regular expression l . Rule (Lab1) reads as follows. Rule (Lab1) takes a trace i , an input regular expression literal l , a matching literal regular expression $i:l$, and append the matching source location i to the trace $\bar{i}$, $\{(i:\epsilon, \bar{i} + i)\}$. In this case, the l in $i:l$ is being replaced with epsilon, $i:\epsilon$, due to a match.

Rules (Seq1) and (Seq2) deal with a sequence expression, $r_1 r_2$. Rule (Seq1) is catered for the case where $\epsilon \in r_1$, otherwise rule (Seq2) is used. In rule (Seq1), both r_1 and r_2 are matched with literal l . This is because r_1 may just be accepting an empty sequence. As for rule (Seq2), since r_1 does not accept epsilon, ϵ , we only need to consider matching r_1 to literal l .

Rule (Seq1) reads as follows. Rule (Seq1) is applicable where input expression ϵ is an element of r_1 , $\epsilon \in r_1$. Rule (Seq1) takes a trace i , an input literal l , a first of a sequence of expression r_1 , and return with $\overline{(r'_1, \bar{j})}$, which denotes a set of possible partial derivatives and traces by taking away l from r_1 . Further, Rule (Seq1) takes a trace i , an input literal l , a second of the sequence of expression, namely, r_2 , and return with $\overline{(r'_2, \bar{j})}$, which denotes a set of possible partial derivatives and traces by taking away l from r_2 .

Rule (Seq2) reads as follows. Rule (Seq2) is applicable where empty input sequence ϵ is not an element of r_1 , $\epsilon \notin r_1$. Rule (Seq2) takes a trace i , an input literal l , a first of a sequence of expression r_1 , and return with $\overline{(r'_1, \bar{j})}$, which denotes a set of possible partial derivatives and traces by taking away l from r_1 .

5

Rule (Choice) handles the choice of two regular expressions, $r_1 + r_2$. Rule (Choice) reads as follows. Rule (Choice) takes a trace i , an input literal l , and a first of the two regular expressions r_1 , and returns with $\overline{(r'_1, \bar{j})}$, which denotes a set of possible partial derivatives and traces by taking away l from r_1 . Further, rule (Choice) takes a trace i , an input literal l , and a second of the two regular expressions r_2 , and returns with $\overline{(r'_2, \bar{j}')}$, which denotes a set of possible partial derivatives and traces by taking away l from r_2 . Rule (Choice) considers the possibilities of consuming the input literal l with r_1 and r_2 .

Rule (Star) handles the Kleene's star expression, r^* , where we simply use the regular expression r to consume the input literal, l . Rule (Star) reads as follows. Rule (Star) takes in a trace i , a regular expression r , an input literal l , and returns with $\overline{(r', \bar{l}')}$, which denotes a set of possible partial derivatives and traces by taking away l from r' .

The second deduction system is denoted as

20

$$\bar{i} \vdash r/w \Rightarrow \overline{(r', \bar{l}')} \quad (3)$$

Equation (3) applies the rules for equation (1) that handles the single literal recursively to a sequence of literals. In brief, equation (1) handles a single literal while equation (3) handles a string of literal (i.e. an input word). The second deduction system, equation (3), applies a set of rules, namely, WordEps and WordCon. These rules are described as follows.

Rule (WordEps) deals with empty input sequence, ϵ . Rule (WordEps) reads as follows. Rule (WordEps) takes a trace i , a regular expression r , an empty input expression ϵ , and returns $\{(r, \bar{l}')\}$. In this case, as the input word is empty, there is a definite match. If

30

the regular expression r accepts epsilon, the matching is successful and hence $\{(r, \overline{l'})\}$ is the result.

Rule (WordCon) deals with handling a string of literal, w . Rule (WordCon) reads as follows. Rule (WordCon) is separated into two parts. The first part examines a first literal from the string of literal while the second part examines the rest of the string of literal. The first part takes a trace i , a regular expression r , a first literal from the string of literal l , and returns $\overline{(r', l')}$. The second part takes an appended trace from the first part $\overline{l'}$, an intermediate partial derivative regular expression from the first part r'_j , the rest of the string of literal w , and returns with $\overline{(r_j'', \overline{l_j''})}$, which is the intermediate result obtained by the induction step, which happen to the final result for this rule if we aggregate across all the j in $\{1 \dots n\}$.

Example 1

Consider matching regular expression $r = (a+b)^*$ with the word $w = ab$.

The first step is to annotate the regular expression, r . For $(a+b)^*$, the first function is the Kleene's star, $*$, and the second function is the $r+r$. Thus, 1 is assigned to $(a+b)^*$, 2 is assigned to $(a+b)$, 3 is assigned to a and 4 is assigned to b . Hence, the results after annotation for r is $1:2:(3:a + 4:b)^*$.

The next step is to execute first rule (WordCon) since ab is not a literal but a word. The top left of figure 6 illustrates the first part of first rule (WordCon) for the regular expression $r = 1:2:(3:a + 4:b)^*$. The top right of figure 6 the second part of first rule (WordCon) for the appended regular expression.

The first part of the first rule (WordCon) is executed in the following manner. The first step is to execute Rule (Star) since annotation number 1 is Kleene's star. The second step is to execute Rule (Choice) as annotation number 2 is $a+b$ within the bracket. The third step is to execute Rule (Lab1) as annotation number 3 contains a which matches the first literal of the word.

In the third step, Rule (Lab1) takes a trace i which is empty at this juncture, an input literal l which is a , a matching literal expression $i:l$ which is $3:a$, and append the matching source location i to the trace $\bar{i}$, $\{(3:\epsilon, \{3\})\}$. As described above, the matching literal expression in $3:a$ is being replaced with epsilon due to a match and hence, $3:\epsilon$.

5 Further, as the set only contains one element, there is only one set $\{3\}$ in the result.

In the second step, Rule (Choice) takes a trace i which is empty at this juncture, an input literal l which is a , and a first of the two regular expressions r_1 which is $3:a$, and returns with $\{(3:\epsilon, \{3\})\}$. As described above, the first of the two regular expression $3:a$ is being replaced with epsilon due to a match and hence, $3:\epsilon$. Further, as the set only contains one element, there is only one set $\{3\}$ in the result. With regard to the second of the two regular expressions r_2 is $4:b$ which is not a match to the literal l , the second of the two regular expressions returns empty. Hence, the overall result for Rule (Choice) is $\{(3:\epsilon, \{3\})\}$.

15

In the first step, Rule (Star) takes in a trace i which is empty at this juncture, a regular expression r which is $*$ of $(a \text{ or } b)$, an input literal $l = a$, and returns with $\{(1:(3:\epsilon)(1:2:(3:a + 4:b)^*, \{3\}))\}$ as a set of possible partial derivatives and trace. As $3:a$ matches with the input literal, $3:a$ is being replaced with $3:\epsilon$.

20

The second part of the first rule (WordCon) is executed in the following manner. The first step is to execute a second Rule (WordCon) for $w = b$. Since the w contains a single literal, only the first part of the second Rule (WordCon) is presented. Since the second part of the second Rule (WordCon) is empty, $w=\epsilon$, a rule (WordEps) will be applied, which is just an identity function. Hence, the second part of the second Rule (WordCon) is omitted for brevity. The second step is to execute rule (Seq1) since annotation number 1 of the appended regular expression $r = 1:(3:\epsilon)(1:2:(3:a + 4:b)^*)$ includes a sequence expression, $r_1 r_2$ where $r_1 = 3:\epsilon$ and $r_2 = 1:2:(3:a + 4:b)^*$. The third step is to execute Rule (Star) since annotation number 1 is kleene star. The fourth step is to execute Rule (Choice) as annotation number 2 is $a+b$ within the bracket. The fifth step is to execute Rule (Lab1) as annotation number 3 contains b which matches the second literal of the word.

30

In the fifth step, Rule (Lab1) takes a trace i which is $\{3\}$ at this juncture, an input literal l which is b , a matching literal expression $i: l$ which is $4:b$, and append the matching source location i to the trace $\bar{i}$, $\{(4:\epsilon, \{3,4\})\}$. As described above, the matching literal expression in $4:b$ is being replaced with epsilon due to a match and hence, $4:\epsilon$. Further, 4 is being appended to the current set $\{3\}$ resulting in $\{3,4\}$.

In the fourth step, Rule (Choice) takes a trace i which is $\{3\}$ at this juncture, an input literal l which is b , and a first of the two regular expressions r_1 which is $3:a$, and returns empty. Rule (Choice) then takes a trace i which is $\{3\}$ at this juncture, an input literal l which is b , and a first of the two regular expressions r_2 which is $4:b$, and returns $\{(4:\epsilon, \{3,4\})\}$. As described above, the second of the two regular expression $4:b$ is being replaced with epsilon due to a match and hence, $4:\epsilon$. Further, as the set only contains $\{3\}$, 4 is being appended to $\{3\}$ resulting in $\{3,4\}$. Hence, the overall result for Rule (Choice) is $\{(4:\epsilon, \{3,4\})\}$.

In the third step, Rule (Star) takes in a trace i which is $\{3\}$ at this juncture, a regular expression r which is $*$ of a or b , an input literal $l = b$, and returns with $\{(1:(4:\epsilon)(1:2:(3:a + 4:b)^*), \{3,4\})\}$. As $4:b$ matches with the input literal, $4:b$ is being replaced with $4:\epsilon$. Annotation 1 is added to $4:\epsilon$. Hence, the modified regular expression is $1:(4:\epsilon)(1:2:(3:a + 4:b)^*)$.

In the second step, Rule (Seq1) is applicable since the input expression is in the following sequence, $1:(3:\epsilon)(1:2:(3:a + 4:b)^*)$, where $r_1 = (3:\epsilon)$ and $r_2 = (1:2:(3:a + 4:b)^*)$, which satisfy the condition $\epsilon \in r_1$. Rule (Seq1) takes a trace i which is $\{3\}$ at this juncture, an input literal $l=b$, a first of a sequence of expression $r_1 = 3:\epsilon$, and return with an empty set $\{\}$ as we cannot consume b with ϵ . This is achieved via an application of the rule (Eps) which is omitted here for simplicity. Further, Rule (Seq1) takes a trace i which is $\{3\}$ at this juncture, an input literal $l=b$, a second of the sequence of expression $r_2 = 1:2:(3:a + 4:b)^*$, and return with $\{(1:(4:\epsilon)(1:2:(3:a + 4:b)^*), \{3,4\})\}$, which is the result propagated from the third step.

The bottom part of figure 6 illustrates the end results which read as follows. For regular expression $r=1:2:(3:a + 4:b)^*$ matching the word $w=ab$, we conclude that $w=ab$

matches $1:2:(3:a + 4:b)^*$ yielding the trace $\{3,4\}$. For brevity, we have omitted the last step of applying Rule (WordEps) which is straight-forward since Rule (WordEps) is essentially a recursive of Rule (Eps).

5 Let $e \rightarrow^* v$ denotes the evaluation of a program expression e to some value v .

Proposition 1. Let match $r w \rightarrow^* True$. Then $\{\} \vdash r/w \Rightarrow \overline{(r', \bar{l'})}$ and $\overline{(r', \bar{l'})} > 0$.

Proposition 2. Let $\{\} \vdash r/w \Rightarrow \overline{(r', \bar{l'})}$ and $\overline{(r', \bar{l'})} > 0$. Then match $r w \rightarrow^* True$.

10 Proposition 1 states the completeness property while Proposition 2 states the soundness property.

Figure 7 illustrates the second phase of the extension. Accordingly, the second phase of the extension is to take an extra step from the first phase. In the second extension, besides keeping track of the traces of source locations, the arising match errors are also tracked. There are two kinds of error reports, namely the literal mismatch error $E_l^{\bar{l}}$ and the epsilon mismatch error $E_\epsilon^{\bar{l}}$. Both errors carry a trace of the partial match. In case of literal mismatch, the last i in $\bar{l}$ indicates where the error arises. Similar to the first extension in figure 5, this extension consists of two sub-systems. The first sub-system is denoted as follows.

20

$$\bar{l} \vdash r/l \Rightarrow \overline{(r', \bar{l'})}, \bar{E} \quad (4)$$

Equation (4) performs a partial derivative operation of r with respect to l . It may either produce the resulting partial derivatives $(r', \bar{l'})$ or some error $\bar{E}'$. It is similar to the first extension, except for the mismatch cases. Rule (Eps) raises an epsilon mismatch error. Rule (Lab2) raises a literal mismatch error. Rule (Lab1) consumes an input literal with the matching literal expression and append the matching source location i to the trace $\bar{l}$. Rules (Seq1) and (Seq2) deal with a sequence expression $r_1 r_2$. Rule (Seq1) is catered for the case where $\epsilon \in r_1$, otherwise rule (Seq2) is used. Rule (Choice) handles the choice expression and rule (Star) handles the Kleene's star expression.

25

30

The second sub-system is denoted as

$$\bar{l}, \bar{E} \vdash r/w \Rightarrow (\overline{r'}, \overline{l'}), \bar{E}' \quad (5)$$

Equation (5) is similar to the second extension shown in Figure 5. In addition, it propagates the set of errors over the recursion.

Example 2

Consider failing case of matching regular expression $r = (a+b)^*$ with the word $w = ac$.

The first step is to annotate the regular expression, r . For $(a+b)^*$, the first function is the Kleene's star, $*$, and the second function is the $r+r$. Thus, 1 is assigned to $(a+b)^*$, 2 is assigned to $(a+b)$, 3 is assigned to a and 4 is assigned to b . Hence, the results after annotation for r is $1:2:(3:a + 4:b)^*$.

The next step is to execute first rule (WordCon) since ac is not a literal but a word. The top left of figure 8 illustrates the first part of first rule (WordCon) for the regular expression $r = 1:2:(3:a + 4:b)^*$. The top right of figure 8 illustrates the second part of first rule (WordCon) for the appended regular expression.

The first part of the first rule (WordCon) is executed in the following manner. The first step is to execute Rule (Star) since annotation number 1 is Kleene's star. The second step is to execute Rule (Choice) as annotation number 2 is $a+b$ within the bracket. The third step is to execute Rule (Lab1) for $r=3:a$ and Rule (Lab2) for $r=4:b$.

In the third step, Rule (Lab1) takes a trace i which is empty at this juncture, an input literal l which is a , a matching literal expression $i:l$ which is $3:a$, and append the matching source location i to the trace $\bar{l}, \{3: \epsilon, \{3\}\}, \{\}$. Rule (Lab2) takes a trace i which is empty at this juncture, an input literal l which is a , a non-matching literal expression $i:l$ which is $4:b$, and returns the empty set with the matching error $E_l^{\{4\}}$.

In the second step, Rule (Choice) takes a trace i which is empty at this juncture, an input literal l which is a , and a first of the two regular expressions r_1 which is $3:a$, and returns with $\{(3:\epsilon, \{3\})\}, \{\}$. As the second of the two regular expressions r_2 is $4:b$ which is not a match to the literal l , the second of the two regular expressions returns empty with the matching error $E_l^{\{4\}}$, thanks to Rule (Lab2). Hence, the overall result is $\{(3:\epsilon, \{3\})\}, \{E_l^{\{4\}}\}$. In this instance, rule (Choice) combines the results from the applications of (Lab1) and (Lab2) in the first step.

In the first step, Rule (Star) takes in a trace i which is empty at this juncture, a regular expression r which is a^* or b^* , an input literal $l = a$, and returns with $\{1:(3:\epsilon)(1:2:(3:a + 4:b)^*), \{3\}, \{E_l^{\{4\}}\}\}$ as a set of possible partial derivatives and trace with the matching error. As $3:a$ matches with the input literal, $3:a$ is being replaced with $3:\epsilon$.

The second part of the first rule (WordCon) is executed in the following manner. The first step is to execute a second Rule (WordCon) for $w = c$. Since the w contains a single literal, only the first part of the second Rule (WordCon) is presented. Since the second part of the second Rule (WordCon) is empty, $w=\epsilon$, a rule (WordEps) will be applied, which is just an identify function. Hence, the second part of the second Rule (WordCon) is omitted for brevity. The second step is to execute rule (Seq1) since added annotation number 1 of the appended regular expression $r = (1:(3:\epsilon)(1:2:(3:a + 4:b)^*)$ includes a sequence expression, $r_1 r_2$ where $r_1 = 3:\epsilon$ and $r_2 = 1:2:(3:a + 4:b)^*$. The third step is to execute Rule (Star) since annotation number 1 is Kleene's star. The fourth step is to execute Rule (Choice) as annotation number 2 is $a+b$ within the bracket. The fifth step is to execute Rule (Lab2) for $r=3:a$ and $r=4:b$, since both are non-matching.

In the fifth step, Rule (Lab2) is executed twice, one for $r=3:a$ and one for $r=4:b$. For $r=3:a$, Rule (Lab2) takes a trace which is $\{3\}$ at this juncture, an input literal l which is c , a non-matching literal expression l' which is $3:a$, and returns the empty set since there is a mismatch and a matching error $E_l^{\{3,3\}}$. Matching error $E_l^{\{3,3\}}$ refers to input a is matching with $r=3:a$ but input c is mismatching with $r=3:a$. In particular, the first 3 refers to a match

while the second 3 refers to a mismatched. More particularly, the first 3 is taken from the trace which is $\{3\}$ while the second 3 indicates the mismatch.

For $r=4:b$, Rule (Lab2) takes a trace which is $\{3\}$ at this juncture, an input literal l which is c , a non-matching literal expression l' which is $4:b$, and returns the empty set since there is a mismatch and matching error $E_l^{\{3,4\}}$. $E_l^{\{3,4\}}$ refers to input a is matching with $r=3:a$ but input c is mis-matching with $r=4:b$. In particular, the 3 refers to a match while the 4 refers to a mismatched. More particularly, the 3 is taken from the trace which is $\{3\}$ while the 4 indicates the mismatch.

In the fourth step, Rule (Choice) takes a trace i which is $\{3\}$ at this juncture, an input literal l which is c , and a first of the two regular expressions r_1 which is $3:a$, and returns empty set since there is a mismatch and a matching error $E_l^{\{3,3\}}$. This is similar to rule (Lab2) above for $r=3:a$. Rule (Choice) then takes a trace i which is $\{3\}$ at this juncture, an input literal l which is c , and a second of the two regular expressions r_2 which is $4:b$, and returns an empty set since there is a mismatch and matching error $E_l^{\{3,4\}}$. This is similar to rule (Lab2) above for $r=4:b$.

In the third step, Rule (Star) takes in a trace i which is $\{3\}$ at this juncture, a regular expression r which is $(a+b)^*$, an input literal $l = c$, and returns with an empty set. This is because a^* means 0 or more as , b^* means 0 or more bs , and therefore $(a+b)^*$ means 0 or more literal $(a \text{ or } b)s$. Hence, a^* and b^* do not match with the input literal l and returns an empty set of partial derivatives. In this instance, the matching errors are the same as the previous step.

In the second step, Rule (Seq1) is applicable since the input expression $3:\epsilon$ is an element of $r_1 = a$, which satisfy the condition $\epsilon \in r_1$. Rule (Seq1) takes a trace i which is $\{3\}$ at this juncture, an input literal $l=c$, a first of a sequence of expression $r_1 = 3:\epsilon$, and return with an empty set $\{\}$ as we cannot consume c with ϵ and a matching error $E_\epsilon^{\{3\}}$. This is achieved via an application of the rule (Eps). Further, Rule (Seq1) takes a trace i which is $\{3\}$ at this juncture, an input literal $l=c$, a second of the sequence of expression $r_2 = 1:2:(3:a + 4:b)^*$, and return with an empty set and matching errors $E_l^{\{3,3\}}$ and $E_l^{\{3,4\}}$.

Hence, the result for rule (Seq1) is an empty set of partial derivative and a set of matching error $\{E_l^{\{3,3\}}, E_l^{\{3,4\}}, E_e^{\{3\}}\}$.

In the first step, a second Rule (WordCon) is applied for $w = c$. Since the w contains a single literal, only the first part of the second Rule (WordCon) is presented which is the propagation of the result from second step.

The bottom part of figure 8 illustrates the end results which read as follows. For regular expression $r=1:2:(3:a + 4:b)^*$ matching the word $w=ac$, we conclude that $w=ac$ does not match with $1:2:(3:a + 4:b)^*$ because of the following possible errors.

- $E_l^{\{4\}}$ reports that the input a is mismatching with $4:b$.
- $E_l^{\{3,3\}}$ reports that the input a matches successfully with $3:a$. However the following c is mismatching with $3:a$ coming from the kleene's star.
- $E_l^{\{3,4\}}$ reports that the input a matches successfully with $3:a$. However the following c is mismatching with $4:b$ coming from the kleene's star.
- $E_e^{\{3\}}$ reports that the input a matches successfully with $3:a$. However the following c is mismatching with the empty expression.

Note that it tries to report all possible failures. For practical use, we are often interested in the errors with the longest partial match. In this case, $E_l^{\{3,3\}}$ and $E_l^{\{3,4\}}$ are possible candidates.

Figure 9 illustrates a flow diagram of process 900 performed by the processor in processing system 200 in accordance with an embodiment of this invention. Particularly, process 900 is performed by the extraction pattern diagnostic subsystem 120 in accordance with an embodiment of this invention. Process 900 is a process for matching error detection of data 110 received based on the regular expression retrieved from the extraction pattern database 140. Process 900 begins with step 905 by annotating the regular expressions structurally with a series of running integers as the source locations. This is illustrated above with reference to figure 4 and examples 1 and 2.

In step 910, process 900 applies a first phase of the error detection extension as illustrated by the second deduction system denoted as equation (3) comprising Rule (WordEps) and Rule (WordCon) as shown in figure 5 to obtain any match results. Particularly, the matching algorithm is applied to the data 110 received. More particularly,
5 the first deduction system denoted by equation (1) comprising Rule (Eps), Rule (Lab1), Rule (Lab2), Rule (Seq1), Rule (Seq2), Rule (Choice), and Rule (Star) as shown in figure 5 is applied recursively as illustrated above in example 1.

In step 915, process 900 applies a second phase of the error detection extension as
10 illustrated by the second sub-system denoted as equation (5) comprising Rule (WordEps) and Rule (WordCon) as shown in figure 7 to obtain and mismatch results. Particularly, the match error algorithm is applied to the data 110 received. More particularly, the first sub-system denoted by equation (4) comprising Rule (Eps), Rule (Lab1), Rule (Lab2), Rule (Seq1), Rule (Seq2), Rule (Choice), and Rule (Star) is applied recursively as illustrated
15 above in example 2.

In step 920, process 900 generates a report containing match and mismatch results obtained from steps 910 and 915. Process 900 ends after step 920.

20 Figure 10 illustrates a flow diagram of process 1000 performed by the processor in processing system 200 in accordance with an embodiment of this invention. Process 1000 is a process for matching algorithm. Process 1000 begins with step 1005 by determining the number of characters in the data.

25 In step 1007, process 1000 determines the number of annotations in the regular expression. Particularly, each annotation contains at least one function as illustrated above in examples 1 and 2.

Process 1000 then selects a first character from the data in step 1010. The
30 following steps are repeated for each of the characters in the data.

In step 1015, process 100 determines the rule or rules based on the annotated regular expression and the input character.

In step 1020, the rules are executed accordingly. Particularly, the rules are executed from rule from the highest annotation to rule from the lowest annotation. Details on the execution of each of the rules are illustrated above in example 1.

5

In step 1025, process 1000 generates the regular expression partial derivative and appends the trace based on that derived from the execution of the determined rules in step 1020.

10

In step 1030, process 1000 determines whether the selected character is the last character in the data. If the selected character is not the last character in the data, process 1000 proceeds to step 1035 to select the next character and repeats from step 1015. Step 1015 is required to determine the rule or rules based on the annotated regular expression and the next input character. Process 1000 proceeds to step 1040 if the selected character is the last character in the data. In step 1040, process 1000 generates the match result using the trace.

15

Figure 11 illustrates a flow diagram of process 1100 performed by the processor in processing system 200 in accordance with an embodiment of this invention. Process 1100 is a process for match error algorithm. Process 1100 begins with step 1105 by determining the number of characters in the data.

20

In step 1107, process 1100 determines the number of annotations in the regular expression. Particularly, each annotation contains at least one function as illustrated above in examples 1 and 2.

25

Process 1100 then selects a first character from the data in step 1110. The following steps are repeated for each of the characters in the data.

30

In step 1115, process 1100 determines the rule or rules based on the annotated regular expression and the input character.

In step 1120, the rules are executed accordingly. Particularly, the rules are executed from rule from the highest annotation to rule from the lowest annotation. Details on the execution of the rules are illustrated above in example 2.

5 In step 1125, process 1100 generates regular expression partial derivative and appends the matching error derived from the execution of the determined rules.

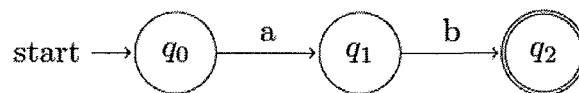
In step 1130, process 1100 determines whether the selected character is the last character in the data. If the selected character is not the last character in the data, process 1100 proceeds to step 1135 to select the next character and repeats from step 1115. Step 10 1115 is required to determine the rule or rules based on the annotated regular expression and the input character. Process 1100 proceeds to step 1140 if the selected character is the last character in the data. In step 1140, process 1100 generates the mismatch result. Particularly, the matching errors with the longest partial match to selected to generate the 15 mismatch result.

Refinement process

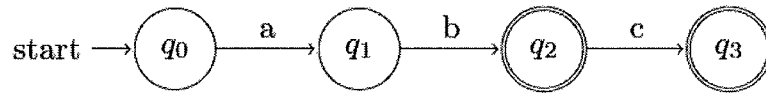
The refinement process is a process for refining the regular expression. This 20 refinement process is based on four operations, namely, 1) State-extension, 2) Transition-extension, 3) State-finalization, and 4) No-change.

State-extension

Assuming the regular expression is “ ab ”, matching a string of “ abc ” would result 25 in a match failure. In order to refine the regular expression, we can extend “ ab ” by adding another state, q_3 , and finalising the last two states, q_2 and q_3 . Hence, the refined regular expression using state-extension for this case would be $ab(c + \epsilon)$.

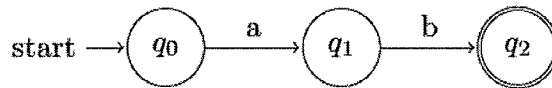


into

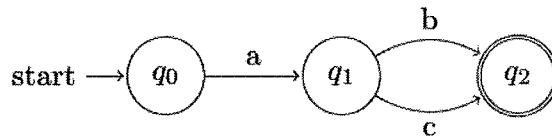


Transition-extension

Supposing the string is now “ac”, the regular expression “ab” would return with a
 5 match failure. In order to refine the regular expression, we can extend “ab” by adding a
 new transition. Hence, the refined regular expression using transition-extension for this
 case would be $a(b + c)$.



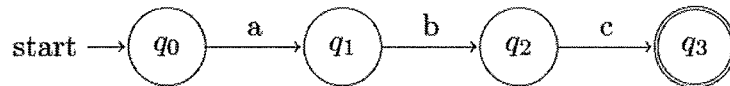
into



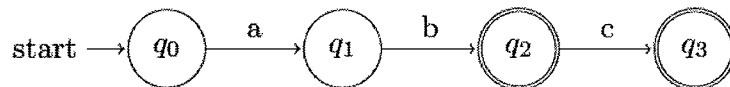
10

State-finalization

Supposing the string is now “ab” and the regular expression is “abc”, a match
 failure would arise. In order to refine the regular expression, we can finalised the last
 15 second state, q_2 . Hence, the refined regular expression using state-finalisation for this case
 would be $ab(c + \epsilon)$.



will be adjusted into



No-change

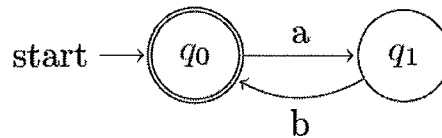
20 The no-change operation is needed in case that the new input string is already
 accepted by the original regular expression.

There are more challenges that we notice. Given an input string w and an original regular expression r , there exist multiple ways of modifying r into r' so that $w \in r'$. For convenience, we use $w \in r'$ as a short-hand to denote $w \in L(r')$.

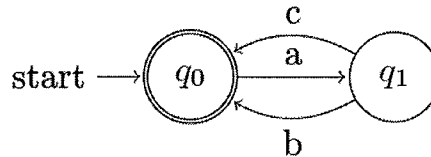
5

Let's consider the following example. We would like to modify regular expression $(ab)^*$ to accept input ac . We have at least two possible modifications namely, $(a(b + c))^*$ and $(ab)^*ac$. To make the algorithm deterministic, we need to favour one to the other. Let's compare the two possible modifications in terms of automata changes. The automaton constructed from $(ab)^*$ is

10

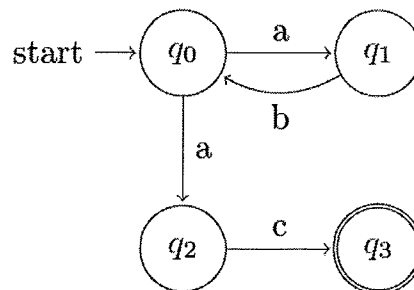


The automaton constructed from $(a(b + c))^*$ is



15

We compare it against the automaton constructed from $(ab)^*ac$



20

The modification operations introduced in $(a(b + c))^*$ is simpler than $(ab)^*ac$, because the former introduces only one new transition while the latter introduces two new

states and two new transitions. In other words, we favour the modifications with few changes.

However the objective of the invention is to perform modification directly over the regular expressions without constructing the minimized automata. Transferring the ideas of expressing the modification in terms of the four types of modification operations, the plan is to apply modifications to the source locations where the mismatch arising.

Consider the regular expression a^*a^* . We annotate it with source location annotations as follows $1:((2:(4:a)^*)(3:(5:a)^*))$ where 1, 2, 3, 4 and 5 are five different source locations. Suppose we send an input string ac to this regular expression. There are at least two possible modifications to cater for the change, namely

$$1:((2:(6:(4:a + 7:c))^*)(3:(5:a)^*))$$

and

$$1:((2:(4:a)^*)(3:(6:(5:a + 7:c))^*))$$

According to the comparison strategy described earlier, these two alternatives are equally small. To address this ambiguity issue, we take into account the user requirement. Before introducing the user requirement, let's consider the regular expression sub-matching problem. In real world applications, regular expression sub-matching is a common use case for using regular expression. Regular expression sub-matching is often realized as the “grouping” in most of Perl Compatible Regular Expression (PCRE) implementation.

Recalling the regular expression our last example,

$$1:((2:(4:a)^*)(3:(5:a)^*))$$

There are two possible ways of matching the input string a , and they are as follows:

1. a is “consumed” by the sub expression $(2:(4:a)^*)$;
2. a is “consumed” by the sub expression $(3:(5:a)^*)$.

In many existing implementation, we solve the ambiguity by simply applying a match policy, i.e. we favour the first matching if the implementation favours a greedy matching policy. On the other hand, some implementations allow the programming to specify the greediness of a “grouping”. By augmenting the “grouping” with extra syntactic annotation, the programmers are able to “force” the match to favour the second match result.

The regular expression sub matching also allows us to extract a sub-portion of the string from the input string.

We now consider

$$(1:(.))(2:(<h><s>))(3:(a + b + c)^*)(4:(</s></h>))(5:(.))$$

a simplified version of a realworld extraction pattern which extracts the text between the XML tags `<h><s>` and `</s></h>`. Note that `.` denotes a wild card expression which matches any single character. For simplicity, we only consider the source locations at the top level and omit those at the lower level (e.g. leaf nodes). This technique is often known as boundary in text extraction using regular expression.

The main idea is to allow programmer to express their requirement for the matching and the fix of the broken matches. For instance, the user requirement for the above regular expression could be that the sub-matching under source location 3 must not be empty and it must be only a string consists of *a*, *b* and *cs*, because it is what the programmers would like to extract.

Consider the input `<h><s><d>abc</d></s></h>`. The best fix to above pattern must be

$$(1:(.))(2:(<h><s><d>))(3:(a + b + c)^*)(4:(</d></s></h>))(5:(.))$$

instead of

$$(1:(.))(2:(<h><s>))(3:<d>(a + b + c)^*</d>)(4:(</s></h>))(5:(.))$$

In summary, the idea is to express all possible modification in terms of the four operations. In addition, we incorporate the user requirement when searching for the “best” modification by ranking the modification based on their simplicity levels.

5

It is important to note that there are many methods to refine the regular expression. Some methods may introduce a more complex regular expression that requires more transitions and states than the others. Hence, one objective of this refinement process is to apply refinements with the least changes. The above four operations will now be described as follows.

10

Figure 12 illustrates a formulation of a regular expression replacement relation in terms of $R, r \vdash w : r'$ where R denotes a set of user requirements of form (i, r) in which the source location i must capture the set of possible values $v \in r$. R_r returns a subset of R whose domain is the set of source locations appearing in r . The $\Leftrightarrow$ sign reads as if and only if. The judgment $R, r \vdash w : r'$ reads as follows, under the requirement environment R , r can be replaced by r' to accept input word w .

15

There are seven cases. The first case 1210 deals with the scenario in which the regular expression is being replaced with an epsilon, ϵ . The replacement relation holds if and only if the input word w can be found in the replacement expression r and the to-be-replaced expression $i : \epsilon$ is subsumed by r . In addition, if the source location i is mentioned in the requirement environment R , w must be accepted by r' which is the regular expression specified by the user requirement. It is observed that in practice, r' can be defined by the user, and r will often be precise as it will be “inferred” by the refinement algorithm which we will introduce shortly.

20

25

The second case 1220 deals with the scenario where regular expression to be replaced is a l . Similar to the first case, the relation holds if and only if the input word w can be found in r and $i : l$ is subsumed by r .

30

The third case 1230 deals with the scenario where the regular expression to be replaced is a concatenation $i: r_1 r_2$. The replacement relation holds if and only if that w can be split into $w_1 w_2$ and the relation holds for w_1 , r_1 , and r'_1 , as well as for w_2 , r_2 , and r'_2 .

5 The fourth case 1240 and the fifth case 1250 deal with choice regular expression. The relation holds if and only if the relation holds either among w , r_1 , and r'_1 , or among w , r_2 , and r'_2 .

10 The sixth case 1260 deals with Kleene's star regular expression r^* . The relation holds if and only there exists a split of the non-empty word w into $w_1 \dots w_n$ such that the relation holds among r and r' for each w_i .

The seventh case 1270 is a special scenario of the Kleene's star regular expression, in which the input is an epsilon, ϵ . It holds immediately because $\epsilon \in r^*$ for all r .

15

In brief, the replacement relation $R, r \vdash w :: r'$ is a correctness assertion to ensure that the refinement algorithm proposed is correct, i.e. the refinement algorithm will generate a r' that is correct with respect to the user refinement requirement environment R and the input w .

20

Proposition 3. Let $w \in r$. Then $R, r \vdash w :: r$

Proposition 4. Let $w \in r$ and $R, r \vdash w :: r'$. Then $w \in r'$

25 Proposition 3 states that the replacement relation is reflexive. Proposition 4 states that the replacement relation implies the subsumption relation. In other words, the refined regular expression will match with all inputs that are matched by the original regular expression.

Definition 1. We use $r \lesssim r'$ to denote a relation among r and r' as follows

30
$$i: \epsilon \lesssim i: r \quad i: l \lesssim i: r \quad \frac{r_1 \lesssim r'_1 \quad r_2 \lesssim r'_2}{i: r_1 r_2 \lesssim i: r'_1 r'_2} \quad \frac{r_1 \lesssim r'_1 \quad r_2 \lesssim r'_2}{i: r_1 + r_2 \lesssim i: r'_1 + r'_2} \quad \frac{r \lesssim r'}{i: r^* \lesssim i: r'^*}$$

In short, $r \lesssim r'$ enforces r and r' shares the same syntactic structure modulo as the leaf nodes (such as l). Hence, r has a simpler structure than r' .

The next property, $w \in r \Rightarrow \Delta$, states the replacement relation preserves the sub-matching results. The regular expression sub-matching relation is defined in Figure 13. Δ denotes the value binding environment generated by the matching. It is an ordered bag or a multi-set. $\Delta_1 \uplus \Delta_2$ concatenates the elements in both multi-sets. The judgement $w \in r \Rightarrow \Delta$ defines the sub-matching algorithm, which always favours the left most longest sub match. The detailed definitions can be found in the following publication, “Martin Sulzmann and Kenny Zhuo Ming Lu. Regular expression sub-matching using partial derivatives. In Proc. Of PPDP’12, pages 79-90.ACM, 2012”.

The relation $\Delta \vdash R$ states that the value binding environment Δ is conforming to the refinement requirement R . It is defined as

Definition 2.

$$\Delta \vdash R \Leftrightarrow \forall (i, r) \in R \text{ we have } (i, w_1) \in \Delta, \dots, (i, w_n) \in \Delta \wedge w_1 \dots w_n \in r$$

Proposition 5. Let $R, r \vdash w :: r'$ and $w \notin r$. Then $r \lesssim r'$ and $w \in r' \Rightarrow \Delta$ and $\Delta \vdash R$.

$R_1 \oplus R_2$ returns a refinement requirement environment same as R_1 except for all source location i mentioned in R_2 will be replaced by $R_2(i)$.

In figures 14 and 15, we present the regular expression matching refinement algorithm. The judgement $R \vdash_A^L r/l \Rightarrow \overline{(R, r, \theta, A)}$ extends from the partial derivative operation introduced above.

L denotes the levels of precedence, range over strong (S) and weak (W). The default level of precedence is W . It switches to S when the current source location is in the domain of the user requirement environment R . A denotes an alternative indication to the source location.

θ denotes the refinement environment for each source location, i , and a corresponding refinement operation, op . As mentioned above, there are four operations

namely, 1) state-extension, $i: l + \overset{o}{L}$, 2) transition-extension, $i: l + \vec{L}$, 3) state-finalisation, $+^\epsilon$, and 4) no change, id_L .

The rule (EpsW) proposes a fix to epsilon-literal mismatch by adding a state-extension, in case where the current source location i is not in the domain of the refinement requirement, $dom(R)$. Refinement requirement, R is $\{(i_1, r_1), (i_2, r_2)\}$ and domain of the refinement requirement, $dom(R)$ is $\{i_1, i_2\}$. As the dual, the rule (EpsS) in addition ensures that $(i:r) \in R$ and r can consume the input literal l . Rules (EpsW) and (EpsS) deal with cases where the input regular expression is an epsilon and the input literal is a l . Based on the earlier discussion, it is obvious to see that there will be mismatches. The objective is to look for a fix for the mismatches. Rule (EpsW) is specifically applied when the source location annotated with the epsilon is not in the domain of the refinement requirement R . In this case, the rule returns a singleton set containing a tuple of fours, namely, the (unchanged) refinement requirement R , the partial derivative $j:\epsilon$, the refinement operation $\{(i, j: l + \overset{o}{L'})\}$, the (unchanged) alias environment A . The refinement requirement R is unchanged, because the source location i is not in the domain of R . The resulting partial derivative is $j:\epsilon$, because we have constructed a possible fix to turn the mismatch into a match. Note that j is a new source location. The refinement operation that we construct is to add a state transition at source location i , with the literal l . The level L' is determined by taking the maximum of the input level L and a weak level (W). In case of the source location i is in the domain of the refinement requirement R , there is a regular expression r associated with i in R as a requirement constraint. We need to apply rule (EpsS). Besides computing partial derivative $j:\epsilon$, the refinement operation $\{(i, j: l + \overset{o}{L'})\}$, the alias environment A , we need to compute a set of new refinement requirement environments. This is because the requirement R is given by the user as the refinement guideline, the requirement have to be updated when source location i is refined. For each partial derivative r' of r with respect to l , we create a new refinement environment by replacing (i, r) by (i, r') in R .

Rules (LitS1), (LitS2), (LitW1) and (LitW2) deal with the case in which the regular expression to be refined is a literal. Rules (LitS2) and (LitW2) generate a fix to the literal mismatch.

Rules (LitS1) and (LitW1) take care of the cases in which the input literal matches with the regular expression l . Hence a “no-change” refinement operation is added. In addition, Rules (LitS2) and (LitS1) update the user requirement environment R with $(i:r')$ where r' is the partial derivative of r and $(i:r) \in R$. Rule (LitW1) is applied when the input regular expression l matches with the input literal l , and the source location i is not in the domain of R . We compute the partial derivative, and a “no-change” refinement operation. On the other hand, when the source location i is in the domain of R , we apply rule (LitS1). The result is the same as rule (LitW1) except that we need to compute a new set of requirements by computing the partial derivatives of r associated with i in R . This is for the same the reason given in the explanation of rule (EpsS). Rule (LitW2) is applied when the input regular expression l does not match with the input literal l' . A refinement operation $i:l' + \vec{l'}$ is generated and a partial derivative $i:\epsilon$ is generated. Note that there is no change to the requirement R . On the other hand, Rule (LitS2) is applied when the input regular expression l does not match with the input literal l' , in addition the source location i is found in the domain of R . The output is a variant of the one of Rule (LitW2) except that the requirement R is modified with partial derivative of r with respect to l , where (i,r) in R .

Rules (SeqS1), (SeqS2), (SeqW1) and (SeqW2) take care of the cases where the regular expression to be refined is a concatenation. Rules (SeqS1) and (SeqW1) take care of the situations in which r_1 possesses the epsilon, ϵ . In such cases, we need to consider the refinement of r_1 as well as the refinement of r_2 . Unlike the normal partial derivative operation, in case of $\epsilon \notin r_1$, i.e. (SeqS2) and (SeqW2), we still try to refine r_2 with literal l by making r_1 final via the $mkfin(r_1)$ refinement operations. This allows us to fix the mismatching of $l:2:a_b$ with b . In the case of (SeqS1) and (SeqS2), the user requirement environment need to be updated with $(i:r')$ where r' is the partial derivative of r and $(i:r) \in R$.

Rule (SeqW1) is applied when the input literal is l , the input regular expression is a sequence $r_1 r_2$, where r_1 is accepting epsilon. In addition, the source location i is not in the domain of the requirement R . The rule proceeds by making two sub-calls to compute the refinement results of refining r_1 with l and refining r_2 with l respectively, yielding intermediate results V_1 and V_2 . The result of this rule is computed by taking the union of

V_2 and the modified V_I . We need to modify V_I because we need to “restore” the r_2 to every partial derivative r_1' coming from V_I and “restore” the refinement requirement related to source locations appearing in r_2 to R_I' , that is all the resulted requirements coming from V_I .

5

The rule (SeqS1) is almost the same as rule (SeqW1) except that it is applicable when the source location i annotated to the input regular expression can be found in the domain of the requirement R . In this case, we need to compute the set of updated requirements besides the partial derivatives and the refinement operations using the same “trick” that we used in rule (EpsS). In addition, we force the two sub-calls to be made under a strong refinement level (S).

10

The rule (SeqW2) is applied when the input literal is l , the input regular expression is a sequence r_1r_2 , where r_1 does not accept the epsilon. In this case we need to get the refinement results by making a sub-call to refine r_1 with l . In addition, we make a sub-call to try to refine r_2 with l , under the “assumption” that r_1 can be turned into an expression that accepts epsilon. This is achieved via the operation, $mkfin(r_1)$. As highlighted in figure 17, $mkfin(r_1)$ structurally walks through r_1 looking for possible refinement operations to make r_1 accepting epsilon. The output of the rule (SeqW2) is obtained by combining the sub-call results V_2 and the modified of sub-call result V_I . The idea is the same as what we have seen in rule (SeqW1).

15

20

The rule (SeqS2) is almost the same as rule (SeqW2) except that it is applicable when the source location i annotated to the input regular expression can be found in the domain of the requirement R . In this case, we need to compute the set of updated requirements besides the partial derivatives and the refinement operations using the same “trick” that we use in rule (EpsS).

25

The rules (ChoiceS) and (ChoiceW) take care of the situations where the regular expression to be refined is a choice. We consider the results by refining r_1 as well as those by refining r_2 . In the case of (ChoiceS), we update the user requirement environment with $(i:r')$ where r' is the partial derivative of r and $(i:r) \in R$. Rule (ChoiceW) handles the case where the input literal is l , the input regular expression is (r_1+r_2) and the source

30

location i is not in the domain of the requirement R . In this case, we make two sub-calls of refining r_1 with l and r_2 with l respectively. Given the intermediate results as V_1 and V_2 , we compute the output via a set union of V_1 and V_2 .

5 Rule (ChoiceS) is applied when the input literal is l , the input regular expression is (r_1+r_2) and the source location i is in the domain of the requirement R . Its output is almost the same as the one of (ChoiceW) except that we need to update the requirement R with $(i:r')$ where r' is the partial derivative of r (associated with i in R) with respect to l . In addition, the sub-calls are made under a strong refinement level (S).

10 The rules (StarS) and (StarW) require extra attention. In the premises, we refine r with l . It is noted that we have to re-label the source locations under r^* , creating a new source alias environment A' . For instance,

15 Example 3.

$$\text{relabel}(1:2:a^*) = (3:4:a^*, \{1 \mapsto 3, 2 \mapsto 4\})$$

20 Then we concatenate the result r' with the relabeled of r^* instead of just r^* . It is a technical necessity arising from the proof we introduce later. The main reason behind is that without the re-labelling step, the partial derivative $i:r'r^*$ will have overlapping source locations among r' and r^* . This will complicate the correctness proof. Hence we demand a re-labeling step and keeping the source location alias explicit.

25 Like other S-rules, in case of (StarS), we update the user requirement environment with $(i:r')$ where r' is the partial derivative of r and $(i:r) \in R$. Rule (StarW) handles the case when the input literal is l , the input regular expression is r^* and the source location i is not in the domain of the requirement R . To avoid source location conflict, as highlighted earlier, we relabel the sub-expression r to r' and refine r' with l . Given the intermediate output is V , we compute the final output by appending $(i:r^*)$ back to all the partial derivatives in V . Rule (StarS) handles the case when the input literal is l , the input regular expression is (r^*) and the source location i is in the domain of the requirement R . Its output is similar to (StarW), except that we need to update the requirement R with (i, r'_o)

where r'_o is the partial derivative of r_o . In addition, the sub-call is made under a strong refinement level (S).

The rules (WordCon) and (WordEps) extend the refinement operation to support a sequence of literal input. We apply the refinement operation for a single literal recursively over the sequence. The user requirement environments and the refinement operations are ‘propagated’ over the recursive call.

In figure 16, we define the application of refinement environment to a regular expression. The application process is defined in ML-style pseudo-code. The application is needed for the construction of the “refined” regular expression given the refinement environment θ and the source location alias environment A from the original regular expression. For instance, $\theta(A)(r) = r'$ where r' is the refined regular expression constructed from r based on the refinement environment θ and the source location alias environment A . These environments are the results generated by the refinement algorithms found in figures 14 and 15. There are five cases 1610-1650 in the application of refinement environment. Note that a refinement operation is only making changes to the leaf node source locations. In case of concatenation regular expression, choice regular expression and kleene’s star, (i.e. the third 1630, fourth 1640 and fifth 1650 cases), we apply the refinement environment structurally to the sub expressions. In case of epsilon $i: \epsilon$ 1610, we need to construct the closure of the source location i in A , denoted by $A^+(i)$. By looking up all the refinement operations associated with i and its aliases $\{j_1, \dots, j_n\}$ in Θ , we categorize the operations into transition-extension operations and state-extension operations.

For transitions-extension operations, we need to build a choice over all the transition literals. For state-extension operations, we need to apply $\theta(A)(\cdot)$ recursively to all the appended literals to cater for the subsequent refinement associated with the new literals. As the final result, using the choice operator, we aggregate the sub expressions obtained from the transition-extension operations and the state-extension operations with the original regular expression, $(i: \epsilon)$. In case of literal $(i: l)$ 1620, it operates similar to the epsilon case, except that we need to:

- (1) add an epsilon choice in the presence of $+$ operations;

(2) introduce an epsilon choice together with sub-expression coming from the state-extension operations. This is to make sure refined expression subsumes the original one.

5 An important property of the refinement environment application is that it does not break semantic of the partial derivatives.

Proposition 6. Let $w \in \theta(A)(r')$ and $r' \in r/P l$. Then $lw \in \theta(A)(r)$.

10 Given the user requirement R and the refinement level L , taking away the literal l from the regular expression r from the left most position results in a set of possible tuples containing an updated user requirement, partial derivative and a refinement environment,

Let $srcs(r)$ returns all source locations in r , which is defined as

$$\begin{aligned}
 15 \quad srcs(i: \epsilon) &= \{i\} \\
 srcs(i: l) &= \{i\} \\
 srcs(i: r_1 r_2) &= \{i\} \cup srcs(r_1) \cup srcs(r_2) \\
 srcs(i: r_1 + r_2) &= \{i\} \cup srcs(r_1) \cup srcs(r_2) \\
 srcs(i: r^*) &= \{i\} \cup srcs(r)
 \end{aligned}$$

20

And let $\theta(A) \cdot \theta'(A')(r)$ be the function composition over the refinement environment application, i.e.

$$\theta(A) \cdot \theta'(A')(r) = \theta(A)(\theta'(A')(r))$$

we have

Proposition 7. Let $R \vdash_A^L r/l \Rightarrow \overline{\{R, r, \theta, A\}}$. Let w be a word, $(R_i, r_i, \theta_i, A_i) \in \overline{\{R, r, \theta, A\}}$, a refinement environment θ' and an alias environment A' such that $dom(\theta') \cup dom(A') - srcs(r_i) = \emptyset$ and $R_i, r_i \vdash w :: \theta'(A')(r_i)$. Then $R, r \vdash lw :: \theta'(A') \cdot \theta_i(A)(r)$.

25

Proof. Here is the proof sketch. We prove by induction on over structure of r . Case (EpsS) Consider

30

$$\frac{(i, r) \in R \quad j \text{ is a fresh src loc}}{R \vdash_A^L i: \epsilon/l \Rightarrow \{R \oplus \{(i, r')\}, j: \epsilon, \{(i, j: l +_s^o)\}, A | r' \in r/P l\}}$$

Without loss of generality, let $r_k \in r^p l$ and $(R_k, r'_k, \theta_k, A_k) \in \{R \oplus \{(i, r')\}, j: \epsilon, \{(i, j: l +_s^o)\}, A | r' \in r^p l\}$ i.e, $R_k = R \oplus \{(i, r')\}$, $r'_k = j: \epsilon$ and $\theta_k = \{(i, j: l +_s^o)\}$. Note that $A_k = A$ where $i \notin \text{dom}(A)$, this is because the source location alias is only introduced in (StarS) and (StarW) rules and the updated alias environment to re-label the locations under the star, the updated alias environment will be picked up in the (WordCon) for subsequent steps. Note that r_k is the updated user requirement expression and $r'_k = j: \epsilon$ is the residual of the r after refining it with l . Let θ' be a refinement operation environment, and A' be a source location alias environment such that

$$i \notin \text{dom}(\theta') \cup \text{dom}(A') \quad (1)$$

$$R_k, j: \epsilon \vdash w : : \theta'(A')(j: \epsilon) \quad (2)$$

By definition of the replacement relation we have

$$w \in \theta'(A')(j: \epsilon) \quad (3)$$

Note that

$$\theta_k = \{(i, j: l +_s^o)\} \quad (4)$$

Note that

$$j: \epsilon \in \theta_k, (A_k)(i: \epsilon) / l \quad (5)$$

Thanks to (1) and (5) we can apply Proposition 7

$$lw \in \theta'(A')(\theta_k, (A_k)(i: \epsilon)) \quad (6)$$

By definition of the replacement relation we have

$$R, i: \epsilon \vdash lw : : \theta'(A')(\theta_k(A_k)(i: \epsilon)) \quad (7)$$

That is

$$R, i: \epsilon \vdash lw : : \theta'(A') \cdot (\theta_k(A_k)(i: \epsilon)) \quad (8)$$

The case is proven.

Case (SeqS1) Consider

$$(i, r) \in R \quad \epsilon \in r_1 \quad R_{\overline{r_1}} \vdash_A^S r_1 / l \Rightarrow V_1 \quad R_{\overline{r_2}} \vdash_A^S r_2 / l \Rightarrow V_2 \\ V'_1 = \{(R'_1 \cup R_{\overline{r_2}} \cup \{(i, r')\}, i: (r'_1 r'_2), \theta, A) | (R'_1, r'_1, \theta, A) \in V_1, r' \in r^p l\}$$

$$\frac{V'_2 = \{(R'_2 \cup \{(i, r')\}, r'_2, \theta, A) \mid (R'_2, r'_2, \theta, A) \in V_2, r' \in r/p l\}}{R \vdash_A^L i: (r_1, r_2)/l \Rightarrow V'_1 \cup V'_2}$$

We consider two different sub-cases.

1. Consider $(R_k, r'_k, \theta_k, A_k) \in V_1$ and $r' \in r/p l$. Let θ' be a regular expression refinement environment, A' be an source location alias environment and w be a word such that

$$R_k \cup R'_2, i: r'_k r_2 \vdash w :: \theta'(A')(i: r'_k r_2) \quad (9)$$

where $R'_2 = R_{\overline{r_2}} \cup \{(i, r')\}$. By definition of the replacement relation, we have

$$w = w_1 w_2 \quad (10)$$

and

$$R_k, r'_k \vdash w_1 :: \theta'(A') \cdot \theta_k(A_k)(r'_k) \quad (11)$$

and

$$R_{\overline{r_2}}, r_2 \vdash w_2 :: \theta'(A') \cdot \theta_k(A_k)(r_2) \quad (12)$$

By applying induction hypothesis to premise $R_{\overline{r_1}} \vdash_A^S r_1/l \Rightarrow V_1$ and (11) we have

$$R_{\overline{r_1}}, r_1 \vdash l w_1 :: \theta'(A') \cdot \theta_k(A_k)(r_1) \quad (13)$$

Given (12) and (13), by definition of the replacement relation, we have

$$R_{\overline{r_1}} \cup R_{\overline{r_2}} \cup \{(i, r)\}, i: r_1 r_2 \vdash l w_1 w_2 :: i: (\theta'(A') \cdot \theta_k(A_k)(r_1))(\theta'(A') \cdot \theta_k(A_k)(r_2)) \quad (14)$$

which is

$$R, i: r_1 r_2 \vdash \theta'(A') \cdot \theta_k(A_k)(i: r_1 r_2) \quad (15)$$

This case is proven.

2. Consider $(R_k, r'_k, \theta_k, A_k) \in V_2$ and $r' \in r/p l$. Let θ' be a regular expression refinement environment, A' be an source location alias environment and w be a word such that

$$R_k \cup \{(i, r')\}, r'_k \vdash w :: \theta'(A')(r'_k) \quad (16)$$

Since r'_k is coming from V_2 that is $r'_k \in \theta_k(A')r_2/l$. By applying induction hypothesis to premise $R_{\overline{r_2}} \vdash \hat{S}r_2/l \Rightarrow V_2$ we have

$$5 \quad R_{\overline{r_2}}, r_2 \vdash lw :: \theta'(A') \cdot \theta_k(A_k)(r_2) \quad (17)$$

Note that $\text{dom}(\theta') \cup \text{dom}(A') - \text{srcs}(r_k) = \emptyset$ implies $(\text{srcs}(r_l) \cup \text{dom}(A')) \cap \text{src}(r_k) = \emptyset$. It follows that

$$10 \quad \theta'(A') \cdot \theta_k(A_k)(r_1) = \theta_k(A_k)(r_1) \quad (18)$$

Given $\epsilon \in r_1$, applying propositions 3 and 4, we have

$$R_{\overline{r_1}}, r_1 \vdash \epsilon :: \theta'(A') \cdot \theta_k(A_k)(r_1) \quad (19)$$

15

Given (17) and (19), by definition of the replacement relation, we have

$$R_{\overline{r_1}} \cup R_{\overline{r_2}} \cup \{(i, r)\}, i: r_1 r_2 \vdash \epsilon lw :: i: (\theta'(A') \cdot \theta_k(A_k)(r_1))(\theta'(A') \cdot \theta_k(A_k)(r_2)) \quad (20)$$

20

That is

$$R, i: r_1 r_2 \vdash lw :: \theta'(A') \cdot \theta_k(A_k)(i: r_1 r_2) \quad (21)$$

Hence this case is also proven.

25

Proposition 8. Let $R \vdash_A^L r/l \Rightarrow \overline{\{R, r, \theta, A\}}$. Then for all word w , there exists index i and regular expression r' . we have $R, r \vdash lw :: \theta_i(A_i)(lr')$ implies $R_i, r_i \vdash w :: \theta_i(A_i)(r')$.

Proof. By induction over structure of r .

30

In figure 15, we extend the judgement $R \vdash_A^L r/w \Rightarrow \overline{\{R, r, \theta, A\}}$ to support words. The extension is straight-forwardly defined as a recursive application of $R \vdash_A^L r/l \Rightarrow$

$\overline{\{R, r, \theta, A\}}$ by viewing w as a list of literals. Please note that for brevity, $v = \overline{\{R, r, \theta, A\}}$.

It follows immediately that we can define the refinement algorithm in some pseudo-ML code as in figure 17. The *refineAlgo* algorithm takes a refinement requirement R , the target regular expression r and a input word w . First, it applies the refinement operation defined in figures 14 and 15 to the input word w . Note that the immediate result set of tuples are all possible states. The resulting refinement environments from the first step are divided into two categories, namely $\bar{\theta}$ and $\bar{\theta}'$ where $\bar{\theta}$ contains those refinement environments that makes resulting partial derivative r' final, i.e. accepting epsilon. The other set contains those that makes the resulting partial derivative r' non- final, hence we need to extend these refinement environment with the *mkFin* refinement operations. This is achieved by applying $\text{mkfin}(\cdot)$ to r' . We call this set of refinement environments $\bar{\theta}'$. The second last step is to search for the refinement environment from $\bar{\theta} \cup \bar{\theta}'$ with the fewest changes to the original regular expression. This is implemented by searching for the minimum value according to the $<$ order. The last step applies the refinement environment with the fewest changes to the original regular expression.

The total order $<$ among refinement environments is defined in figure 18. $|\theta|$ returns the number of refinement operations in θ . $|\theta|id_s$ returns the number of no-change operations associated with S precedence level in θ . $|\theta|_{-}+\vec{s}$ returns the number of transition-extension operations associated with S precedence level in θ . $|\theta|_{-}+s^0$ returns the number of state-extension operations associated with S precedence level in θ . $|\theta_1|+\epsilon$ returns the the number of state-finalization operations in θ_1 . Note that in $<$ relation, we favor θ with fewer operations. In case they share the same number of operations, we favor one with fewer weak precedencies then with fewer state-extension operations, etc. $\theta_1 < \theta_2$ means that either θ_1 has fewer refinement changes than θ_2 , or they have the same number of changes, but θ_1 has a higher precedence than θ_2 .

Proposition 9. Let $R \vdash_A^L r/w \Rightarrow V$. Then for all $(R_i, r_i, \theta_i, i) \in V$, we have

1. $\epsilon \in r_i$ implies $R, r \vdash :: \theta_i(A_i)(r_i)$
2. $\epsilon \in r', \text{mkfin}(r') = \theta_e$ implies $R, r \vdash w :: \theta_e(A_i) \cdot \theta_i(A_i)(r_i)$

Proof. Prove by induction over w .

Case (WordEps)

$$R \vdash_A^L r/\epsilon \Rightarrow \{(R, r, \{\}, A)\}$$

5

1. In case $\epsilon \in r$. By definition of replacement relation, we have

$$R, r \vdash w :: \{\}(A)(r) \quad (22)$$

2. In case $\epsilon \notin r$. we construct $\Theta_e = mkfin(r)$. By definition of $mkfin$ in Figure 16, we have $\epsilon \in \Theta_e(A) \cdot \{\}(A)(r)$. By definition of replacement relation, we have

$$10 \quad R, r \vdash w :: \Theta_e(A) \cdot \{\}(A)(r) \quad (23)$$

Case

$$\begin{array}{c} R \vdash_A^L r/l \Rightarrow \overline{\{(R, r, \theta, A)\}} \\ \text{for } i \in \{1, \dots, n\} \quad R_i \vdash_{A \cup A_i}^L r_i/w \Rightarrow V_i \\ V = \{(R'_i, r'_i, \theta_i \cup \theta'_i, A') \mid (R'_i, r'_i, \theta'_i, A') \in \cup_i V_i\} \\ \hline R \vdash_A^L r/lw \Rightarrow V \end{array}$$

By induction hypothesis, we have for $i \in \{1, \dots, n\}$ for $(R_k, r_k, \theta_k, A_k) \in V_1$ we have

$$15 \quad \epsilon \in r_i \text{ implies } R_i, r_i \vdash w :: \theta_k(A_k)(r_i) \quad (24)$$

$$\epsilon \notin r_k \wedge \theta_e = mkfin(r_k) \text{ implies } R_i, r_i \vdash w :: \theta_e(A_k) \cdot \theta_k(A_k)(r_i) \quad (25)$$

By applying Proposition 7 we can conclude that for $(R_k, r_k, \theta_k \cdot \theta_i A_k) \in V$ we have

$$20 \quad \epsilon \in r_k \text{ implies } R, r \vdash lw :: \theta_k(A_k) \cdot \theta_i(A_k)(r) \quad (26)$$

$$(27) \quad \epsilon \notin r_k \wedge \theta_e = mkfin(r_k) \text{ implies } R, r \vdash lw :: (\theta_e(A_k) \cdot (\theta_k(A_k) \cdot \theta_i(A_k)))(r)$$

The soundness property of the *refineAlgo* follows from Proposition 9 immediately,

25

Proposition 10. Let $refineAlgo(R)(r)(w) = r'$. Then $R, r \vdash w :: r'$.

Example 4

Consider fixing the mismatching of regular expression $r = \langle(p+s)\rangle$ with data = $\langle i \rangle$.

Given the user requirement

$$R = \{4: (a+\dots+z)^*\}$$

The data comprises 3 characters, namely '<,' 'i' and '>' comprises 3 characters. The regular expression is annotated as follows. The first function is $\langle(p+s)\rangle$. Due to rule (a), the second function is, rr , which in this case is either $\langle(p+s)$ or $(p+s)\rangle$. However, rule (b) applies and we select $\langle(p+s)$ as the second function. The third function is, $r+r, (p+s)$. Thus, 1 is assigned to $\langle(p+s)\rangle$, 2 is assigned to $\langle(p+s)$, 3 is assigned to $\langle$, 4 is assigned to $(p+s)$, 5 is assigned to p , 6 is assigned to s , and 7 is assigned to $\rangle$. Thus, the annotation for $r = 1:(2:(3:(4:(5:p+6:s))7:\rangle)$.

From the derivation tree presented in figures 19-24, we derive at the following partial derivatives and refinement environments:

$$\begin{array}{ll} (1) \ 2:3:4:(5:p+6:s)7: > & \{(2, id_W), (2, 8:i+\overset{\circ}{W}), (2, 9: > +\overset{\circ}{W})\} \\ (2) \ 3:5:\epsilon 7: > & \{(2, id_W), (5, 8:i+\overset{\circ}{S}), (5, 10: > +\overset{\circ}{W})\} \\ (3) \ 7:\epsilon & \{(2, id_W), (5, 8:i+\overset{\circ}{S}), (7, id_W)\} \\ (4) \ 3:6:\epsilon 7: > & \{(2, id_W), (6, 8:i+\overset{\circ}{S}), (6, 10: > +\overset{\circ}{W})\} \\ (5) \ 7:\epsilon & \{(2, id_W), (6, 8:i+\overset{\circ}{S}), (7, id_W)\} \end{array}$$

Figure 19 illustrates the final result of the derivation tree. In particular, the bottom part of the figure 19 shows the final result. The top left part illustrates the process for the first character, "<". The top left of figure 20 and figure 24 illustrate the process for the second character "i". Figures 21-23 illustrate the process for the third character ">".

Now we need to apply the remaining steps found in the *refineAlgo*. First, for cases (1), (2) and (4), the partial derivatives are not possessing ϵ , hence we need to add $a+\epsilon$ operation to the respective environments.

$$\begin{aligned}
(1) \quad & 2:3:4:(5:p + 6:s)7: > \{(2, id_W), (2, 8:i + \overset{\circ}{W}), (2, 9: > + \overset{\circ}{W}), (5, +^\epsilon)\} \\
(2) \quad & 3:5:\epsilon 7: > \{(2, id_W), (5, 8:i + \vec{S}), (5, 10: > + \overset{\circ}{W}), (7, +^\epsilon)\} \\
(3) \quad & 7:\epsilon \{(2, id_W), (5, 8:i + \vec{S}), (7, id_W)\} \\
(4) \quad & 3:6:\epsilon 7: > \{(2, id_W), (6, 8:i + \vec{S}), (6, 10: > + \overset{\circ}{W}), (7, +^\epsilon)\} \\
(5) \quad & 7:\epsilon \{(2, id_W), (6, 8:i + \vec{S}), (7, id_W)\}
\end{aligned}$$

Next we apply the $<$ ordering among the refinement environments. We conclude that (3) and (5) are minimal refinements. Being concrete, to fix the pattern, we either add a
5 i transition to source location 1 or 2, that is

$$1:2: < 3:4:(5:(9:p + 8:i) + 6:s)7: >$$

or

$$1:2: < 3:4:(5:p + 6:(9:s + 8:i))7: >$$

10

Figure 25 illustrates a flow diagram of process 2500 performed by the processor in processing system 200 in accordance with an embodiment of this invention. Particularly, process 2500 is performed by the refinement subsystem 130 in accordance with an embodiment of this invention. Process 2500 is a process for refining the regular expression
15 in the extraction pattern database 140 and is executable upon receiving a mismatch result in process 320. Process 2500 begins with step 2505 by determining the number of characters in the data 110 received.

In step 2507, process 1100 determines the number of annotations in the regular
20 expression. Particularly, each annotation contains at least one function as illustrated above in examples 1 and 2.

Process 2500 then selects a first character from the data in step 2510. The following steps are repeated for each of the characters in the data.

25

In step 2515, process 2500 determines the rule or rules based on the annotated regular expression and the input character. Details on the process of the annotation are described above.

In step 2520, the rules are executed accordingly. Particularly, the rules are executed from rule from the highest annotation to rule from the lowest annotation. Details on the execution of the rules are illustrated above in figures 14-16 and example 4.

5 In step 2525, process 2500 generates regular expression partial derivative and appends the refinement operation derived from the execution of the determined rules.

10 In step 2530, process 2500 determines whether the selected character is the last character in the data. If the selected character is not the last character in the data, process 2500 proceeds to step 2535 to select the next character and repeats from step 2515. Step 2515 is required to determine the rule or rules based on the annotated regular expression and the input character. Process 2500 proceeds to step 2540 if the selected character is the last character in the data. In step 2540, process 2500 applies the refinement operations to the regular expression and store the refined regular expression in the database 140. The
15 refinement process is described above with reference to figures 17-18.

It is possible to apply the refinement subsystem without the second phase of the error detection extension. However, it is noted that the refinement subsystem requires extensive computing resource and inevitably also requires substantial amount of time.
20 Hence, the second phase of the error detection extension which requires less computing resource is in placed to trigger the refinement subsystem as and when required.

It is noted that the first three steps in process 1000, 1100 and 2500 are identical. Hence, the number of characters in the data determined and number of annotations in
25 regular expression determined may only be performed in process 1000 and stored in memory so that the details may be retrieved with processing processes 1100 and 2500.

It is further noted that the examples above illustrate the processes via data with fewer than 4 characters. One skilled in the art will recognise that the data are typically
30 much longer than 4 characters and the above examples are only meant to illustrate the processes in a simplified form.

The above is a description of exemplary embodiments of a system and method in accordance with this invention. It is foreseeable that those skilled in the art can and will design alternative systems based on this disclosure that infringe upon this invention as set forth in the following claims.

5

Definition

$\notin$ - is not an element of

$\in$ - is an element of

$\forall$ - for all

10 $\oplus$ - an update operation for refinement requirement

$\cup$ - Union

Δ - a regular expression sub-matching value binding environment

$\vdash$ - infers, derivable from

$\Leftrightarrow$ - if and only if

15 $\mapsto$ - maps to

$\emptyset$ - empty set

$<$ - a partial order operator among refinement environment

20

Claims:

1. A computer-implemented method of refining rules for extracting information comprising:

5 receiving data;
retrieving a regular expression from an extraction pattern database;
annotating said regular expression;
applying a first phase of error detection extension having a plurality of first set of rules by subjecting said data to said annotated regular expression and said plurality of first
10 set of rules;

generating a match result from said application of said first phase of error detection extension;

applying a second phase of error detection extension having a plurality of second set of rules by subjecting said data to said annotated regular expression and said plurality
15 of second set of rules;

generating a mismatch result from said application of said second phase of error detection extension in response to a mismatch in said data;

applying a refinement process to said data in response to generating said mismatch report from said second phase of extraction of information from said data.

20 2. The computer-implemented method of claim 1 wherein said step of annotating said regular expression comprises:

determining functions in said regular expression starting from the outermost function to the innermost function; and

25 assigning annotation from left to right.

3. The computer-implemented method of claim 2 wherein said step of determining functions in said regular expression comprises:

adding parenthesis to each determine function.

30 4. The computer-implemented method of claim 3 wherein said step of assigning annotation from left to right comprises:

assigning in ascending order from the left to the right of the determined functions.

5. The computer-implemented method of claim 1 wherein said step of applying a first phase of error detection extension comprises:

- i) determining number of characters in said data;
- ii) determining number of annotations in said annotated regular expression;
- iii) selecting a first character;
- iv) determining a rule among said plurality of first set rules based on annotated regular expression and said selected character;
- v) executing determined rules;
- vi) generating a regular expression partial derivative and append trace;
- vii) repeating steps (iv)-(vi) for each character in said data; and
- viii) generating a match result.

6. The computer-implemented method of claim 5 wherein said plurality of first set of rules comprises:

$$\boxed{\bar{i} \vdash r/l \Rightarrow (\overline{r'}, \bar{i}')} \quad \boxed{\bar{i} \vdash r/w \Rightarrow (\overline{r'}, \bar{i}')}$$

$$(Eps) \quad \bar{i} \vdash \epsilon/l \Rightarrow \{\} \quad (Lab1) \quad \bar{i} \vdash i:l/l \Rightarrow \{(i:\epsilon, \bar{i} + i)\} \quad (Lab2) \quad \bar{i} \vdash i:l/l' \Rightarrow \{\}$$

$$(Seq1) \quad \frac{\epsilon \in r_1 \quad \bar{i} \vdash r_1/l \Rightarrow (\overline{r'_1}, \bar{j}) \quad \bar{i} \vdash r_2/l \Rightarrow (\overline{r'_2}, \bar{j}')}{\bar{i} \vdash i:r_1r_2/l \Rightarrow \{(r'_1r_2, \bar{j}) \mid (r'_1, \bar{j}) \in (\overline{r'_1}, \bar{j})\} \cup (\overline{r'_2}, \bar{j}')}$$

$$(Seq2) \quad \frac{\epsilon \notin r_1 \quad \bar{i} \vdash r_1/l \Rightarrow (\overline{r'_1}, \bar{j})}{\bar{i} \vdash i:r_1r_2/l \Rightarrow (\overline{r'_1}, \bar{j})}$$

$$(Choice) \quad \frac{\bar{i} \vdash r_1/l \Rightarrow (\overline{r'_1}, \bar{j}) \quad \bar{i} \vdash r_2/l \Rightarrow (\overline{r'_2}, \bar{j}')}{\bar{i} \vdash i:r_1 + r_2/l \Rightarrow (\overline{r'_1}, \bar{j}) \cup (\overline{r'_2}, \bar{j}')}$$

$$(Star) \quad \frac{\bar{i} \vdash r/l \Rightarrow (\overline{r'}, \bar{i}')}{\bar{i} \vdash i:r^*/l \Rightarrow \{(i:r' i, \bar{i}') \mid (r', \bar{i}') \in (\overline{r'}, \bar{i}')\}}$$

$$\boxed{\bar{i} \vdash r/w \Rightarrow (\overline{r'}, \bar{i}')} \quad \boxed{\bar{i} \vdash r/\epsilon \Rightarrow \{(r, \bar{i})\}}$$

$$(WordEps) \quad \bar{i} \vdash r/\epsilon \Rightarrow \{(r, \bar{i})\}$$

$$(WordCon) \quad \frac{\bar{i} \vdash r/l \Rightarrow (\overline{r'}, \bar{i}') \quad \text{for } j \in \{1, \dots, n\} \quad \bar{i}'_j \vdash r'_j/w \Rightarrow (\overline{r''_j}, \bar{i}''_j)}{\bar{i} \vdash r/lw \Rightarrow \bigcup_j (\overline{r''_j}, \bar{i}''_j)}$$

Where

$\bar{l}$ is the trace associated to the source location, r is the regular expression, l is a literal of said data, w is a string of literals of said data;

rule (Eps) takes a trace i , an input expression ϵ , an input regular expression literal l , and returns an empty set when there is a mismatch between the input expression ϵ and the regular expression literal l ;

rule (Lab1) takes a trace i , an input regular expression literal l , a matching literal expression $i: l$, and append the matching source location i to the trace $\bar{l}$;

rule (Lab2) takes a trace i , an input literal l , another literal, literal prime l' and returns an empty set when there is a mismatch between l and l' ;

rule (Seq1) takes a trace i , an input regular expression literal l , a first of a sequence of expression r_1 , and return with $\overline{(r'_1, \bar{j})}$, which denotes a set of possible partial derivatives and trace by taking away l from r_1 , and further takes a second of the sequence of expression, r_2 , and return with $\overline{(r'_2, \bar{j})}$, which denotes a set of possible partial derivatives and trace by taking away l from r_2 , said rule (Seq1) is applicable where input expression ϵ is an element of r_1 ;

rule (Seq2) takes a trace i , an input literal l , a first of a sequence of expression r_1 , and return with $\overline{(r'_1, \bar{j})}$, which denotes a set of possible partial derivatives and trace by taking away l from r_1 , said rule (Seq2) is applicable where empty input sequence ϵ is not an element of r_1 ;

rule (Choice) takes a trace i , an input literal l , and a first of the two regular expressions r_1 , and returns with $\overline{(r'_1, \bar{j})}$, which denotes a set of possible partial derivatives and trace by taking away l from r_1 and further takes a trace i , an input literal l , and a second of the two regular expressions r_2 , and returns with $\overline{(r'_2, \bar{j})}$, which denotes a set of possible partial derivatives and trace by taking away l from r_2 ;

rule (Star) takes a trace i , a regular expression r , an input literal l , and returns with $\overline{(r', \bar{l'})}$, which denotes a set of possible partial derivatives and trace by taking away l from r' ;

rule (WordEps) takes a trace i , a regular expression r , an empty input expression ϵ , and returns $\{(r, \bar{l'})\}$; and

rule (WordCon) is separated into two parts, the first part takes a trace i , a regular expression r , a first literal from the string of literal l , and returns $\overline{(r', \bar{l'})}$, the second part

takes an appended trace from the first part $\bar{l}_j$, an intermediate partial derivative regular expression from the first part r_j' , the rest of the string of literal w , and returns with $(\overline{r_j''}, \bar{l}_j'')$, which is the final result for this rule by aggregating across all the j in $\{1...n\}$.

7. The computer-implemented method of claim 6 wherein said step of applying said second phase of error detection extension comprises:

i) determining a rule among said plurality of second set rules based on annotated regular expression and said selected character;

ii) executing determined rules;

iii) generating a regular expression partial derivative and append matching error;

iv) repeating steps (i)-(iii) for each character in said data; and

v) generating a mismatch result.

8. The computer-implemented method of claim 7 wherein said a plurality of second set of rules comprises:

$$(\text{Match Error}) \ E ::= E_l^{\bar{}} \mid E_e^{\bar{}}$$

$$\boxed{\bar{i} \vdash r/l \Rightarrow (\overline{r'}, \bar{i}'), \bar{E}}$$

$$(\text{Eps}) \ \bar{i} \vdash \epsilon/l \Rightarrow \{\}, \{E_e^{\bar{}}\}$$

$$(\text{Lab1}) \ \bar{i} \vdash i:l/l \Rightarrow \{(i, \epsilon, \bar{i} + i)\}, \{\} \quad (\text{Lab2}) \ \bar{i} \vdash i:l/l' \Rightarrow \{\}, \{E_l^{\bar{}}\}$$

$$(\text{Seq1}) \ \frac{\epsilon \in r_1 \quad \bar{i} \vdash r_1/l \Rightarrow (\overline{r'_1}, \bar{j}), \bar{E}_1 \quad \bar{i} \vdash r_2/l \Rightarrow (\overline{r'_2}, \bar{j}'), \bar{E}_2}{\bar{i} \vdash i:r_1r_2/l \Rightarrow \{(r'_1r_2, \bar{j}) \mid (r'_1, \bar{j}) \in (\overline{r'_1}, \bar{j})\} \cup (\overline{r'_2}, \bar{j}'), \bar{E}_1 \cup \bar{E}_2}$$

$$(\text{Seq2}) \ \frac{\epsilon \notin r_1 \quad \bar{i} \vdash r_1/l \Rightarrow (\overline{r'_1}, \bar{j}), \bar{E}'}{\bar{i} \vdash i:r_1r_2/l \Rightarrow (\overline{r'_1}, \bar{j}), \bar{E}'}$$

$$(\text{Choice}) \ \frac{\bar{i} \vdash r_1/l \Rightarrow (\overline{r'_1}, \bar{j}), \bar{E}_1 \quad \bar{i} \vdash r_2/l \Rightarrow (\overline{r'_2}, \bar{j}'), \bar{E}_2}{\bar{i} \vdash i:r_1 + r_2/l \Rightarrow (\overline{r'_1}, \bar{j}) \cup (\overline{r'_2}, \bar{j}'), \bar{E}_1 \cup \bar{E}_2}$$

$$(\text{Star}) \ \frac{\bar{i} \vdash r/l \Rightarrow (\overline{r'}, \bar{i}'), \bar{E}'}{\bar{i} \vdash i:r^*/l \Rightarrow \{(i, r', \bar{i}') \mid (r', \bar{i}') \in (\overline{r'}, \bar{i}')\}, \bar{E}'}$$

$$\boxed{\bar{i}, \bar{E} \vdash r/w \Rightarrow (\overline{r'}, \bar{i}'), \bar{E}'}$$

$$(\text{WordEps}) \ \bar{i}, \bar{E} \vdash r/\epsilon \Rightarrow \{(r, \bar{i})\}, \bar{E}$$

$$(\text{WordCon}) \ \frac{\bar{i} \vdash r/l \Rightarrow (\overline{r'}, \bar{i}'), \bar{E}' \quad \text{for } j \in \{1, \dots, n\} \quad \bar{i}_j, \bar{E}'_j \vdash r_j/w \Rightarrow (\overline{r''_j}, \bar{i}''_j), \bar{E}''_j}{\bar{i}, \bar{E} \vdash r/lw \Rightarrow \bigcup_j (\overline{r''_j}, \bar{i}''_j), \bar{E}' \cup \bigcup_j \bar{E}''_j}$$

wherein said plurality of second set of rules is an extension of said plurality of first set of rules to track literal mismatch error $E_l^{\bar{l}}$ and the epsilon mismatch error $E_e^{\bar{l}}$;

rule (Eps) raises an epsilon mismatch error;

rule (Lab2) raises a literal mismatch error;

5 rule (Lab1) consumes an input literal with the matching literal expression and append the matching source location i to the trace $\bar{l}$;

rules (Seq1) and (Seq2) handle sequence expression r_1r_2 , rule (Seq1) is applicable where $\epsilon \in r_1$, otherwise rule (Seq2) is applied;

rule (Choice) handles choice expression; and

10 rule (Star) handles the Kleene's star expression.

9. The computer-implemented method of claim 8 wherein said step of applying said refinement process to said data comprises:

15 i) determining a rule among said plurality of third set rules based on annotated regular expression and said selected character;

ii) executing determined rules;

iii) generating a regular expression partial derivative and refinement operation;

iv) repeating steps (i)-(iii) for each character in said data; and

v) applying refinement operations to said regular expression

20 vi) updating said refined regular expression in said extraction pattern database.

10. The computer-implemented method of claim 9 wherein said refinement operation comprises:

25 state-extension where a literal is added to the regular expression and the last two literals in the regular expression are finalised;

transition-extension where a new transition is added to the regular expression;

state-finalisation where the last second literal of the regular expression is finalised;

and

no-change where no changes are made.

30

11. The computer-implemented method of claim 10 wherein said plurality of third set rules comprises:

(Label) $l ::= 'a' \mid 'b' \mid \dots$
 (Level) $L ::= W \mid S$
 (Refinement Environment) $\Theta ::= \{(i, op), \dots, (i, op)\}$
 (Refinement Operation) $op ::= i.l +_L^\circ \mid i.l +_L^\rightarrow \mid +^\epsilon \mid id_L$
 (Source Location Aliases) $\mathcal{A} ::= \{i \mapsto j, \dots, i \mapsto j\}$

$$\boxed{\mathcal{R} \vdash_{\mathcal{A}}^L r/l \Rightarrow (\mathcal{R}, r, \Theta, \mathcal{A})}$$

$$(EpsS) \frac{(i, r) \in \mathcal{R} \quad j \text{ is a fresh src loc}}{\mathcal{R} \vdash_{\mathcal{A}}^L i:\epsilon/l \Rightarrow \{(\mathcal{R} \oplus \{(i, r')\}, j:\epsilon, \{(i, j:l +_S^\circ)\}, \mathcal{A}) \mid r' \in r^{pl}\}}$$

$$(EpsW) \frac{i \notin \text{dom}(\mathcal{R}) \quad j \text{ is a fresh src loc} \quad L' = \max(L, W)}{\mathcal{R} \vdash_{\mathcal{A}}^L i:\epsilon/l \Rightarrow \{(\mathcal{R}, j:\epsilon, \{(i, j:l +_{L'}^\circ)\}, \mathcal{A})\}}$$

$$(LitS1) \frac{(i, r) \in \mathcal{R}}{\mathcal{R} \vdash_{\mathcal{A}}^L i:l/l \Rightarrow \{(\mathcal{R} \oplus \{(i, r')\}, i:\epsilon, \{(i, id_S)\}, \mathcal{A}) \mid r' \in r^{pl}\}}$$

$$(LitS2) \frac{(i, r) \in \mathcal{R} \quad l \neq l' \quad j \text{ is a fresh src loc}}{\mathcal{R} \vdash_{\mathcal{A}}^L i:l/l' \Rightarrow \{(\mathcal{R} \oplus \{(i, r')\}, i:\epsilon, \{(i, j:l' +_S^\rightarrow)\}, \mathcal{A}) \mid r' \in r^{pl}\}}$$

$$(LitW1) \frac{i \notin \text{dom}(\mathcal{R}) \quad L' = \max(L, W)}{\mathcal{R} \vdash_{\mathcal{A}}^L i:l/l \Rightarrow \{(\mathcal{R}, i:\epsilon, \{(i, id_{L'})\}, \mathcal{A})\}}$$

$$(LitW2) \frac{i \notin \text{dom}(\mathcal{R}) \quad l \neq l' \quad L' = \max(L, W) \quad j \text{ is a fresh src loc}}{\mathcal{R} \vdash_{\mathcal{A}}^L i:l/l' \Rightarrow \{(\mathcal{R}, i:\epsilon, \{(i, j:l' +_{L'}^\rightarrow)\}, \mathcal{A})\}}$$

$$(SeqS1) \frac{\begin{array}{l} (i, r) \in \mathcal{R} \quad \epsilon \in r_1 \quad \mathcal{R}_{\overline{r_1}} \vdash_{\mathcal{A}}^S r_1/l \Rightarrow V_1 \quad \mathcal{R}_{\overline{r_2}} \vdash_{\mathcal{A}}^S r_2/l \Rightarrow V_2 \\ V'_1 = \{(\mathcal{R}'_1 \cup \mathcal{R}_{\overline{r_2}} \cup \{(i, r')\}, i:(r'_1 r_2), \Theta, \mathcal{A}) \mid (\mathcal{R}'_1, r'_1, \Theta, \mathcal{A}) \in V_1, r' \in r^{pl}\} \\ V'_2 = \{(\mathcal{R}'_2 \cup \{(i, r')\}, r'_2, \Theta, \mathcal{A}) \mid (\mathcal{R}'_2, r'_2, \Theta, \mathcal{A}) \in V_2, r' \in r^{pl}\} \end{array}}{\mathcal{R} \vdash_{\mathcal{A}}^L i:(r_1 r_2)/l \Rightarrow V'_1 \cup V'_2}$$

$$(SeqS2) \frac{\begin{array}{l} (i, r) \in \mathcal{R} \quad \epsilon \notin r_1 \quad \mathcal{R}_{\overline{r_1}} \vdash_{\mathcal{A}}^S r_1/l \Rightarrow V_1 \quad \mathcal{R}_{\overline{r_2}} \vdash_{\mathcal{A}}^S r_2/l \Rightarrow V_2 \\ V'_1 = \{(\mathcal{R}'_1 \cup \mathcal{R}_{\overline{r_2}} \cup \{(i, r')\}, i:(r'_1 r_2), \Theta, \mathcal{A}) \mid (\mathcal{R}'_1, r'_1, \Theta, \mathcal{A}) \in V_1, r' \in r^{pl}\} \\ V'_2 = \{(\mathcal{R}'_2 \cup \{(i, r')\}, r'_2, \Theta \cup \Theta', \mathcal{A}) \mid (\mathcal{R}'_2, r'_2, \Theta, \mathcal{A}) \in V_2, \Theta' \in \text{mkfin}(r_1)\} \end{array}}{\mathcal{R} \vdash_{\mathcal{A}}^L i:(r_1 r_2)/l \Rightarrow V'_1 \cup V'_2}$$

$$\begin{array}{c}
\text{(SeqW1)} \quad \frac{i \notin \text{dom}(\mathcal{R}) \quad \epsilon \in r_1 \quad \mathcal{R}_{\overline{r_1}} \vdash_{\mathcal{A}}^L r_1/l \Rightarrow V_1 \quad \mathcal{R}_{\overline{r_2}} \vdash_{\mathcal{A}}^L r_2/l \Rightarrow V_2 \quad \mathcal{R}'_2 = \mathcal{R}_{\overline{r_2}}}{\mathcal{R} \vdash_{\mathcal{A}}^L i:(r_1 r_2)/l \Rightarrow \{(\mathcal{R}'_1 \cup \mathcal{R}'_2, i:(r'_1 r_2), \Theta) \mid (\mathcal{R}'_1, r'_1, \Theta, \mathcal{A}) \in V_1\} \cup V_2} \\
\\
\text{(SeqW2)} \quad \frac{i \notin \text{dom}(\mathcal{R}) \quad \epsilon \notin r_1 \quad \mathcal{R}_{\overline{r_1}} \vdash_{\mathcal{A}}^L r_1/l \Rightarrow V_1 \quad \mathcal{R}'_2 = \mathcal{R}_{\overline{r_2}} \cup \{(i, r')\} \quad V'_1 = \{(\mathcal{R}'_1 \cup \mathcal{R}'_2, i:(r'_1 r_2), \Theta, \mathcal{A}) \mid (\mathcal{R}'_1, r'_1, \Theta, \mathcal{A}) \in V_1\} \quad V'_2 = \{(\mathcal{R}'_2 \cup \{(i, r')\}, r'_2, \Theta \cup \Theta', \mathcal{A}) \mid (\mathcal{R}'_2, r'_2, \Theta, \mathcal{A}) \in V_2, \Theta' \in \text{mkfin}(r_1)\}}{\mathcal{R} \vdash_{\mathcal{A}}^L i:(r_1 r_2)/l \Rightarrow V'_1 \cup V'_2} \\
\\
\text{(ChoiceS)} \quad \frac{(i, r) \in \mathcal{R} \quad \mathcal{R} \vdash_{\mathcal{A}}^S r_1/l \Rightarrow V_1 \quad \mathcal{R} \vdash_{\mathcal{A}}^S r_2/l \Rightarrow V_2 \quad V'_1 = \{(\mathcal{R}'_1 \cup \{(i, r')\}, r'_1, \Theta, \mathcal{A}) \mid (\mathcal{R}'_1, r'_1, \Theta, \mathcal{A}) \in V_1, r' \in r^{pl}\} \quad V'_2 = \{(\mathcal{R}'_2 \cup \{(i, r')\}, r'_2, \Theta, \mathcal{A}) \mid (\mathcal{R}'_2, r'_2, \Theta, \mathcal{A}) \in V_2, r' \in r^{pl}\}}{\mathcal{R} \vdash_{\mathcal{A}}^L i:(r_1 + r_2)/l \Rightarrow V'_1 \cup V'_2} \\
\\
\text{(ChoiceW)} \quad \frac{i \notin \text{dom}(\mathcal{R}) \quad \mathcal{R} \vdash_{\mathcal{A}}^L r_1/l \Rightarrow V_1 \quad \mathcal{R} \vdash_{\mathcal{A}}^L r_2/l \Rightarrow V_2}{\mathcal{R} \vdash_{\mathcal{A}}^L i:(r_1 + r_2)/l \Rightarrow V_1 \cup V_2} \\
\\
\text{(StarS)} \quad \frac{(i, r_0) \in \mathcal{R} \quad \text{relabel}(r) = (r', \mathcal{A}') \quad \mathcal{A}'(\mathcal{R}) \vdash_{\mathcal{A} \cup \mathcal{A}'}^S r'/l \Rightarrow V}{\mathcal{R} \vdash_{\mathcal{A}}^L i:r^*/l \Rightarrow \{(\mathcal{R}' \oplus (i, r'_0), i:(r'' i:r^*), \Theta, \mathcal{A}'') \mid (\mathcal{R}', r'', \Theta, \mathcal{A}'') \in V, r'_0 \in r_0^{pl}\}} \\
\\
\text{(StarW)} \quad \frac{i \notin \text{dom}(\mathcal{R}) \quad \text{relabel}(r) = (r', \mathcal{A}') \quad \mathcal{A}'(\mathcal{R}) \vdash_{\mathcal{A} \cup \mathcal{A}'}^L r'/l \Rightarrow V}{\mathcal{R} \vdash_{\mathcal{A}}^L i:r^*/l \Rightarrow \{(\mathcal{R}', i:(r''(i:r^*)), \Theta, \mathcal{A}'') \mid (\mathcal{R}', r'', \Theta, \mathcal{A}'') \in V\}} \\
\\
\boxed{\mathcal{R} \vdash_{\mathcal{A}}^L r/w \Rightarrow \{R, r, \Theta, \mathcal{A}\}} \\
\\
\text{(WordCon)} \quad \frac{\mathcal{R} \vdash_{\mathcal{A}}^L r/l \Rightarrow \{R, r, \Theta, \mathcal{A}\} \quad \text{for } i \in \{1, \dots, n\} \quad \mathcal{R}_i \vdash_{\mathcal{A} \cup \mathcal{A}_i}^L r_i/w \Rightarrow V_i \quad V'_i = \{(\mathcal{R}_j, r_j, \Theta_i \cup \Theta_j, \mathcal{A}_j) \mid (\mathcal{R}_j, r_j, \Theta_j, \mathcal{A}_j) \in V_i\}}{\mathcal{R} \vdash_{\mathcal{A}}^L r/lw \Rightarrow V'_1 \cup \dots \cup V'_n} \\
\\
\text{(WordEps)} \quad \mathcal{R} \vdash_{\mathcal{A}}^L r/\epsilon \Rightarrow \{(\mathcal{R}, r, \{\}, \mathcal{A})\}
\end{array}$$

wherein judgement $R \vdash_{\mathcal{A}}^L r/l \Rightarrow \overline{(R, r, \theta, \mathcal{A})}$ extends from the plurality of first set of rules, L denotes the levels of precedence, range over strong (S) and weak (W), $\mathcal{A}$ denotes an alternative indication to the source location, θ denotes the refinement environment for each source location, i , and a corresponding refinement operation, $op, i: l +_L^o$ refers to

state-extension, $i:l + \vec{l}$ refers to transition-extension, $+^\epsilon$ refers to state-finalisation, and id_L refers to no-change,

rule (EpsW) proposes a fix to epsilon-literal mismatch by adding a state-extension, in case where the current source location i is not in the domain of the refinement requirement,

rule (EpsS) ensures that $(i:r) \in R$ and r can consume the input literal l ,

rules (LitS1) and (LitW1) handle cases where the input literal matches with the regular expression l where a no-change refinement operation is added,

rules (LitS2) and (LitS1) update the user requirement environment R with $(i:r')$

where r' is the partial derivative of r and $(i:r) \in R$,

rules (SeqS1), (SeqS2), (SeqW1) and (SeqW2) handle cases where the regular expression to be refined is a concatenation,

rules (ChoiceS) and (ChoiceW) handle cases where the regular expression to be refined is a choice,

rules (StarS) and (StarW) handle cases where the regular expression to be refined is a Kleene's star, and

rules (WordCon) and (WordEps) extend the refinement operation to support a sequence of literal input.

13. The computer-implemented method of claim 12 wherein said step of applying refinement operations to said regular expression comprises:

applying the refinement operation obtained to the input word w ;

dividing result from the application of the refinement operation obtained to the input word into two categories, namely $\bar{\theta}$ and $\bar{\theta}'$ where $\bar{\theta}$ contains refinement environments that makes resulting partial derivative r' final, where $\bar{\theta}'$ contains refinement environments that makes the resulting partial derivative r' non-final;

determine the refinement environment from $\bar{\theta} \cup \bar{\theta}'$ with the fewest changes to the original regular expression; and

applying the refinement environment with the fewest changes to the original regular expression.