(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) **World Intellectual Property Organization**
International Bureau

(43) **International Publication Date**
21 June 2012 (21.06.2012)

WIPO | PCT

(10) **International Publication Number**
WO 2012/082362 A1

(51) **International Patent Classification**:
G06F 17/30 (2006.01)

(21) **International Application Number**:
PCT/US2011/062479

(22) **International Filing Date**:
29 November 2011 (29.11.2011)

(25) **Filing Language**: English

(26) **Publication Language**: English

(30) **Priority Data**:
12/968,618 15 December 2010 (15.12.2010) US

(71) **Applicant** *(for all designated States except US)*: **MICROSOFT CORPORATION** [US/US]; One Microsoft Way, Redmond, WA 98052-6399 (US).

(72) **Inventors**: **LAMANNA, Charles William**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, WA 98052-6399 (US). **GANDHI, Mauktik H.**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, WA 98052-6399 (US). **BREWER, Jason Eric**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, WA 98052-6399 (US).

(81) **Designated States** *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) **Designated States** *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:

— *with international search report (Art. 21(3))*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

(54) Title: DECOMPOSING AND MERGING REGULAR EXPRESSIONS

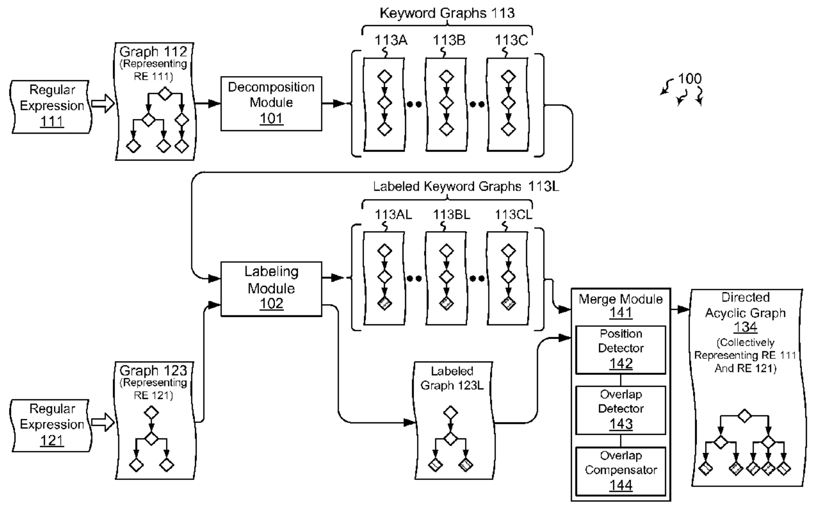


Figure 1

(57) **Abstract**: The present invention extends to methods, systems, and computer program products for decomposing and merging regular expressions. Embodiments of the invention decompose a regular expression into multiple simple keyword graphs, merge those keyword graphs in a compact and efficient manner, and produce a directed acyclic graph (DAG) that can execute a simplified regular expression alphabet. Several of these regular expression DAG's can then be merged together to produce a single DAG that represents an entire collection of regular expressions. DAGs along with other text processing algorithms and a heap collection can be combined in a multi-pass approach to expand the regular expression alphabet.

DECOMPOSING AND MERGING REGULAR EXPRESSIONS

BACKGROUND

[0001] Computer systems and related technology affect many aspects of society. Indeed, the computer system's ability to process information has transformed the way we live and work. Computer systems now commonly perform a host of tasks (e.g., word processing, scheduling, accounting, etc.) that prior to the advent of the computer system were performed manually. More recently, computer systems have been coupled to one another and to other electronic devices to form both wired and wireless computer networks over which the computer systems and other electronic devices can transfer electronic data. Accordingly, the performance of many computing tasks are distributed across a number of different computer systems and/or a number of different computing environments.

[0002] In some computing environments, regular expressions are used to match strings of text, such as, for example, particular characters, words, or patterns of characters. Regular expressions can be written in a formal language that can be interpreted by a regular expression processor. The regular expression processor is a program that serves as a parser generator or examines text and identifies parts that match a provided specification.

[0003] Regular expressions are used by many text editors, utilities, and programming languages to search and manipulate text based on patterns. For example, anti-spam services can utilize regular expressions to determine if strings of text known to be indicative of SPAM are contained in an electronic message. Similarly, data leakage protection services can utilize regular expressions to detect and prevent the unauthorized use and transmission of confidential information.

[0004] In environments that utilize regular expressions, it is not uncommon for large collections of regular expressions to be executed sequentially. For example, an anti-spam service can use tens of thousands of regular expressions when determining if an electronic message contains SPAM. Regular expressions within a set of regular expressions can be run sequentially against each received electronic message. Sequential execution of regular expressions limits scalability and can consume significant resources as the number of regular expressions and/or portions of text being checked for matches increases.

BRIEF SUMMARY

[0005] The present invention extends to methods, systems, and computer program products for decomposing and merging regular expressions. One or more keyword graphs are accessed. The one or more keyword graphs were decomposed from a first regular expression. Each of the one or more keyword graphs has a root node, one or more intermediate nodes, and a leaf node. Each of the one or more intermediate nodes and the leaf node indentify a character pattern that partially matches the first regular expression. The root node and each of the one or more intermediate nodes have a single child node. One of the intermediate nodes has the leaf node as a child node. Each leaf node is labeled as a matching state for the first regular expression.

[0006] A second graph is accessed. The second graph represents a second regular expression. The second graph has a root node, one or more intermediate nodes, and one or more leaf nodes. Each of the one or more intermediate nodes and the one or more leaf nodes indentify a character pattern that partially matches the second regular expression. The second graph has one or more terminal nodes labeled as a matching state for the second regular expression.

[0007] The one or more keyword graphs and the second graph are merged into a directed acyclic graph that collectively represents both the first regular expression and the second regular expression. Merging includes identifying any similarly positioned intermediate nodes within the one or more keyword graphs and the second graph that have at least partially overlapping character patterns. For any identified intermediate nodes that have partially overlapping character patterns, the character pattern of at least one of the indentified intermediate nodes is altered to eliminate the partially overlapping character pattern. An edge is added between the keyword graph and the second graph to compensate for altering the character pattern of the at least one of the identified intermediate nodes. For any identified intermediate nodes that have fully overlapping character patterns, the intermediate node in the keyword graph and the intermediate node in the second graph are combined into a single node representing the fully overlapping character pattern.

[0008] This summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This Summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used as an aid in determining the scope of the claimed subject matter.

[0009] Additional features and advantages of the invention will be set forth in the description which follows, and in part will be obvious from the description, or may be learned by the practice of the invention. The features and advantages of the invention may be realized and obtained by means of the instruments and combinations particularly pointed out in the appended claims. These and other features of the present invention will become more fully apparent from the following description and appended claims, or may be learned by the practice of the invention as set forth hereinafter.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] In order to describe the manner in which the above-recited and other advantages and features of the invention can be obtained, a more particular description of the invention briefly described above will be rendered by reference to specific embodiments thereof which are illustrated in the appended drawings. Understanding that these drawings depict only typical embodiments of the invention and are not therefore to be considered to be limiting of its scope, the invention will be described and explained with additional specificity and detail through the use of the accompanying drawings in which:

[0011] Figure 1 illustrates an example computer architecture that facilitates decomposing and merging regular expressions.

[0012] Figure 2 illustrates an example of decomposing a graph that represents a regular expression.

[0013] Figures 3 illustrate an example of merging graphs that represent different regular expressions.

[0014] Figure 4 illustrates another example of decomposing a graph that represents a regular expression.

[0015] Figure 5 illustrates another example of merging graphs that represent different regular expressions.

[0016] Figure 6 illustrates a flow chart of an example method for decomposing and merging regular expressions.

DETAILED DESCRIPTION

[0017] The present invention extends to methods, systems, and computer program products for decomposing and merging regular expressions. One or more keyword graphs are accessed. The one or more keyword graphs were decomposed from a first regular expression. Each of the one or more keyword graphs has a root node, one or more intermediate nodes, and a leaf node. Each of the one or more intermediate nodes and the leaf node indentify a character pattern that partially matches the first regular expression. The root node and each of the one or more intermediate nodes have a single child node. One of the intermediate nodes has the leaf node as a child node. Each leaf node is labeled as a matching state for the first regular expression.

[0018] A second graph is accessed. The second graph represents a second regular expression. The second graph has a root node, one or more intermediate nodes, and one or more leaf nodes. Each of the one or more intermediate nodes and the one or more leaf nodes indentify a character pattern that partially matches the second regular expression. The second graph has one or more terminal nodes labeled as a matching state for the second regular expression.

[0019] The one or more keyword graphs and the second graph are merged into a directed acyclic graph that collectively represents both the first regular expression and the second regular expression. Merging includes identifying any similarly positioned intermediate nodes within the one or more keyword graphs and the second graph that have at least partially overlapping character patterns. For any identified intermediate nodes that have partially overlapping character patterns, the character pattern of at least one of the indentified intermediate nodes is altered to eliminate the partially overlapping character pattern. An edge is added between the keyword graph and the second graph to compensate for altering the character pattern of the at least one of the identified intermediate nodes. For any identified intermediate nodes that have fully overlapping character patterns, the intermediate node in the keyword graph and the intermediate node in the second graph are combined into a single node representing the fully overlapping character pattern.

[0020] Embodiments of the present invention may comprise or utilize a special purpose or general-purpose computer including computer hardware, such as, for example, one or more processors and system memory, as discussed in greater detail below. Embodiments within the scope of the present invention also include physical and other computer-readable media for carrying or storing computer-executable instructions and/or data structures. Such computer-readable media can be any available media that can be accessed by a general purpose or special purpose computer system. Computer-readable media that store computer-executable instructions are computer storage media (devices). Computer-readable media that carry computer-executable instructions are transmission media. Thus, by way of example, and not limitation, embodiments of the invention can comprise at least two distinctly different kinds of computer-readable media: computer storage media (devices) and transmission media.

[0021] Computer storage media (devices) includes RAM, ROM, EEPROM, CD-ROM or other optical disk storage, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store desired program code means in the form of computer-executable instructions or data structures and which can be accessed by a general purpose or special purpose computer.

[0022] A "network" is defined as one or more data links that enable the transport of electronic data between computer systems and/or modules and/or other electronic devices. When information is transferred or provided over a network or another communications connection (either hardwired, wireless, or a combination of hardwired or wireless) to a computer, the computer properly views the connection as a transmission medium. Transmissions media can include a network and/or data links which can be used to carry or desired program code means in the form of computer-executable instructions or data structures and which can be accessed by a general purpose or special purpose computer. Combinations of the above should also be included within the scope of computer-readable media.

[0023] Further, upon reaching various computer system components, program code means in the form of computer-executable instructions or data structures can be transferred automatically from transmission media to computer storage media (devices) (or vice versa). For example, computer-executable instructions or data structures received over a network or data link can be buffered in RAM within a network interface module (e.g., a "NIC"), and then eventually transferred to computer system RAM and/or to less volatile computer storage media (devices) at a computer system. Thus, it should be understood that computer storage media (devices) can be included in computer system components that also (or even primarily) utilize transmission media.

[0024] Computer-executable instructions comprise, for example, instructions and data which, when executed at a processor, cause a general purpose computer, special purpose computer, or special purpose processing device to perform a certain function or group of functions. The computer executable instructions may be, for example, binaries, intermediate format instructions such as assembly language, or even source code. Although the subject matter has been described in language specific to structural features and/or methodological acts, it is to be understood that the subject matter defined in the appended claims is not necessarily limited to the described features or acts described above. Rather, the described features and acts are disclosed as example forms of implementing the claims.

[0025] Those skilled in the art will appreciate that the invention may be practiced in network computing environments with many types of computer system configurations, including, personal computers, desktop computers, laptop computers, message processors, hand-held devices, multi-processor systems, microprocessor-based or programmable consumer electronics, network PCs, minicomputers, mainframe computers, mobile telephones, PDAs, pagers, routers, switches, and the like. The invention may also be practiced in distributed system environments where local and remote computer systems, which are linked (either by hardwired data links, wireless data links, or by a combination of hardwired and wireless data links) through a network, both perform tasks. In a distributed system environment, program modules may be located in both local and remote memory storage devices.

[0026] Within this description and the following claims a "regular expression", is a construct used to match strings of text, such as, for example, particular characters, words, or patterns of characters. In some embodiments, a regular expression has a limited alphabet. A regular expression can be written in a formal language that can be interpreted by a regular expression processor. The regular expression processor serves as a parser generator or examines text and identifies parts of the text that match a provided regular expression.

[0027] In generally, graphs can be used to represent regular expressions and their matching states. For example, turning briefly to Figure 2, graph 201 represents the regular expression "(\d\d)|(a(b|c))". Similarly, turning briefly to Figure 4, graph 401 represents the regular expression "([a,b,c]x)|(\d(cd|[1,3,5]([a,c,d]|ea)))". A graph can "run" by executing a state machine with input text, which allows parallelization of graphs.

[0028] Figure 1 illustrates an example computer architecture 100 that facilitates decomposing and merging regular expressions. Referring to Figure 1, computer architecture 100 includes decomposition module 101, labeling module 102, and merge module 141. Each of the depicted components can be connected to one another over (or is part of) a network, such as, for example, a Local Area Network ("LAN"), a Wide Area Network ("WAN"), and even the Internet. Accordingly, each of the depicted comments as well as any other connected computer systems and their components, can create message related data and exchange message related data (e.g., Internet Protocol ("IP") datagrams and other higher layer protocols that utilize IP datagrams, such as, Transmission Control

Protocol ("TCP"), Hypertext Transfer Protocol ("HTTP"), Simple Mail Transfer Protocol ("SMTP"), etc.) over the network.

[0029] In general, decomposition can be used to produce a set of simple graphs that represent a regular expression from a more complex graph that represents the regular expression. Accordingly, decomposition module 101 is configured to decompose a graph, such as, for example, a graph representing a regular expression, into a corresponding plurality of keyword graphs. Decomposition module 101 can essentially remove disjunctive portions of a more complex regular expression to break the more complex regular expression into a plurality of simpler regular expressions. A leaf node of each keyword graph represents a terminal condition from the more complex graph (which may be at an intermediate node or leaf node in the more complex graph). Decomposition module 101 can decompose labeled or unlabeled graphs.

[0030] Labeling module 102 is configured to label nodes of a graph or keyword graph to indicate matching states for a represented regular expression. Labeling module 102 can label nodes before or after decomposition.

[0031] Turning again to Figure 2, Figure 2 illustrates an example of decomposing a graph that represents a regular expression. As depicted, decomposition module 101 receives graph 201 as input. Graph 201 was previously labeled (represented by the diagonal hatching) to indicate matching states for the regular expression "(\d\d)|(a(b|c))". Decomposition module 101 decomposes graph 201 and outputs keyword graphs 202. The labels in graph 201 are carried over to keyword graphs 202. Thus, when text is compared to (run against) graph 201 or any of keyword graphs 202 any match is indicated as a match to "(\d\d)|(a(b|c))".

[0032] Turning again to Figure 4, Figure 4 illustrates another example of decomposing a graph that represents a regular expression. As depicted, decomposition module 101 receives graph 401 as input. Graph 401 was previously labeled (represented by the diagonal hatching) to indicate matching states for the regular expression "([a,b,c]x)|(\d(cd|[1,3,5]([a,c,d]|ea)))". Decomposition module 101 decomposes graph 401 and outputs keyword graphs 402. The labels in graph 401 are carried over to keyword graphs 402. Thus, when text is compared to (run against) graph 401 or any of keyword graphs 402 any match is indicated as a match to "([a,b,c]x)|(\d(cd|[1,3,5]([a,c,d]|ea)))".

[0033] In some embodiments, a graph is decomposed into keyword graphs in accordance with the following algorithm:

Start at the root node.

Identify all the children nodes of the root node.

For each of these nodes:

a. Copy the parent nodes above the node (call this "prefix.i").

b. Add this node and its subtree as a child to "prefix.i".

c. Start again from (2) except use the current node as the root node.

[0034] The algorithm can produce a collection of keyword graphs (e.g., DAGs) representing the graph. Each keyword graph has a single terminal node that is a leaf node. Within each graph, each node has a single child node.

[0035] In general, merging can be used to produce a single Directed Acyclic Graph ("DAG") representing a collection of regular expressions. Accordingly, merge module 101 is configured to receive two graphs as input and merge the two graphs into a single DAG that collectively represents matching states for the two input graphs. To eliminate processing redundancies, merge module 101 can combine overlapping character patterns at similarly positioned nodes in the two input graphs into a single node in the single DAG. When character patterns partially overlap, merge module 101 can alter the character pattern at a node in one input graph. Merge module 101 can then compensate by adding an additional edge between the node and a corresponding node in the other input graph. Adding an additional edge facilitates equivalence in matching states between the two input graphs and the single DAG.

[0036] In some embodiments, merge module 141 merges two keyword graphs into a single DAG. In other embodiments, merge module 141 merges a keyword graph and another graph into a single DAG. The functionality of merge module 141 can be reused as needed to merge larger sets of graphs together.

[0037] Turning to Figure 3, merge module 141 merges keyword graphs 301 (e.g., previously decomposed from another graph) and graph 302 into directed acyclic graph 304. Merge module 141 utilizes graph 302 and keyword graph 301A as input. Merge module 141 merges graph 302 and keyword graph 301A into intermediate graph 303. Subsequently, merge module 141 utilizes intermediate graph 330 and keyword graph 301B. Merge module 141 merges intermediate graph 303 and keyword graph 301B into directed acyclic graph 304. Since the character patterns nodes 312 and 313 overlap, nodes 312 and 313 are merged into a single node 314 in directed acyclic graph 304.

[0038] Labels (as indicated by different diagonal hatching) are maintained throughout the merge process. Thus, terminal nodes indicate a regular expression that is matched. Nodes 316 and 317 indicate a match to the regular expression "\d\d|um" (the regular expression they were decomposed from) and node 318 indicates a match to the regular expression "un".

[0039] As depicted in Figure 3, inputs to merge module 141 are external. In other embodiments, merge module 141 receives a set of graphs as input and outputs a DAG. During processing, intermediate graphs are maintained and processed internally within merge module 141.

[0040] As depicted, merge module 141 includes position detector 142, overlap detector 143, and overlap compensator 144. During merging position, position detector 142 is configured to identify similarly positioned nodes within different graphs. Similarly positioned nodes can be identified based on a distance from the root node. For example, in Figure 3, nodes 312 and 313 are similarly positioned. During merging, overlap detector 143 is configured to detect if character patterns of different nodes at least partially overlap. For example, the character pattern [1, 3, 5] partially matches the character pattern \d. On the other hand, the character pattern [a, b, c] and the character pattern [a, b, c] fully overlap. During merging, overlap compensator 144 is configured to compensate when nodes with partially overlapping character patterns are merged into a single node. Compensation can include adding edges between input graphs that are being merged. The additional edges facilitate equivalence between matching states of the input graphs and matching states of resulting DAG.

[0041] Figure 5 illustrates another example of merging graphs that represent different regular expressions. Keyword graph 501 and graph 502 can be received as input (e.g., at merge module 141). Position detector 142 can detect that node 511 and node 512 are similarly positioned within keyword graph 501 and graph 502 respectively. Overlap detector 143 can identify partially overlapping patterns 503 (or common edges). That is, character pattern \d partially overlaps the character pattern [2, 3]. Overlap compensator 144 can remove the partial overlap (remove common edges) by altering the character pattern of node 511 to "\d-[2,3]". Overlap compensator can also add edge 514 from node 512 to node 513. Merge module 114 can then combine root nodes to add (altered) keyword graph 501 to graph 502. Overlap compensation allows graphs to be merged yet still represent equivalent matching states. For example, the text string "2cd" still matches keyword graph 501 even though a comparison is made at node 512 (and node 511 is bypassed).

[0042] As depicted, the different hatching within terminal nodes indicates matching states for keyword graph 501 and graph 502 respectively.

[0043] In some embodiments, graphs are merge in accordance with the following algorithm:

```
Create an empty DAG with just a Root node. Label this as the Final.DAG.
For each DAG (i.DAG) in the collection, do the following:
    a. Set i.node as the root node of i.DAG.
    b. Set final.node as the root node of Final.DAG.
    c. Iterate through i.node and final.node as long as final.node has the exact
       same edge.
    d. If the i.node edge is a superset of the final.node edge:
        i. Add an edge representing the non-common characters between
           i.node and final.node. This edge points to the child of i.node.
        ii. For each common (edge, node)
            1. Iterate along final.node and i.node as long as they have the
               exact same edge.
            2. If the terminal node was reached, label it as terminal for
               i.DAG.
            3. If not, add an edge from the final.node to the i.node's child.
    e. If the final.node edge is a superset of the i.node edge:
        i. Add an edge representing the non-common characters between
           i.node and final.node. This edge points to the child of final.node.
        ii. For each common (edge, node)
            1. Iterate along final.node and i.node as long as they have the
               exact same edge.
            2. If the terminal node was reached, label it as terminal for
               final.DAG.
            3. If not, add an edge from the i.node to the final.node's
               children.
```

[0044] Figure 6 illustrates a flow chart of an example method 600 for decomposing and merging regular expressions. Method 600 will be described with respect to the components and data of computer architecture 100 and some reference to Figures 3 and 5.

[0045] Method 600 includes an act of accessing a graph representing a first regular expression (act 601). For example, decomposition module 101 can access graph 112, representing regular expression 111. Method 600 includes an act of decomposing the graph into one or more keyword graphs, each of one or more keyword graphs having a root node, one or more intermediate nodes, and a leaf node, each of the one or more intermediate nodes and the leaf node indentifying a character pattern that partially matches the first regular expression, the root node and each of the one or more intermediate nodes having a single child node, one of the intermediate nodes having the leaf node as a child node (act 602). For example, decomposition module 101 can decompose graph 112 into keyword graphs 113 (e.g., 113A, 113B, 113C, etc.).

[0046] Method 600 includes an act of labeling the leaf node of each of the one or more keyword graphs as a matching state for the first regular expression (act 603). For example, labeling module 102 can label the leaf nodes of keyword graphs 113 to generate labeled keyword graphs 113AL, 113BL, 113BL, etc.

[0047] Method 600 includes an act of accessing a second graph representing a second regular expression, the second graph having a root node, one or more intermediate nodes, and one or more leaf nodes, each of the one or more intermediate nodes and the one or more leaf nodes indentifying a character pattern that partially matches the second regular expression (act 604). For example, labeling module 102 can access graph 123, representing regular expression 121. Method 600 includes an act of labeling one or more terminal nodes in the second graph as a matching state for the second regular expression (act 605). For example, labeling module 102 can label the terminal nodes of graph 123 to generate labeled graph 123L

[0048] Method 600 includes an act of merging the one or more keyword graphs and the second graph into a directed acyclic graph that collectively represents both the first regular expression and the second regular expression (act 606). For example, merge module 141 can mere labeled keyword graphs 113L and labeled graph 123L into directed acyclic graph 134. Directed acyclic graph 134 collectively represents regular expression 111 and regular expression 121.

[0049] Act 606 includes an act of an act of identifying any similarly positioned intermediate nodes within the one or more keyword graphs and the second graph that have at least partially overlapping character patterns (act 607). For example, position detector 142 can identify similarly positioned intermediate nodes in one more labeled keyword graphs 113L and labeled graph 123L. Similarly positioned nodes can be nodes that are equidistance from their root node. For example, referring to Figure 3 nodes 312 and 313 are similarly positioned (both are one edge from their corresponding root node). Similarly, in Figure 5, nodes 511 and 512 are similarly position. Nodes 513 and 514 are also similarly positioned in Figure 5.

[0050] Among similarly positioned intermediate nodes, overlap detector 143 can detect when nodes have at least partially overlapping character patterns. In Figure 3, nodes 312 and 313 fully overlap. In Figure 5, nodes 511 and 512 partially overlap and nodes 513 and 514 do not overlap.

[0051] For any identified intermediate nodes in a keyword graph and indentified intermediate nodes in the second graph that are similarly positioned and have partially overlapping character patterns, act 606 includes an act of altering the character pattern of at least one of the indentified intermediate nodes to eliminate the partially overlapping character pattern (act 608). For example, overlap compensator 144 can alter a character pattern at an intermediate node to eliminate a partial overlap with another node. Referring to Figure 5, character pattern "\d" at node 511 can be altered to "\d-[2,3]" (which is equivalent to [0, 1, 4, 5, 6, 7, 8, 9]) to eliminate the partial overlap with node 512.

[0052] For any identified intermediate nodes in a keyword graph and indentified intermediate nodes in the second graph that are similarly positioned and have partially overlapping character patterns, act 606 includes an act of adding an edge between the keyword graph and the second graph to compensate for altering the character pattern of the at least one of the identified intermediate nodes (act 609). For example, overlap compensator 144 can add an edge from a non-altered node to a node below the altered node to compensate for altering the character pattern of the altered node. Referring to Figure 5, edge 514 can be added from node 512 to node 513 to compensate for altering the character pattern of node 511.

[0053] For any identified intermediate nodes in a keyword graph and indentified intermediate nodes in the second graph that are similarly positioned and have fully overlapping character patterns, act 606 includes an act of combining together the keyword graph and the second graph by combining the intermediate node in the keyword graph and the intermediate node in the second graph into a single node representing the fully overlapping character pattern (act 610). For example, overlap compensator 144 can combine an intermediate node of a labeled keyword graph 113L and an intermediate node of labeled graph 123L. Referring to Figure 3, node 312 and node 313 can be combined into node 314.

[0054] Subsequent to creation of a DAG, the DAG can be run on a state machine against a portion of text to determine if the portion of text matches any regular expressions represented in the DAG.

[0055] In some embodiments, merging graphs is combined with other passes over regular expressions to facilitate expanding regular expression syntax (e.g. *, +, or number sets). For example, when constructing a DAG to represent regular expressions, it is possible that the entirety of a regular expression cannot be represented by DAG. For example, a regular expression can be include characters such as, ?: or nested * operators.

[0056] An increasingly complex state machine can be constructed to handle these types of operators. Another alternative is to create multiple "text processors" that include actual regular expressions and a monolithic DAG. The following algorithm can then be used to merge regular expressions:

Decompose regular expressions into its components that can be represented as a complex DAG and that which cannot.

a. Consider: 123\d\d\d(5.*3)*\d\d\d\d

b. This can produce the following components:

i. DAG: 123\d\d\d | \d\d\d\d

ii. Regular Expression: (5.*3)*

Run all the "text processors" for regular expressions and the single DAG

Collect the locations in the text at which these text processors were found (already sorted, as guaranteed by the DAG/Regex).

Reassemble the original regular expression based on the results of the DAG and its regular expressions to determine if it was found.

If the results from step (3) are stored in a collection of heaps (e.g. Fibonacci heap), this step is bound by O(n).

[0057] As such, a generated DAG can be used with a regular expression engine to produce results for an entire regular expression alphabet. A multi-pass approach also allows for the execution of look-ahead or look-behind regular expressions without in place backtracking or forward tracking, which simplifies the complexity of the system and helps performance.

[0058] Accordingly, embodiments of the invention decompose a regular expression into multiple simple keyword graphs, merge those keyword graphs in a compact and efficient manner, and produce a directed acyclic graph (DAG) that can execute a simplified regular expression alphabet. Several of these regular expression DAG's can then be merged together to produce a single DAG that represents an entire collection of regular expressions. DAGs along with other text processing algorithms and a heap collection can be combined in a multi-pass approach to expand the regular expression alphabet.

[0059] The present invention may be embodied in other specific forms without departing from its spirit or essential characteristics. The described embodiments are to be considered in all respects only as illustrative and not restrictive. The scope of the invention is, therefore, indicated by the appended claims rather than by the foregoing description. All changes which come within the meaning and range of equivalency of the claims are to be embraced within their scope.

CLAIMS

What is claimed:

1. At a computer system including one or more processors and system memory, a method for representing one or more regular expressions in a directed acyclic graph, the method comprising:

an act of accessing one or more keyword graphs decomposed from a first regular expression, each of one or more keyword graphs having a root node, one or more intermediate nodes, and a leaf node, each of the one or more intermediate nodes and the leaf node indentifying a character pattern that partially matches the first regular expression, the root node and each of the one or more intermediate nodes having a single child node, one of the intermediate nodes having the leaf node as a child node, each leaf node labeled as a matching state for the first regular expression;

an act of accessing a second graph representing at least a part of a second regular expression, the second graph having a root node, one or more intermediate nodes, and one or more leaf nodes, each of the one or more intermediate nodes and the one or more leaf nodes indentifying a character pattern that partially matches the second regular expression;

an act of merging the one or more keyword graphs and the second graph into a directed acyclic graph that collectively represents both the first regular expression and the second regular expression, including for each of the one or more keyword graphs:

an act of selecting the keyword graph individually;

an act of identifying any similarly positioned intermediate nodes within the selected keyword graph and the second graph that have at least partially overlapping character patterns; and

for any identified intermediate nodes in the selected keyword graph and indentified intermediate nodes in the second graph that are similarly positioned and have partially overlapping character patterns, an act of merging the selected keyword graph and the second graph at the indentified intermediate nodes to represent equivalent matching states from the selected keyword graph and the second graph in the directed acyclic graph, the merging causing the keyword graph to become part of the second graph.

2. The method as recited in claim 1, wherein the act of identifying any similarly positioned intermediate nodes within the selected keyword graph and the second graph that have at least partially overlapping character patterns comprises an act of identifying an intermediate node in the selected keyword graph and an intermediate node in the second graph that fully overlap.

3. The method as recited in claim 2, wherein the act of merging the selected keyword graph and the second graph at the indentified intermediate nodes comprises an act of combining the intermediate node in the selected keyword graph and the intermediate node in the second graph into a single node representing the fully overlapping character pattern.

4. The method as recited in claim 1, wherein the act of identifying any similarly positioned intermediate nodes within the selected keyword graph and the second graph that have at least partially overlapping character patterns comprises an act of identifying an intermediate node in the selected keyword graph and an intermediate node in the second graph that partially overlap.

5. The method as recited in claim 4, wherein the act of merging the selected keyword graph and the second graph at the indentified intermediate nodes comprises an act of altering the character pattern of at least one of the indentified intermediate nodes to eliminate the partially overlapping character pattern

6. The method as recited in claim 4, wherein the act of merging the selected keyword graph and the second graph at the indentified intermediate nodes comprises an act of adding an edge between the selected keyword graph and the second graph to compensate for altering the character pattern of the at least one of the identified intermediate nodes.

7. A computer program product for use at computer system, the computer program product for implementing a method for representing one or more regular expressions in a directed acyclic graph, the computer program product comprising one or more computer storage devices, having stored thereon computer-executable instructions that, when executed at a processor, cause the computer system to perform the method, including the following:

access one or more keyword graphs decomposed from a first regular expression, each of one or more keyword graphs having a root node, one or more intermediate nodes, and a leaf node, each of the one or more intermediate nodes and the leaf node indentifying a character pattern that partially matches the first regular expression, the root node and each of the one or more intermediate nodes having a single child node, one of the intermediate nodes having the leaf node as a child node, each leaf node labeled as a matching state for the first regular expression;

access a second graph representing a second regular expression, the second graph having a root node, one or more intermediate nodes, and one or more leaf nodes, each of the one or more intermediate nodes and the one or more leaf nodes indentifying a character pattern that partially matches the second regular expression;

merge the one or more keyword graphs and the second graph into a directed acyclic graph that collectively represents both the first regular expression and the second regular expression, including:

identify any similarly positioned intermediate nodes within the one or more keyword graphs and the second graph that have at least partially overlapping character patterns; and

for any identified intermediate nodes in a keyword graph and indentified intermediate nodes in the second graph that are similarly positioned and have partially overlapping character patterns, merge the keyword graph and the second graph at the indentified intermediate nodes to represent equivalent matching states represented in the keyword graph and in the second graph in the directed acyclic graph.

8. The computer program product as recited in claim 7, further comprising computer-executable instructions that, when executed, cause the computer system to label the leaf node of each of the one or more keyword graphs as a matching state for the first regular expression.

9. The computer program product as recited in claim 7, further comprising computer-executable instructions that, when executed, cause the computer system to label each terminal node in the second graph as a matching state for the second regular expression.

10. At a computer system including one or more processors and system memory, a method for representing one or more regular expressions in a directed acyclic graph, the method comprising:

an act of accessing one or more keyword graphs decomposed from a first regular expression, each of one or more keyword graphs having a root node, one or more intermediate nodes, and a leaf node, each of the one or more intermediate nodes and the leaf node indentifying a character pattern that partially matches the first regular expression, the root node and each of the one or more intermediate nodes having a single child node, one of the intermediate nodes having the leaf node as a child node, each leaf node labeled as a matching state for the first regular expression;

an act of accessing a second graph representing a second regular expression, the second graph having a root node, one or more intermediate nodes, and one or more leaf nodes, each of the one or more intermediate nodes and the one or more leaf nodes indentifying a character pattern that partially matches the second regular expression, the second graph having one or more terminal nodes labeled as a matching state for the second regular expression; and

an act of merging the one or more keyword graphs and the second graph into a directed acyclic graph that collectively represents both the first regular expression and the second regular expression, including:

an act of identifying any similarly positioned intermediate nodes within the one or more keyword graphs and the second graph that have at least partially overlapping character patterns;

for any identified intermediate nodes in a keyword graph and indentified intermediate nodes in the second graph that are similarly positioned and have partially overlapping character patterns:

an act of altering the character pattern of at least one of the indentified intermediate nodes to eliminate the partially overlapping character pattern; and

an act of adding an edge between the keyword graph and the second graph to compensate for altering the character pattern of the at least one of the identified intermediate nodes;

for any identified intermediate nodes in a keyword graph and indentified intermediate nodes in the second graph that are similarly positioned and have fully overlapping character patterns:

an act of combining together the keyword graph and the second graph by combining the intermediate node in the keyword graph and the intermediate node in the second graph into a single node representing the fully overlapping character pattern.

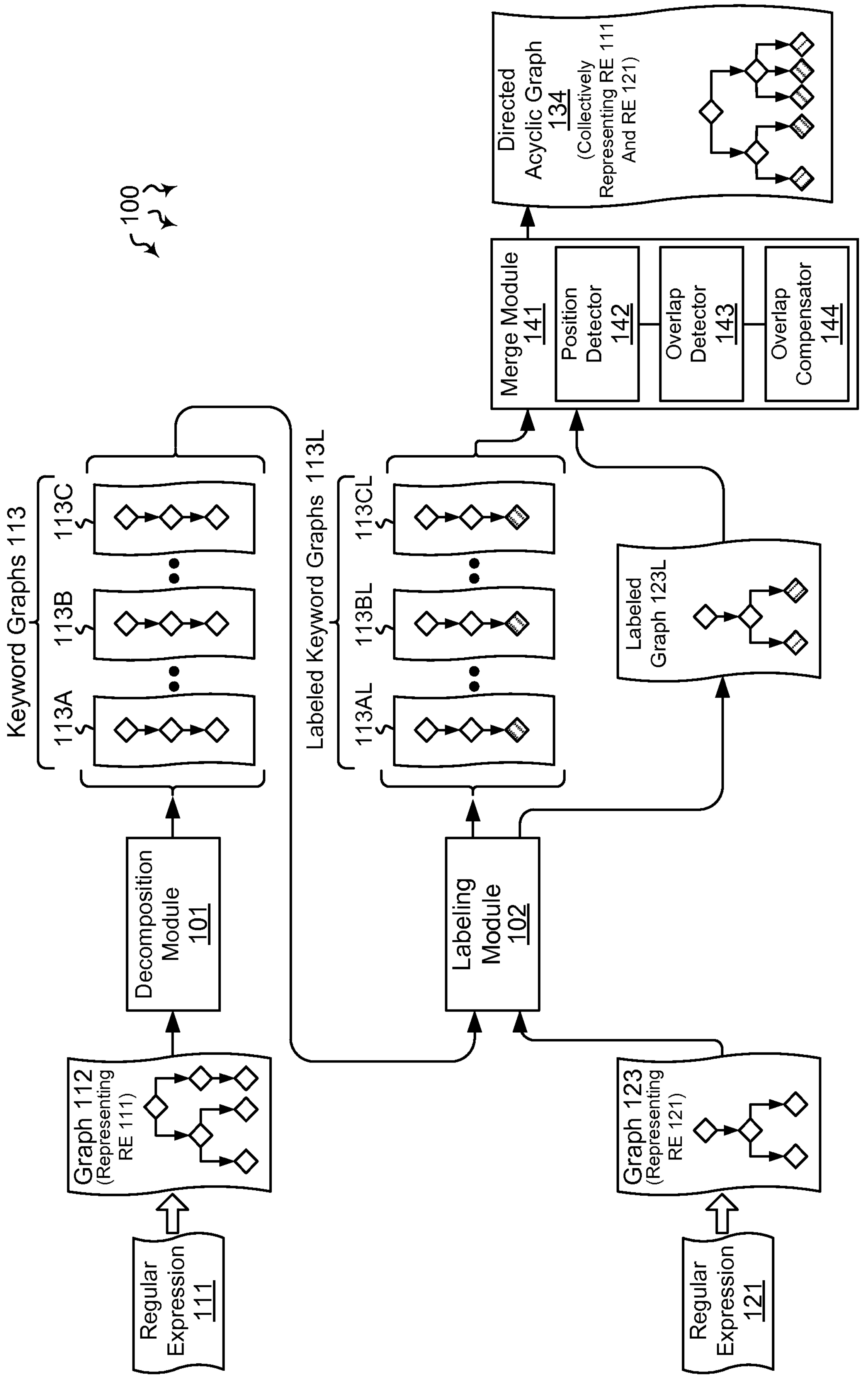

100
Regular Expression 111
Graph 112 (Representing RE 111)
Decomposition Module 101
Keyword Graphs 113
113A
113B
113C
Regular Expression 121
Graph 123 (Representing RE 121)
Labeling Module 102
Labeled Keyword Graphs 113L
113AL
113BL
113CL
Labeled Graph 123L
Merge Module 141
Position Detector 142
Overlap Detector 143
Overlap Compensator 144
Directed Acyclic Graph 134 (Collectively Representing RE 111 And RE 121)

Figure 1

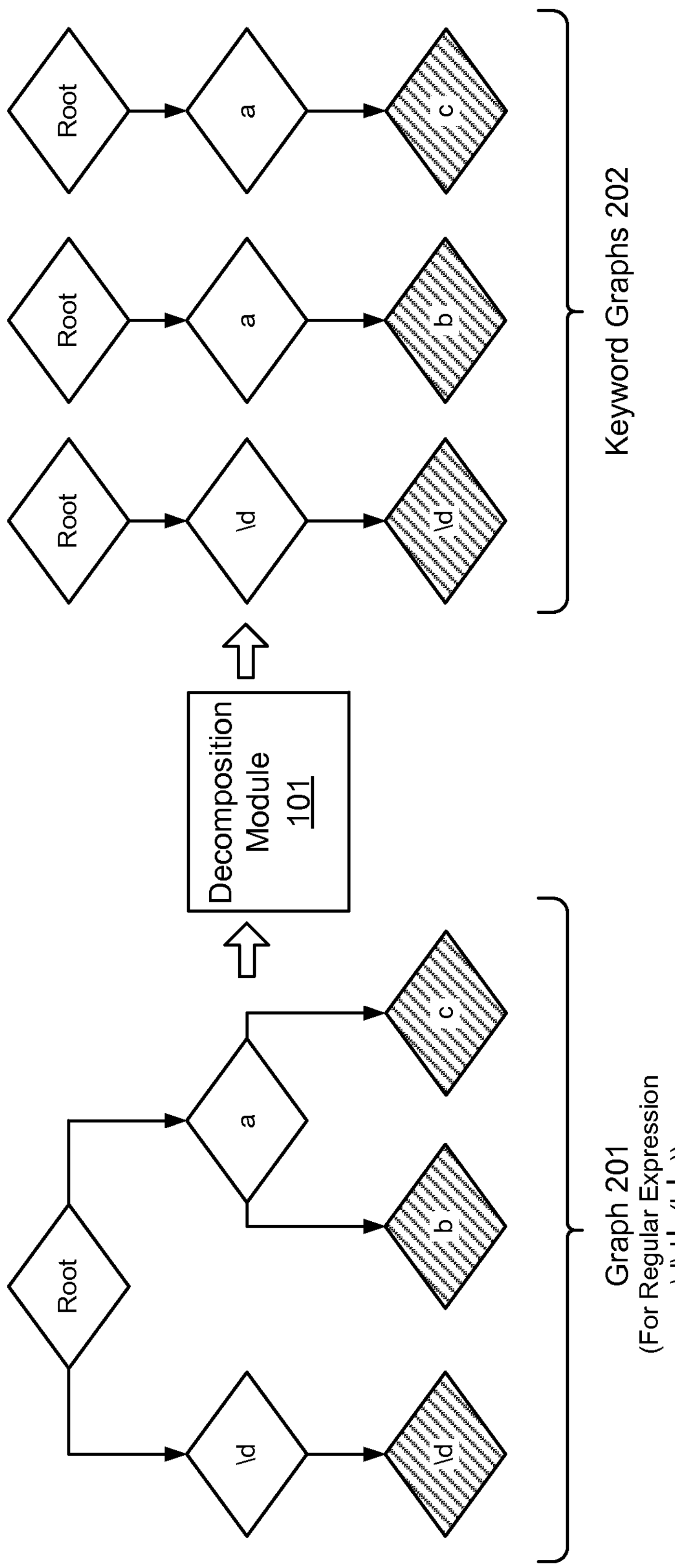

Root
\d
\d
a
b
c
Graph 201
(For Regular Expression
\d\d|a(b|c))
Decomposition
Module
101
Root
\d
\d
Root
a
b
Root
a
c
Keyword Graphs 202

Figure 2

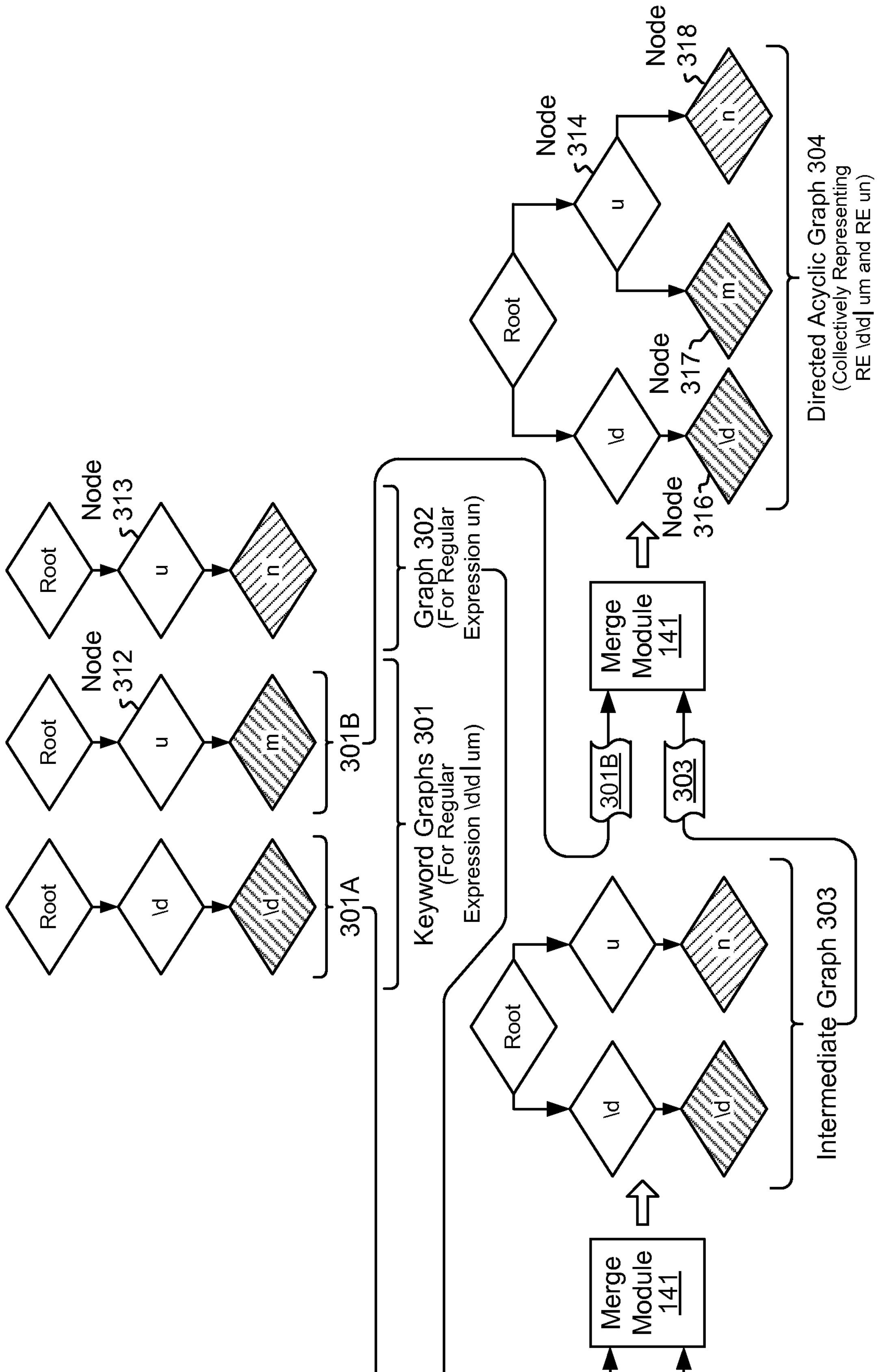

Root
\d
\d
301A
Root
u
m
Node 312
301B
Root
u
n
Node 313
Keyword Graphs 301 (For Regular Expression \d\d | um)
Graph 302 (For Regular Expression un)
302
301A
Merge Module 141
Root
\d
\d
u
n
Intermediate Graph 303
301B
303
Merge Module 141
Root
\d
\d
Node 316
m
Node 317
u
Node 314
n
Node 318
Directed Acyclic Graph 304 (Collectively Representing RE \d\d| um and RE un)

Figure 3

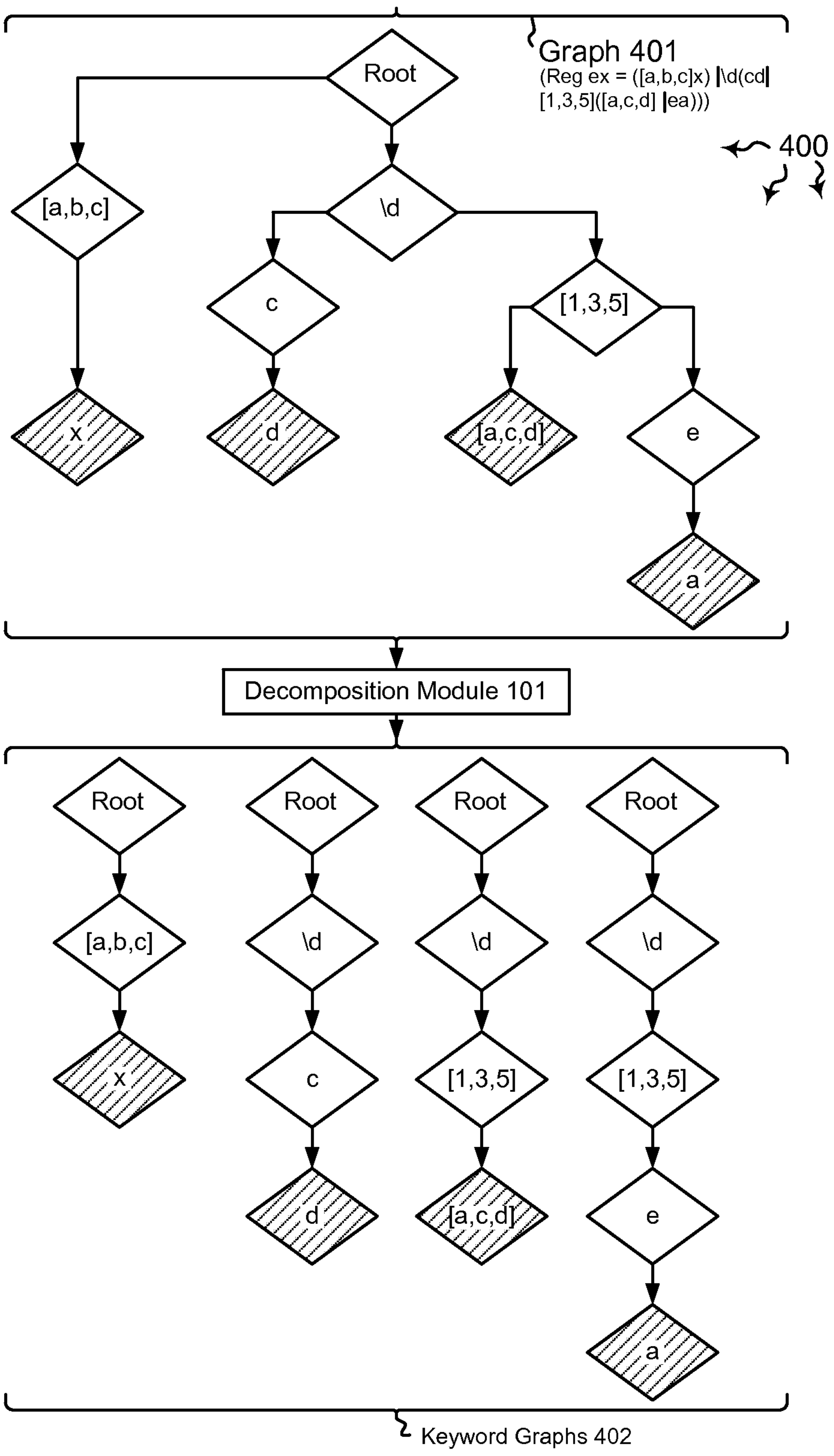
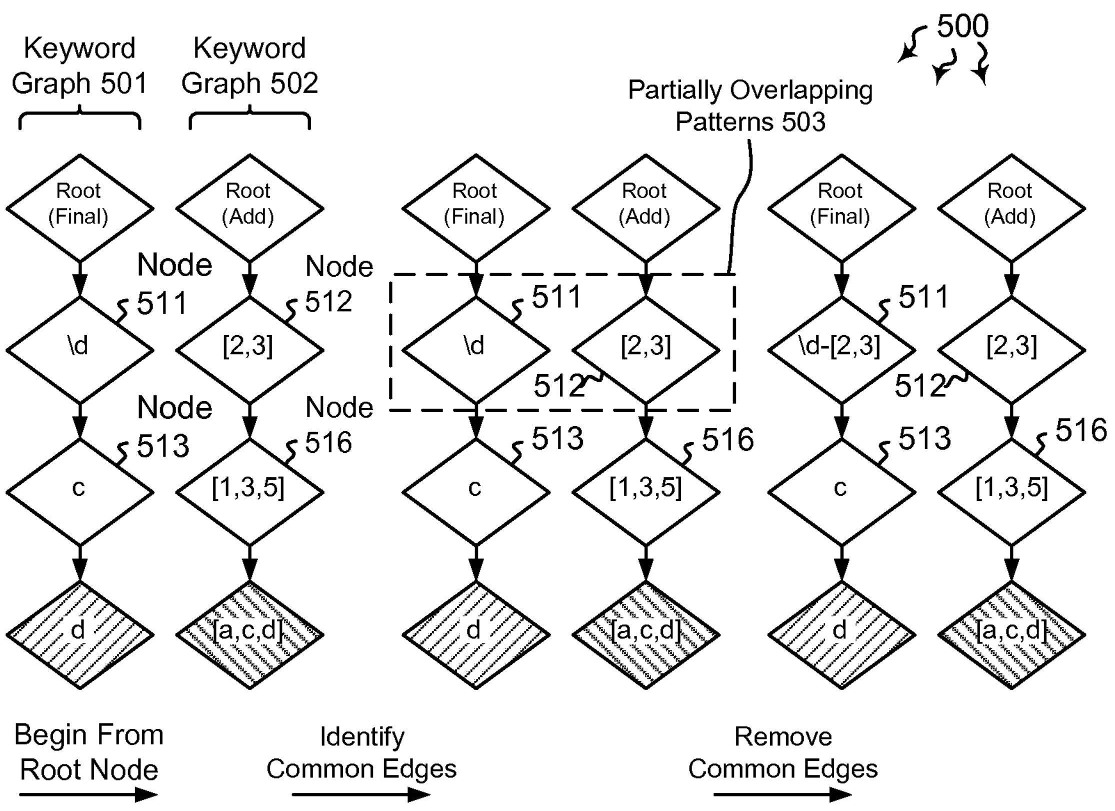

Graph 401
(Reg ex = ([a,b,c]x) |\d(cd| [1,3,5]([a,c,d] |ea)))
400
Root
[a,b,c]
\d
c
[1,3,5]
x
d
[a,c,d]
e
a
Decomposition Module 101
Root
[a,b,c]
x
Root
\d
c
d
Root
\d
[1,3,5]
[a,c,d]
Root
\d
[1,3,5]
e
a
Keyword Graphs 402

Figure 4


500
Keyword Graph 501
Keyword Graph 502
Partially Overlapping Patterns 503
Root (Final)
Root (Add)
Node 511
Node 512
Node 513
Node 516
\d
[2,3]
c
[1,3,5]
d
[a,c,d]
511
512
513
516
\d-[2,3]
Begin From Root Node
Identify Common Edges
Remove Common Edges

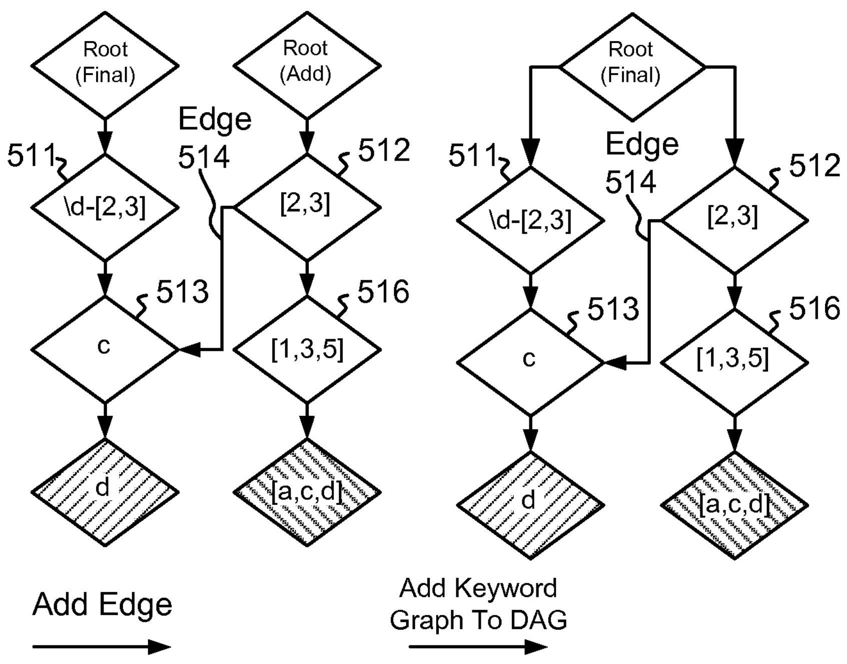

Root (Final)
Root (Add)
Edge 514
511
512
513
516
\d-[2,3]
[2,3]
c
[1,3,5]
d
[a,c,d]
Add Edge
Add Keyword Graph To DAG

Figure 5

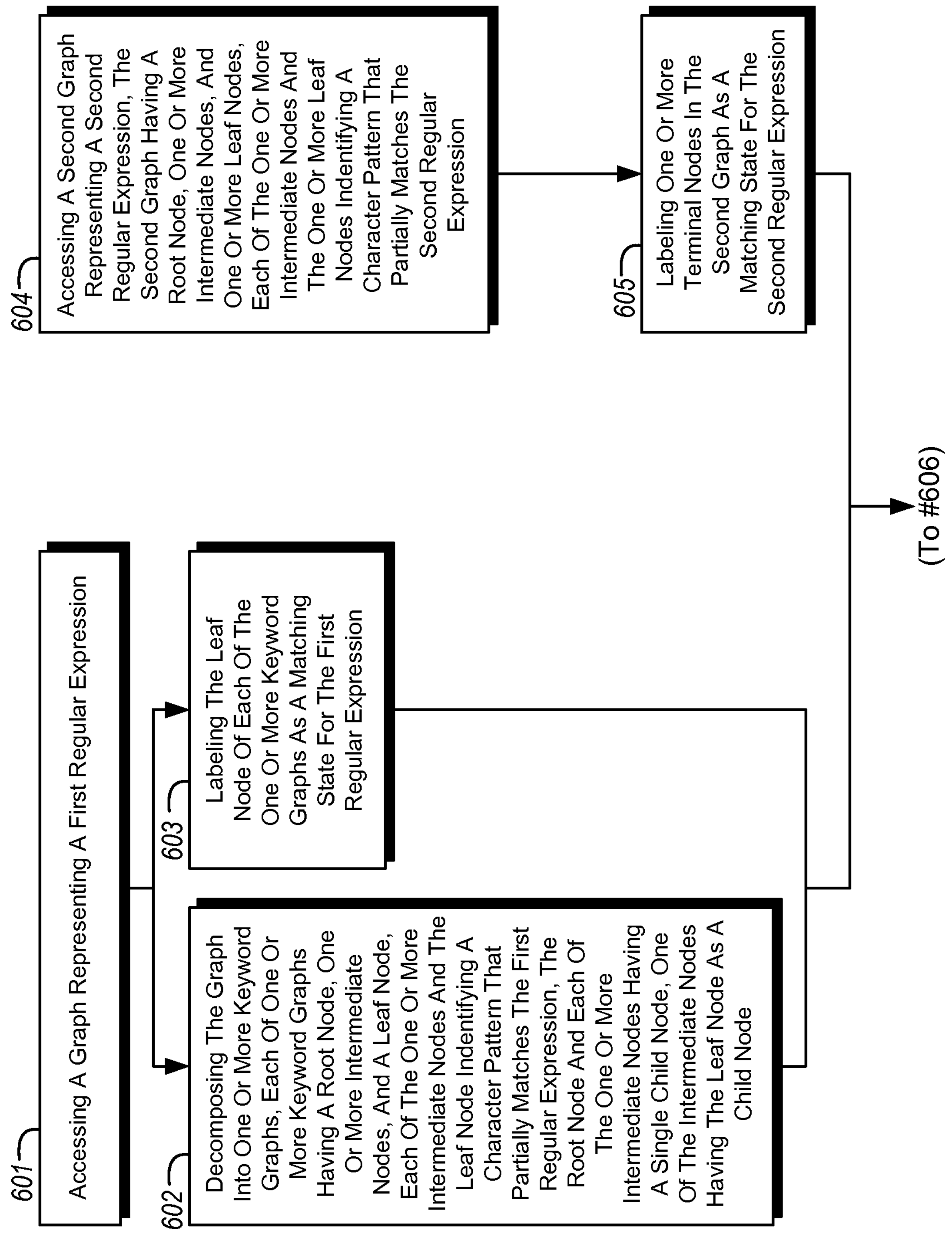
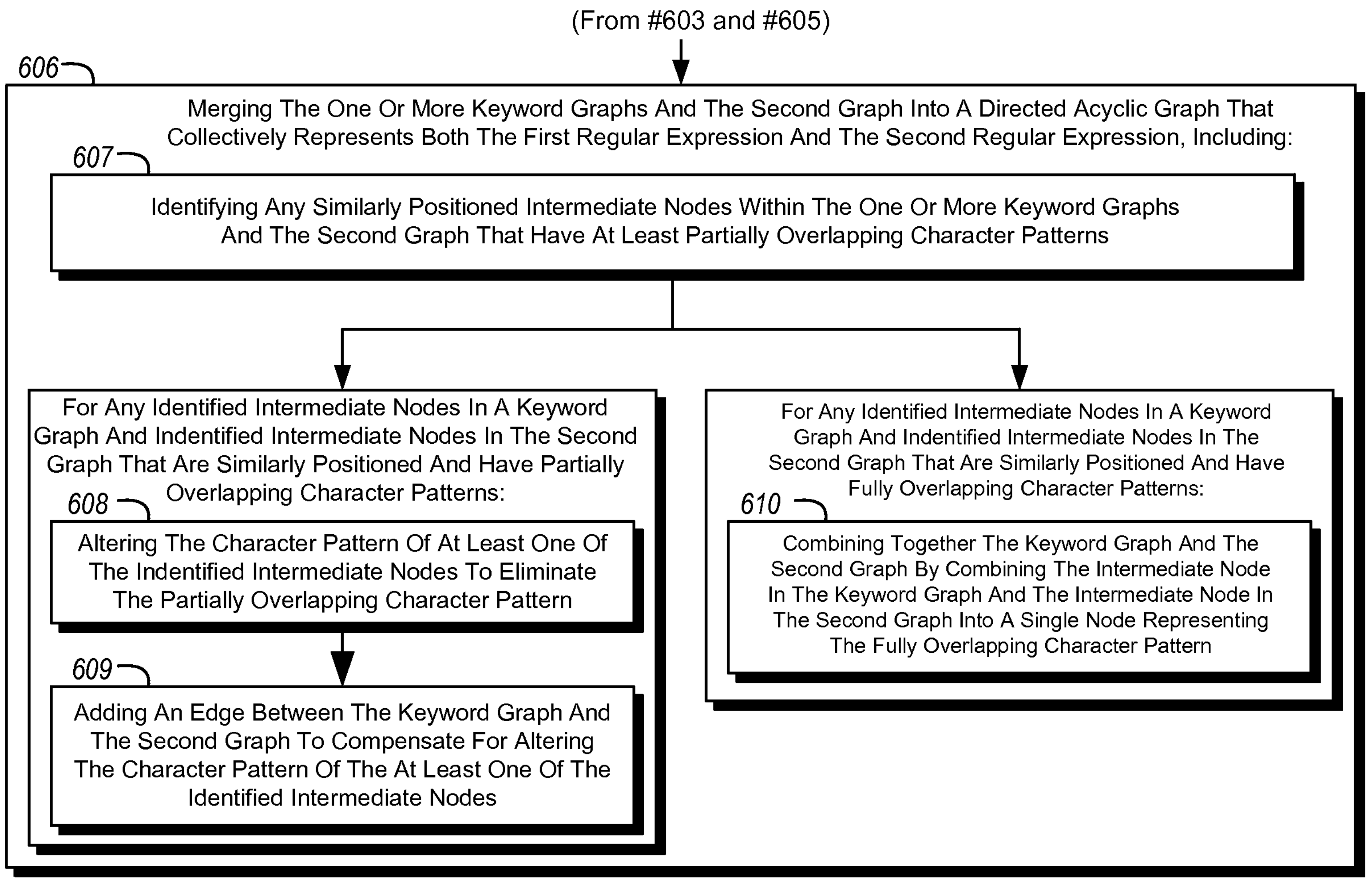

601
Accessing A Graph Representing A First Regular Expression
602
Decomposing The Graph Into One Or More Keyword Graphs, Each Of One Or More Keyword Graphs Having A Root Node, One Or More Intermediate Nodes, And A Leaf Node, Each Of The One Or More Intermediate Nodes And The Leaf Node Indentifying A Character Pattern That Partially Matches The First Regular Expression, The Root Node And Each Of The One Or More Intermediate Nodes Having A Single Child Node, One Of The Intermediate Nodes Having The Leaf Node As A Child Node
603
Labeling The Leaf Node Of Each Of The One Or More Keyword Graphs As A Matching State For The First Regular Expression
604
Accessing A Second Graph Representing A Second Regular Expression, The Second Graph Having A Root Node, One Or More Intermediate Nodes, And One Or More Leaf Nodes, Each Of The One Or More Intermediate Nodes And The One Or More Leaf Nodes Indentifying A Character Pattern That Partially Matches The Second Regular Expression
605
Labeling One Or More Terminal Nodes In The Second Graph As A Matching State For The Second Regular Expression
(To #606)

Figure 6



(From #603 and #605)
606
Merging The One Or More Keyword Graphs And The Second Graph Into A Directed Acyclic Graph That Collectively Represents Both The First Regular Expression And The Second Regular Expression, Including:
607
Identifying Any Similarly Positioned Intermediate Nodes Within The One Or More Keyword Graphs And The Second Graph That Have At Least Partially Overlapping Character Patterns
For Any Identified Intermediate Nodes In A Keyword Graph And Indentified Intermediate Nodes In The Second Graph That Are Similarly Positioned And Have Partially Overlapping Character Patterns:
608
Altering The Character Pattern Of At Least One Of The Indentified Intermediate Nodes To Eliminate The Partially Overlapping Character Pattern
609
Adding An Edge Between The Keyword Graph And The Second Graph To Compensate For Altering The Character Pattern Of The At Least One Of The Identified Intermediate Nodes
For Any Identified Intermediate Nodes In A Keyword Graph And Indentified Intermediate Nodes In The Second Graph That Are Similarly Positioned And Have Fully Overlapping Character Patterns:
610
Combining Together The Keyword Graph And The Second Graph By Combining The Intermediate Node In The Keyword Graph And The Intermediate Node In The Second Graph Into A Single Node Representing The Fully Overlapping Character Pattern

Figure 6 (Continued)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2011/062479

A. CLASSIFICATION OF SUBJECT MATTER

G06F 17/30(2006.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F 17/30; G06F 15/00; G06F 17/00; H04L 9/00; G06N 5/00; G06F 15/177; G06F 15/18

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
Korean utility models and applications for utility models
Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
eKOMPASS(KIPO internal) & Keywords: regular, expression, combine, tree

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2010-0114811 A1 (BRANIMIR LAMBOV) 06 May 2010 See abstract; paragraphs[0036-0042,0081-0084] and figures 1,7B.	1-10
A	US 2005-0278781 A1 (SHI-MING ZHAO et al.) 15 December 2005 See abstract; paragraphs[0030-0043,0052-0060] and figures 1,2A,7.	1-10
A	US 7689530 B1 (JOHN J. WILLIAMS, JR. et al.) 30 March 2010 See abstract; column 6,line 42 - column 19,line 39; figures 6A-6C.	1-10
A	US 2009-0276506 A1 (CHARU TIWARI et al.) 05 November 2009 See abstract; paragraphs[0021-0024,0063-0075] and figures 4A,4B.	1-10

☐ Further documents are listed in the continuation of Box C. ☒ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents,such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search 26 APRIL 2012 (26.04.2012)	Date of mailing of the international search report **04 MAY 2012 (04.05.2012)**
Name and mailing address of the ISA/KR Korean Intellectual Property Office Government Complex-Daejeon, 189 Cheongsa-ro, Seo-gu, Daejeon 302-701, Republic of Korea Facsimile No. 82-42-472-7140	Authorized officer LEE, Myung Jin Telephone No. 82-42-481-8474

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2011/062479

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2010-0114811 A1	06.05.2010	None	
US 2005-0278781 A1	15.12.2005	CN 100574321 C	23.12.2009
		CN 101098331 A	02.01.2008
		CN 101098331 B	09.06.2010
		CN 101098331 C0	02.01.2008
		CN 1716958 A	04.01.2006
		CN 1716958 C0	04.01.2006
		CN 1716959 A	04.01.2006
		CN 1716959 C0	04.01.2006
		CN 1838670 A0	27.09.2006
		EP 1607823 A2	21.12.2005
		EP 1607823 A3	25.01.2006
		EP 1701285 A1	13.09.2006
		EP 1744235 A1	17.01.2007
		EP 1865434 A1	12.12.2007
		TW I268057 B	01.12.2006
		TW I271056 B	11.01.2007
		TW I314705 A	11.09.2009
		TW 268057 A	01.12.2006
		TW 268057 B	01.12.2006
		TW 271056 A	11.01.2007
		TW 271056 B	11.01.2007
		US 2005-0278783 A1	15.12.2005
		US 2006-0005241 A1	05.01.2006
		US 2006-0206939 A1	14.09.2006
		US 2006-0224828 A1	05.10.2006
		US 7216364 B2	08.05.2007
		US 7596809 B2	29.09.2009
		US 7685637 B2	23.03.2010
		US 7779464 B2	17.08.2010
		US 7930742 B2	19.04.2011
US 7689530 B1	30.03.2010	US 7308446 B1	11.12.2007
US 2009-0276506 A1	05.11.2009	US 2010-0174715 A1	08.07.2010
		US 7668942 B2	23.02.2010