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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
24 March 2016 (24.03.2016)

WIPO | PCT

(10) International Publication Number
WO 2016/044321 A1

(51) International Patent Classification:
G06F 17/27 (2006.01)

(21) International Application Number:
PCT/US2015/050263

(22) International Filing Date:
15 September 2015 (15.09.2015)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
62/051,292 16 September 2014 (16.09.2014) US

(72) Inventor; and
(71) Applicant : **TANG, Min** [CN/US]; 12196 Ne 24th Street, Bellevue, WA 98005 (US).

(74) Agents: **KOO, Hean L.** et al.; Sheppard Mullin Richter & Hampton LLP, 2099 Pennsylvania Avenue, N.W., Suite 100, Washington, DC 20006 (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, BN, 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, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, 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, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, 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, KM, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) Title: INTEGRATION OF DOMAIN INFORMATION INTO STATE TRANSITIONS OF A FINITE STATE TRANSDUCER FOR NATURAL LANGUAGE PROCESSING

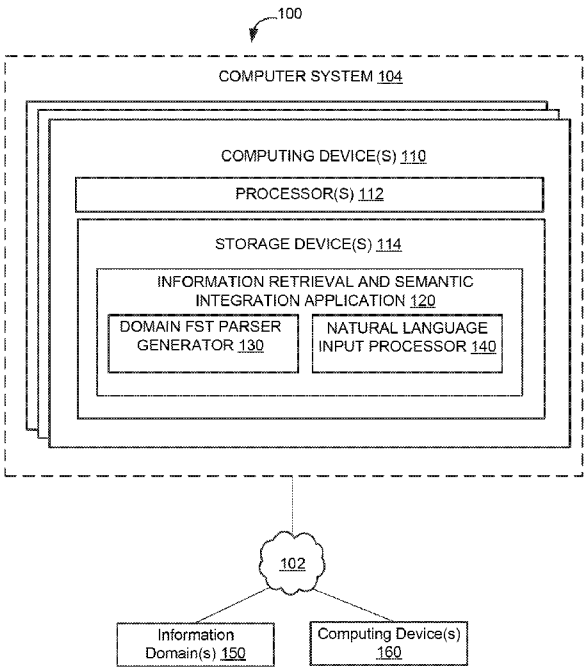

FIG. 1

(57) Abstract: The invention relates to a system and method for integrating domain information into state transitions of a Finite State Transducer ("FST") for natural language processing. A system may integrate semantic parsing and information retrieval from an information domain to generate an FST parser that represents the information domain. The FST parser may include a plurality of FST paths, at least one of which may be used to generate a meaning representation from a natural language input. As such, the system may perform domain-based semantic parsing of a natural language input, generating more robust meaning representations using domain information. The system may be applied to a wide range of natural language applications that use natural language input from a user such as, for example, natural language interfaces to computing systems, communication with robots in natural language, personalized digital assistants, question-answer query systems, and/or other natural language processing applications.

WO 2016/044321 A1

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))*

INTEGRATION OF DOMAIN INFORMATION INTO STATE TRANSITIONS OF A FINITE STATE TRANSDUCER FOR NATURAL LANGUAGE PROCESSING

CROSS REFERENCE TO RELATED APPLICATION

[001] This application claims the benefit of U.S. Provisional Patent Application Serial No. 62/051,292 filed September 16, 2014 entitled "INTEGRATION OF DOMAIN INFORMATION INTO STATE TRANSITIONS OF A FINITE STATE TRANSDUCER FOR NATURAL LANGUAGE PROCESSING", the entirety of which is incorporated herein by reference.

FIELD OF THE INVENTION

[002] The invention relates to systems and methods of integrating domain information into state transitions of a Finite State Transducer for natural language processing.

BACKGROUND OF THE INVENTION

[003] Semantic parsing is a process that transforms natural language (NL) input (e.g., sentences, phrases, words, etc.) into computer executable complete meaning representations (MRs) for domain-specific applications, where Meaning Representation Language (MRL) for an application is assumed to be present. Part of the job of semantic parsing is to extract the domain information from a natural language input. For example, if a user queries "find Dixie's Grill," a semantic parser should be able to determine that the user is looking for a restaurant named "Dixie's BBQ."

[004] Semantic parsing is a difficult computational problem. One reason for this is that NL inputs and their meanings may have many-to-many relationships. For example, multiple NL inputs may correspond to a single meaning, a single input could have different meanings (usually users intend to convey a single meaning, so system would need to interact with/learn from the user to disambiguate them), and other many-to-many relationships can occur. Conventional semantic parsers typically are inefficient at disambiguating a potential meaning

of an NL input, particularly when the criteria size is large. Furthermore, conventional semantic parsers typically have sub-optimal matching capabilities. As a result, conventional semantic parsers do not scale well for complicated information domains and often produce inaccurate MRs.

[005] Other types of information retrieval systems may scale better than some semantic parsers. For example, conventional Inverted-Index-Search ("IIS") information retrieval systems, which are premised on keyword search, typically employ indexed information domains to facilitate efficient search. However, these systems typically ignore structural information (e.g., semantic structural information) that may be included in an NL input and therefore may not account for a user's intent. For example, for an NL input "find bus stop near VOICEBOX," the results from a conventional IIS information retrieval system may relate to documents (e.g., webpages) that contain the keywords in the search input, without semantic information to help determine the intent of the user. Thus, the results may be irrelevant to the user's intended request.

[006] Thus, what is needed is to improve semantic parsing for enhanced user intent recognition, improve the relevancy of information retrieval results, and improve recognition of NL, whether uttered, typed, or otherwise provided by a user. These and other problems exist.

SUMMARY OF THE INVENTION

[007] The invention addressing these and other drawbacks relates to systems and methods of integrating domain information into state transitions of a Finite State Transducer ("FST") for natural language processing. A system may integrate semantic parsing and information retrieval from an information domain to generate an FST parser that represents the information domain. The FST parser may include a plurality of FST paths, at least one of which may be used to generate a meaning representation from a natural language input (e.g., a natural language string based on a natural language utterance or other input). As such, the system may perform domain-based semantic parsing of a natural language input to generate robust meaning representations (e.g., search queries, commands, etc.) using domain information. By generating more relevant meaning representations, results obtained by executing the meaning representations may be more accurately aligned with the user's intent.

[008] The system may be applied to a wide range of natural language applications that use natural language input from a user. For example, and without limitation, applications of domain-based semantic parsing may include natural language interfaces to computing systems, communication with robots in natural language, personalized digital assistants, question-answer query systems, and/or other natural language processing applications.

[009] To generate an FST parser, the system may obtain a semantic grammar and then structure a given FST path based on the semantic grammar. An FST path may therefore have a semantic structure based on the semantic grammar. The semantic grammar may include various criteria. For example, a semantic grammar "find <business names>" includes an action criterion ("find") and a subject criterion ("<business names>"). Other semantic grammars having additional and/or different criteria may be used as well.

[010] An FST path may include a plurality of states, where at least two states are separated by a state transition. An initial state transition may be associated with an action criterion such as "find." The system may add subsequent state transitions to an FST path based on an information domain. For example, subsequent state transitions may be associated with a token (e.g., word) that appears in an entry of an information domain. For an FST parser that represents an information domain related to businesses, a state transition from one state to another state may be associated with a token (e.g., word) that is part of a business name. Other tokens in the business name may be associated with other state transitions, thus growing the FST path.

[011] For example, an entry "Burnstead Construction Inc." may include tokens "Burnstead," "Construction," and "Inc." Each token may be associated with a state transition. For example, the token "Burnstead" may yield an FST path corresponding to "find Burnstead." Other tokens of the entry may be used as well, either to add to the existing FST path or create a new FST path. For instance, another token "Construction" may result in an FST path corresponding to "find Burnstead Construction." The system may generate an FST path that includes some or all tokens in a given entry, and may generate one or more FST paths for each entry of an information domain. The system may also generate different FST paths for different permutations of an entry in an information domain. For example, one FST path may correspond to "find Construction Burnstead Inc." while another FST path may correspond to

"find Burnstead Construction Inc." Still other FST paths may omit certain tokens. For example, such an FST path may correspond to "find Burnstead Inc." The system may repeat these operations until all entries in the information domain are similarly processed into respective FST paths. The FST parser may therefore be representative of an information domain.

[012] The system may associate a token appearing in an information domain with a weight based on its frequency of appearance in the information domain and/or other domains. For example, the system may associate a weight with a token in proportion to a frequency of the token's appearance in an information domain. In a particular example, the token "Construction" may be weighted differently (e.g., less or more depending on the weighting system) than "Burnstead" if "Construction" appears in various business names but "Burnstead" appears only in one business name. As such, a state transition associated with a token may be weighted accordingly.

[013] The system may score an FST path based on cumulative weights associated with its state transitions. For example, the system may score an FST path by summing the weights associated with state transitions in the FST path. The system may use the score to determine whether a given FST path should be used to generate a meaning representation for a given natural language input. For example, the system may use the FST parser to determine a best FST path (or n-best FST paths) by identifying an FST path that is: (1) associated with tokens that have (exact or fuzzy) matches to tokens of a natural language input, and (2) has the highest (or lowest, depending on the scoring system) score. For instance, if tokens of a natural language input match tokens associated with two or more FST paths, then the FST path having the highest score (or n-highest scores) will be selected. In this manner, a semantic grammar (which is used to structure FST paths of a FST parser) integrated with information retrieval from an information domain (which may be used to weight tokens associated with state transitions of FST paths) may be used to generate more robust, relevant meaning representations.

[014] Once an information domain FST parser has been generated, it may be used to compose an input string FST used to generate a meaning representation from a natural language input. In the example that follows, the natural language input "Coffee find Seattle

Best Starbucks Coffee" will be used, although other types of queries/commands/input may be used as well. For example, the system may initialize (e.g., structure) an input FST using the natural language input. The system may compose the input FST based on an information domain FST parser.

[015] The system may find the shortest path (or n-best shortest paths) in the composed input FST, remove redundant epsilon arcs (if any), topologically order the arcs, and generate a parsing result based on the topsorted arcs. Based on the parsing result, an action such as "find" will be mapped its semantic meaning (e.g., "[Action:find]" and the criterion (e.g, "Seattle Best Starbucks Coffee") will be mapped to an entry in an information domain corresponding to businesses, such as "[BusinessName:Best Starbucks Coffee WA]." The system may generate the query string based on the criteria content, i.e., the string between "[" and "]", which is "Best Starbucks Coffee."

[016] In an implementation, the system may compose the input FST with an Information Extraction FST that is used to ignore irrelevant words in the natural language input, add additional words that may relate to the natural language input but were not included in the input, and/or otherwise modify the input FST based on known information. In this manner, the natural language input may be augmented with relevant information (or stripped of irrelevant noise) so that the resulting meaning representation, when executed, causes more relevant results to be returned.

[017] In an implementation, the system may process the input FST to account for phoneme errors. For example, the system may use a phoneme confusion matrix that is trained from speech data to further refine the natural language input to result in a more robust meaning representation that accounts for similar-sounding or otherwise indistinguishable words.

[018] In an implementation, the system may integrate dynamic data, such as information from a personal phonebook, by generating a dynamic data FST based on the dynamic data and insert the dynamic data FST into a dynamic slot of a domain information FST parser. In this manner, dynamic data may be integrated into semantic parsing to generate a meaning representation based on both domain information and dynamic data.

[019] When the input FST is composed (regardless of whether an information extraction FST, a phoneme confusion matrix, or dynamic data is used), the system may find the shortest path

(or n-best shortest paths) from the input FST, remove redundant epsilon arcs (if any), and topsort the arcs. The system may calculate the ID of the best-matched path corresponding to an entry in the information domain based on the sum of the weights along the input FST and generate a meaning representation based on the best-matched path.

[020] Accordingly, the system may generate a meaning representation from a natural language input by using tokens that appear in the natural language input (also referred to herein as input tokens) to select a relevant FST path (which was formed using a semantic grammar and tokens that appear in an information domain (also referred to herein an domain tokens to distinguish from input tokens)). The meaning representation may then be used to extract information from and/or execute a command on an appropriate information domain. In this manner, the system may efficiently generate more robust meaning representations that may lead to more accurate, less error-prone, results.

[021] These and other objects, features, and characteristics of the system and/or method disclosed herein, as well as the methods of operation and functions of the related elements of structure and the combination of parts and economies of manufacture, will become more apparent upon consideration of the following description and the appended claims with reference to the accompanying drawings, all of which form a part of this specification, wherein like reference numerals designate corresponding parts in the various figures. It is to be expressly understood, however, that the drawings are for the purpose of illustration and description only and are not intended as a definition of the limits of the invention. As used in the specification and in the claims, the singular form of "a", "an", and "the" include plural referents unless the context clearly dictates otherwise.

BRIEF DESCRIPTION OF THE DRAWINGS

[022] FIG. 1 illustrates a system of integrating domain information into state transitions of a Finite State Transducer for natural language processing, according to an implementation of the invention.

[023] FIG. 2 which depicts a process of generating an FST parser, according to an implementation of the invention.

[024] FIG. 3 depicts an FST path structured by a semantic grammar, according to an

implementation of the invention.

[025] FIG. 4 depicts a process of integrating information from an information domain to generate semantically structured FST paths based on tokens from the information domain, according to an implementation of the invention.

[026] FIG. 5 depicts an FST parser structured by a semantic grammar and expanded using information and weights from an information domain, according to an implementation of the invention.

[027] FIG. 6 schematically depicts an optimized version of the FST parser illustrated in FIG. 5, according to an implementation of the invention.

[028] FIG. 7 depicts a process of composing a meaning representation from a natural language input using an FST parser, according to an implementation of the invention.

[029] FIG. 8 depicts an input string FST that is initialized, according to an implementation of the invention.

[030] FIG. 9 depicts an input string FST that is composed using a domain information FST parser, according to an implementation of the invention.

[031] FIG. 10 depicts a 1-best FST selected using the input string FST, according to an implementation of the invention.

[032] FIG. 11 depicts an input FST composed with an information extraction FST, according to an implementation of the invention.

[033] FIG. 12 depicts a process of generating a domain information extraction FST, according to an implementation of the invention.

[034] FIG. 13 depicts an FST that expands every possible FST path at the terminal level, according to an implementation.

[035] FIG. 14 depicts a process of using wildcards to process a natural language input having variable text, according to an implementation of the invention.

[036] FIG. 15 illustrates an input FST generated for an input string in which state transitions and a temporary symbol table are generated, according to an implementation of the invention.

[037] FIG. 16 illustrates an example of an input string FST composed with an FST parser, according to an implementation of the invention.

[038] FIG. 17 depicts a modified and expanded input string FST, according to an implementation of the invention.

[039] FIG. 18 depicts a recreated input string FST, according to an implementation of the invention.

[040] FIG. 19 depicts an input string FST composed from a recreated input string FST and a modified and expanded input string FST, according to an implementation of the invention.

[041] FIG. 20A depicts a dynamic data FST using an exact match, according to an implementation of the invention.

[042] FIG. 20B depicts a dynamic data FST using fuzzy matching, according to an implementation of the invention.

[043] FIG. 21A depicts an FST parser in which a dynamic slot is filled with a dynamic data FST, which uses exact matching, according to an implementation of the invention.

[044] FIG. 21B depicts an FST parser in which a dynamic slot is filled with dynamic data FST, which uses fuzzy matching, according to an implementation of the invention.

[045] FIG. 22 depicts an FST having two phonemes and allowing up to two errors (including insertions, deletions, and replacements), with a penalty of 100 corresponding to each error, according to an implementation of the invention.

[046] FIG. 23 depicts a flow diagram of FST parsing at the phoneme level, according to an implementation of the invention.

DETAILED DESCRIPTION OF THE INVENTION

[047] FIG. 1 illustrates a system 100 of integrating domain information into state transitions of a Finite State Transducer for natural language processing, according to an implementation of the invention. A system may integrate semantic parsing and information retrieval from an information domain to generate an FST parser that represents the information domain. The FST parser may include a plurality of FST paths, at least one of which may be used to generate a meaning representation from a natural language input (e.g., natural language utterance and/or other natural language input).

[048] The meaning representation may be used to execute an action (e.g., a search, a command, and/or other action) related to an information domain. As such, in an

implementation, the meaning representation is not necessarily a search result itself, but rather a semantically parsed translation of a natural language input that uses domain information to help understand the intent of the natural language input. Thus, the system may receive a user request in the form of a natural language input from a user (which may or may not have been recognized from an utterance), generate a meaning representation as described herein, and execute the meaning representation responsive to the user's request.

[049] Examples of an FST parser are used herein throughout for illustration and not limitation. The principles of integrating semantic parsing and information retrieval may be applied to other semantic parsing systems and other contexts as well. For example, more generically: [describe claim 1].

[050] Other uses of system 100 are described herein and still others will be apparent to those having skill in the art. Having described a high level overview of some of the system functions, attention will now be turned to various system components that facilitate these and other functions.

[051] System 100 may include a computer system 104, one or more information domains 150, one or more computing device(s) 160, and/or other components.

[052] Information Domains

[053] An information domain 150 may include a set of information that includes subject matter that may be searched or otherwise executed on. For example, an information domain 150 may include a listing of businesses, a listing of addresses, a listing of songs, a listing of personal contacts, and/or other corpus of information that may be searched, played or otherwise executed. Each information domain 150 may have its own type of information. For example, one information domain 150 may relate to businesses while another information domain 150 may relate to songs. In an implementation, an information domain 150 may include a combination of different types of information. For example, one information domain 150 may relate to businesses and songs.

[054] An information item in an information domain 150 may be associated with a token or keyword. For example, a listing of businesses may include the name of a business and/or other business-related information. A listing of songs may include a song title, artist name, genre, and/or other song-related information. An information domain 150 may be indexed to

facilitate rapid searching of its content, such as its tokens.

[055] Computer system 104

[056] Computer system 104 may include one or more computing devices 110. Each computing device 110 may include one or more processors 112, one or more storage devices 114, and/or other components. Processor(s) 112 may be programmed by one or more computer program instructions, which may be stored in storage device(s) 114.

[057] The one or more computer program instructions may include, without limitation, an information retrieval and semantic integration application 120 ("IRSI application 120"). IRSI application 120 may itself include different sets of instructions that each program the processor(s) 112 (and therefore computer system 104). For example, IRSI application 120 may include a domain FST parser generator 130, a natural language input processor 140, and/or other instructions that program computer system 104. As used herein, for convenience, the various instructions will be described as performing an operation, when, in fact, the various instructions program computer system 104 to perform the operation.

[058] Generating an FST Parser Representative of an Information Domain

[059] Domain FST parser generator 130 may generate an FST parser that integrates information retrieval from an information domain 150 and a semantic grammar. For example, FIG. 2 depicts a process 200 of generating an FST parser, according to an implementation of the invention. The various processing operations and/or data flows depicted in FIG. 2 (and in the other drawing figures) are described in greater detail herein. The described operations may be accomplished using some or all of the system components described in detail above and, in some implementations, various operations may be performed in different sequences and various operations may be omitted. Additional operations may be performed along with some or all of the operations shown in the depicted flow diagrams. One or more operations may be performed simultaneously. Accordingly, the operations as illustrated (and described in greater detail below) are exemplary by nature and, as such, should not be viewed as limiting.

[060] Structuring FST paths of an FST Parser using a semantic grammar

[061] In an operation 202, FST parser generator 130 (or other instructions) may obtain a semantic grammar and structure an FST path based on the semantic grammar. An FST path may include a plurality of states, where at least two states are separated by a state transition.

A semantic grammar may include at least an action criterion and a subject criterion. For example, a semantic grammar "find <BusinessNames>" includes an action criterion ("find") and a subject criterion ("<BusinessNames>"), in which an initial state is depicted using bold and a final state is depicted with double circles. Other semantic grammars having additional and/or different criteria may be used as well.

[062] FST parser generator 130 may structure an FST path using the criteria of the semantic grammar to form state transitions from one state to another state such that each criterion is represented as a node in the FST path. For example, FIG. 3 depicts an FST path 300 structured by a semantic grammar (in the illustrated example: "find <BusinessNames>"), according to an implementation of the invention. A given state (or node 0, 1, 2, 3, 4 as illustrated in FIG. 3) may be transitioned to another state via a state transition, which is illustrated by an arrow (also described herein as an "arc," which may be labeled).

[063] Integrating domain information into semantically structured FST paths of an FST Parser

[064] In an operation 204, FST parser generator 130 may integrate information from an information domain 150 to generate semantically structured FST paths based on tokens (e.g., words) from the information domain. For example, FST parser generator 130 may add state transitions to a semantically structured FST path based on tokens from an information domain. Subsequent state transitions may be associated with a token that appears in an entry of an information domain 150.

[065] Table 1 illustrates non-limiting examples of entries in an information domain related to businesses. Such an information domain will typically have many more entries (e.g., an entry for each known business), but the illustrated information domain is illustrated with five entries for illustrative purposes. Furthermore, an information domain may include alternative or additional types of information. The following table is therefore included solely to illustrate aspects of the disclosure and will be described in conjunction with FIGS. 4, 5, and 6 that follow.

entry weight	*Entry Full Name*
3	Best Starbucks Coffee WA
1	Burnstead Construction Inc WA
1	Microsoft WA
1	Seattle Best Coffee WA
1	VoiceBox Technologies Inc WA

[066] For an FST parser representing an information domain illustrated in Table 1, a state transition from one state to another state may be associated with a token that is part of a business name. Other tokens in the business name may be associated with other state transitions, thus growing an FST path that corresponds to an entry in the information domain.

[067] For example, an entry "Burnstead Construction Inc." may include tokens "Burnstead," "Construction," and "Inc." Each token may be associated with a state transition. The token "Burnstead" may yield an FST path corresponding to "find Burnstead." Other tokens of the entry may be used as well, either to add to the existing FST path or create a new FST path. For instance, another token "Construction" may yield an FST path corresponding to "find Burnstead Construction." The system may generate an FST path that includes some or all tokens in a given entry, and may generate one or more FST paths for each entry of an information domain. The system may also generate different FST paths for different permutations of an entry in an information domain. For example, one FST path may correspond to "find Construction Burnstead Inc." while another FST path may correspond to "find Burnstead Construction Inc." The system may repeat these operations until all entries in the information domain are similarly processed into respective FST paths. The FST parser may therefore be representative of an information domain 150.

[068] FST parser generator 130 may associate a token appearing in an information domain with a weight based on its frequency of appearance in the information domain and/or other domains. For example, FST parser generator 130 may associate a weight with a token in proportion to a frequency of the token's appearance in an information domain. In a particular example, the token "Construction" may be weighted differently (e.g., less or more depending on the weighting system) than "Burnstead" if "Construction" appears in various business

names but "Burnstead" appears only in one business name.

[069] The token weight may serve to indicate which tokens may be more relevant than others when matching input tokens from an input request (e.g., a user search request) with tokens from an FST path (e.g., so that the relevance of a given FST path to the input tokens may be determined). For example, an FST path having matching tokens that commonly appear in an information domain may be less relevant to an input request than an FST path having matching token that do not commonly appear in the information domain.

[070] FST parser generator 130 may score an FST path based on cumulative weights associated with its state transitions. For example, the system may score an FST path by summing the weights associated with state transitions in the FST path. FST parser generator 130 may use the score to determine a relevance of a given FST path to a given natural language input. For example, the system may use the FST parser to determine a best FST path (or n-best FST paths) by identifying an FST path that is: (1) associated with tokens that have (exact or fuzzy) matches to tokens of a natural language input, and (2) has the highest (or lowest, depending on the scoring system) score. For instance, if tokens of a natural language input match tokens associated with two or more FST paths, then the FST path having the highest score (or n-highest scores) will be selected. In this manner, a semantic grammar (which is used to structure FST paths of a FST parser) integrated with information retrieval from an information domain (which may be used to weight tokens associated with state transitions of FST paths) may be used to generate more robust, relevant meaning representations.

[071] FIG. 4 depicts a process 400 of integrating information from an information domain 150 to generate semantically structured FST paths based on tokens from the information domain, according to an implementation of the invention. Operation 204 illustrated in FIG. 2 may use some or all of the processing operations of process 400 to integrate information from an information domain 150.

[072] In an operation 402, domain FST parser generator 130 may obtain a criteria list, which may include entries in an information domain. For example, domain FST parser generator 130 may obtain the business name entries in the exemplary information domain illustrated in Table 1.

[073] In an operation 404, domain FST parser generator 130 may determine a weight, such as a term frequency inverse document frequency value ("TF-IDF") value, for each token in the criteria list. For example, domain FST parser generator 130 may determine a TF-IDF value for each token that appears in the criteria list, whether in a single entry or multiple entries. As would be appreciated, a more common token (e.g., one that occurs more than another token) may be associated with a lower TF-IDF value while a less common token may be associated with a higher TF-IDF value.

[074] In an operation 406, domain FST parser generator 130 may obtain and process a next entry in the criteria list. For example, once tokens for each entry of an information domain have been assigned a TF-IDF value depending on their appearance in the information domain, domain FST parser generator 130 may iteratively obtain and process a next business name in the information domain.

[075] In an operation 408, domain FST parser generator 130 may obtain a list weight for a given entry (if any). For example, a given business name entry may be weighted more or less heavily than another business name entry.

[076] In an operation 410, domain FST parser generator 130 may determine a token weight for each token in an entry. The token weight may include a TF-IDF value of the token and may be adjusted based on the list weight (if any) for an entry in which the token appears.

[077] In an implementation, each token in each list may have a different weight. For example, a TF-IDF value may be calculated for each token using equation (1):

$$\text{TF-IDF} = \frac{number\ of\ occurances\ of\ i\ in\ list\ j}{document\ length\ of\ list\ j} * \log \frac{Total\ number\ of\ lists}{number\ of\ lists\ that\ contain\ token\ i} \quad (1),$$

wherein:

$list_j$ corresponds to an entry in an information domain (also referred to herein as list $entry_j$);

i = a given token;

document length of list j = $\sqrt[3]{number\ of\ words}$;

[078] In an implementation, F_{1j} (the closeness of the candidate i for list entry$_j$) may be calculated using equation (2):

$$F_{ij}=$$

$$C_1 * (\#\ of\ matched\ tokens) + C_2 * \sum_{each\ \text{token}\ x\ \text{in List Entry i}} TFIDF(x, j) + C_3 * (word\ \ position\ metrics) + C_4 * Weight_j \quad (2),$$

wherein:

C_1, C_2, C_3, C_4 are pre-defined weights; Weight$_j$ is the priori-likelihood for *j* (e.g., "Starbucks" may be given more weight than "Dixie's BBQ"); and

C_3 is *0*.

[079] According to equation (2), a higher closeness value indicates a better candidate. Referring to equation (2), "word position metrics" refers to a level of closeness of candidate i to list entry j in terms of word order/position.

[080] In the non-limiting example that follows, the "word position metrics" between "i" and "k" should be 1, because it is an exact match. The "word position metrics" between "i" and "j" should be 0.8, because one word ("wa") is not in the same order between Candidate I and List entry j:

Candidate i : "burnstead construction inc. wa"

List entry j: "wa burnstead construction inc."

List entry k: "burnstead construction inc. wa"

[081] In an implementation, to yield the lowest (positive) value for the best candidate, F_{1j} (the closeness of the candidate *j* for token *i* for which the lowest value indicates the best candidate) may be calculated using equation (3):

$$F_{2j}= C_1 * (\#\ of\ not\ matched tokens) + C_2 * \sum (TFIDF_{\max} - tfidf_{ij}) + C_4 * (Weight_{\max} - Weight_j) \quad (3),$$

wherein:

$TFIDF_{max}$ = the maximum value of all TFIDF values for all tokens in all list entries; and

$Weight_{max}$ = the maximum value of all TFIDF values for all tokens in all list entries.

[082] Assuming N is the total number of words in the query string, then:

$$F_{2j} = \begin{aligned} & C_1 * (N - \#\, of\ not\ matched\ tokens) + C_2 * \sum TFIDF_{\max} - C_2 * \sum tfidf_{ij} \\ & + C_4 * Weight_{\max} - C_4 * Weight_j \end{aligned}$$

$$= \begin{aligned} & C_1 * N + C_2 * \sum TFIDF_{\max} + C_4 * Weight_{\max} \\ & - C_1 * (\#\, of\ not\ matched\ tokens) - C_2 * \sum tfidf_{ij} - C_4 * Weight_j \end{aligned}$$

$$= C_1 * N + C_2 * \sum TFIDF_{\max} + C_4 * Weight_{\max} - F_{1j} \quad (4).$$

wherein:

$TFIDF_{max}$ is the maximum value of all TFIDF values for all tokens in all list entries;

Weightmax is the maximum value of all weights for all list entries; and

Weightj is the weight for list entry j.

[083] In an implementation, the token weight may be further weighted based on a list weight (if any) associated with the entry.

[084] In an operation 412, domain FST parser generator 130 may create an arc from the start point (e.g., a command state or node) to each token such that a state transition from the start point to a given token node (e.g., state associated with a token) is associated with the token weight.

[085] In an operation 414, domain FST parser generator 130 may create another arc from a previous token node (if it exists) to the current token node such that a state transition from the previous token node to the current token node is associated with the token weight.

[086] In an operation 416, domain FST parser generator 130 may mark all token nodes as a final state.

[087] In an operation 418, domain FST parser generator 130 may determine whether more entries in the list are to be processed. If more entries are to be processed, domain FST parser generator may return to operation 406. Otherwise, FST parser generator 130 may complete generation of the FST parser in an operation 420.

[088] FIG. 5 schematically depicts an FST parser 500 structured by a semantic grammar and expanded using information and weights from an information domain, according to an implementation of the invention. FST parser 500 schematically depicts results from the processing operations of process 400 using the information domain illustrated in Table 1. FST parser 500 may include a plurality of nodes (0-21). A given node (0-21) may be associated

with a token, an arc, and/or an end point. Each node (0-21) may be part of an FST path.

[089] Optimizing an FST parser

[090] Returning to FIG. 2, in an operation 206, domain FST parser generator 130 may optimize the FST parser generated from operation 204 (additional details of which were described with respect to process 400 in FIG. 4).

[091] For example, domain FST parser generator 130 may determinize an FST parser such that each state may have at most one transition with any given input label and empty labels are eliminated. The resulting FST may contain at most one patch matching any given input string, therefore reducing the time and space needed to process the string. Conditions of a determinizable FST include, for example, weakly left divisible semirings, a determinization algorithm can finish on M, all unweighted acceptors are determinizable, all acyclic acceptors are determinizable, not all weighted acceptors or transducers are determinizable, and characterization based on the twins property.

[092] In an implementation, domain FST parser generator 130 may further reduce the size of an FST parser by applying minimization, saving space and processing time. In an implementation, domain FST parser generator 130 may further reduce the size of an FST parser by performing an arc sort, which sorts the arcs in an FST per state. FIG. 6 schematically depicts an optimized version 600 of the FST parser 500 illustrated in FIG. 5, according to an implementation of the invention. As illustrated, optimized version 600 includes only nodes 0-11, in which various nodes and arcs have been removed from FST parser 500 during an optimization process to generate optimized version 600.

[093] Composing a Meaning representation from a Natural Language Utterance using the FST Parser

[094] In an implementation, natural language input processor 140 may generate a meaning representation from a natural language input by selecting and applying an FST parser, which is generated by FST parser generator 130. The natural language input may include an input string. The input string may have been converted from a user utterance using conventional speech-to-text techniques or may have been input as text from the user. In any event, the natural language input may be processed to generate a meaning representation using an FST parser (e.g., an FST parser generated by domain FST parser generator 130).

[095] For example, FIG. 7 depicts a process 700 of composing a meaning representation from a natural language input using an FST parser, according to an implementation of the invention.

[096] In an implementation, in an operation 702, natural language input processor 140 may initialize (e.g., structure) an input string FST using the input string. For example, for each word in the query string, natural language input processor 140 may:

- add an arc from a current node (corresponding to a given word of the input string being processed) to a next node depending on whether the given word matches with a word from an information domain. For example, natural language input processor 140 may add one arc from a current node to a next node with an input label being the word ID, an output label being the word ID, and a weight being 0. This arc means there is no cost when the input word is matched. Natural language input processor 140 may add one arc from a current node to a next node with an input label being the word ID, an output label being epsilon, and a weight being C1. This arc means the cost will be C1 when the input word is not matched.
- add one self-loop arc to a current node with an input label being epsilon, an output label being left boundary, and a weight being 0. This arc means the left boundary could happen at any location of the input string without consuming any input label.
- add one self-loop arc to next node with input label being epsilon, output label being right boundary, and weight being 0. This arc means the right boundary could happen at any location of the input string without consuming any input label.

[097] For the last word of the input string, natural language input processor 140 may set the last node as the final state. FIG. 8 depicts an input string FST (having nodes 0-8) that is initialized according to operation 702 if the input query string is "Coffee find Seattle Best Starbucks Coffee," according to an implementation of the invention.

[098] In an operation 704, natural language input processor 140 may compose the input FST based on an information domain FST parser (e.g., an FST parser generated by domain FST parser generator 130). FIG. 9 depicts an input string FST 900 that is composed using a domain information FST parser, according to an implementation of the invention. Input string FST 900 is composed in operation 704 using, for example, a domain information FST parser that is

based on Table 1 (e.g., FST parser 500 or FST parser 600). For example, an output of the domain information FST parser may be used as in put to FST 900.

[099] In an operation 706, natural language input processor 140 may find the shortest path (or n-best shortest paths) in the composed input FST. FIG. 9 depicts an input string FST 900 (having nodes 0-24) that is composed using a domain information FST parser, according to an implementation of the invention. FIG. 10 depicts a 1-best FST 1000 (having nodes 0-8) selected using the input string FST, according to an implementation of the invention.

[0100] In an operation 708, natural language input processor 140 may remove redundant epsilon arcs (if any).

[0101] In an operation 710, natural language input processor 140 may topologically order (i.e., "topsort") the arcs.

[0102] In an operation 712, natural language input processor 140 may generate a parsing result based on the topsorted arcs. For example, for the following represent an input, parsing result, and weight:

[0103] Input: Coffee find Seattle Best Starbucks Coffee

[0104] Parsing Result: find [Best Starbucks Coffee]

[0105] Weight: 0 148.92 85.92 148.92

[0106] Based on the parsing result, "find" will be mapped its semantic meaning (e.g., "[Action:find]" the string "Seattle Best Starbucks Coffee" will be mapped to an entry in an information domain, such as "[BusinessName:Best Starbucks Coffee WA]" as it appears in Table 1.

[0107] In an operation 714, natural language input processor 140 may generate the query string based on the criteria content, i.e., the string between "[" and "]", which is "Best Starbucks Coffee."

[0108] In an operation 716, natural language input processor 140 may compose the input FST with the Information Extraction FST (the generation of which is illustrated with respect to FIG. 12). FIG. 11 depicts an input FST 1100 (having nodes 0-24) composed with an information extraction FST, according to an implementation of the invention.

[0109] In an operation 718, natural language input processor 140 may find the shortest path (or nbest shortest paths) from the input FST (e.g., input FST 1100).

[0110] In an operation 720, natural language input processor 140 may remove redundant epsilon arcs (if any) and topsort the arcs.

[0111] In an operation 722, natural language input processor 140 calculate the ID of the best-matched path corresponding to an entry in the information domain based on the sum of the weights along the FST. The query input may be generated based on the best-matched path. In some implementations, by selecting the best-matched path (and corresponding entry in the information domain), natural language input processor 140 may ignore words in the input string that may be irrelevant to the information domain while using words in the input string that are relevant to the information domain to formulate a meaning representation that, when used to obtain information from the information domain, will result in more relevant results.

[0112] Generating an Domain Information Extraction FST

[0113] In an implementation, natural language input processor 140 may generate a domain information extraction FST to obtain detailed information related to an input string. For example, a domain information extraction FST may be used to ignore irrelevant ("noise") words, add relevant words that are likely related to the input string but have not been included in the input string, and/or otherwise use domain information to fine tune an input string (such as by obtaining a full title of a song when only a portion of the song title is included in the input string).

[0114] In another example, the following illustrates an input, a parsed input without domain information extraction, and a parsed input with domain information extraction:

[0115] Input: find Cavanaugh Rd Normal Town Pennsylvania

[0116] After parser FST: find [Cavanaugh Rd Pennsylvania]

[0117] After Extraction FST: find [Cavanaugh Rd Normalville Pennsylvania]

[0118] In the foregoing example, input to the Extraction FST is "Cavanaugh Rd Pennsylvania", the output is "Cavanaugh Rd Normalville Pennsylvania," where the city name "Normalville" is added to fine tune the input.

[0119] In some implementations, natural language input processor 140 may extract (and add to an input) only the best matched information, instead of N-best matches. In other implementations, natural language input processor 140 may extract n-best matches using an

inverted-index-search approach.

[0120] FIG. 12 depicts a process 1200 of generating a domain information extraction FST, according to an implementation of the invention.

[0121] In an operation 1202, natural language input processor 140 may create an FST for the list, adding a last arc with a last arc indicator (e.g., "list_N," where "N" corresponds to a list entry number that is incremented with each list entry) for each entry in the list.

[0122] In an operation 1204, natural language input processor 140 may optimize the FST, such as by removing epsilons (if any), performing determinization, and/or other performing other optimizations.

[0123] In an operation 1206, natural language input processor 140 may remove the non-sense arcs from start state to end state directly. For each arc with input label being real words, natural language input processor 140 may remove its weights. For each arch with input label being list_N, for each pair of states, only keep only one outbound arc (leaving the state) which has the smallest weight or which has the smallest ID when they have the same weights, change the input label to <epsilon> and set the weight as the "N" (the number ID of the list).

[0124] In an operation 1208, natural language input processor 140 may optimize the FST by removing epsilons (if any), performing determinization, performing minimization, and/or performing other operations.

[0125] In an implementation, natural language input processor 140 may use the FST to return the numeric ID of a best matched list as the weight given an input string.

[0126] Grammar to FST Conversion

[0127] In an implementation, the system may apply an FST only once, generating an FST such that the FST expands every possible FST path at the terminal level. FIG. 13 depicts an FST 1300 (having nodes 0-9) that expands every possible FST path at the terminal level, according to an implementation of the invention. FST 1300, as illustrated, is constructed based on the following grammar:

public $Rule1 = word1 $Rule2 word2

protected $Rule2 = word3

[0128] In the illustrated example, in order to show the rule information in the parse result, the

system may create transitions such as “ε:rule_beginning” and “ε:rule_end”.

[0129] Matching Variable Text Using Wildcards

[0130] A natural language input may include variable text that may include different words and meanings. In these instances, the system may need to process such variable text to understand how to generate an input query from the natural language input. The system may do so using wildcards. For example, in a natural language input that takes the form of “text $text_body to $person_name,” the variable text $text_body and $person_name may include any word or words.

[0131] The system may introduce a wildcard (e.g., a special word “__vbt_wild_card__”) that can match any word. For example, the foregoing example, with wildcard added, may be represented as:

public $Text=

text $text_body to $person_name;

protected $text_body=

__vbt_wild_card__ <1->;

protected $person_name=

__vbt_wild_card__ <1->;

[0132] In the description of FIGS. 14-19 that follows, the natural language input of “text lunch together to min tang” will be used by way of illustration and not limitation. FIG. 14 depicts a process 1400 of using wildcards to process a natural language input having variable text, according to an implementation of the invention.

[0133] In an operation 1402, an input string FST may be generated for an input string (e.g., a natural language input string). For each word in the input string, three state transitions may be generated: (1) W:W/0, (2) W:ε/C_1, and (3) W:*/C_1-C_2. A temporary symbol table may also be created, which stores the index of out-of-vocabulary words (e.g., words that do not appear in an information domain). In this case, it contains the following words “lunch”, ”together”, ”min” and ”tang”. In an implementation, the life cycle of the temporary symbol table is only one session. FIG. 15 illustrates an input FST 1500 (having nodes 0-6) generated for an input

string in which state transitions and a temporary symbol table are generated, according to an implementation of the invention. For example, input FST 1500 may be generated based on operation 1402.

[0134] In an operation 1404, the input FST may be composed with a FST parser (e.g., a FST parser generated by domain FST parser generator 130). In an implementation, project to output label may be performed, epsilons may be removed, and the shortest path may be found. FIG. 16 illustrates an example of an input string FST 1600 (having nodes 0-27) composed with an FST parser according to an implementation of the invention. For example, input string FST 1600 may be composed based on operation 1404.

[0135] In an operation 1406, input string FST 1600 may be modified and expanded. For example, for each transition:

- If the output label *w* is not a terminal, change transition from *w*:*w* to *0*:*w*;
- If the output label *w* is a terminal;
- If *w* is the wildcard symbol, expand it to multiple transitions;
- Otherwise keep the transition as *w*:*w*.

[0136] FIG. 17 depicts a modified and expanded input string FST 1700 (having nodes 0-27), according to an implementation of the invention. For example, modified and expanded input string FST 1700 may be generated based on operation 1406.

[0137] In an operation 1408, the input string FST may be recreated without wildcards (since such wildcards are accounted for in operation 1406). FIG. 18 depicts a recreated input string FST 1800 (having nodes 0-6), according to an implementation of the invention. For example, recreated input string FST 1800 may be generated based on operation 1408.

[0138] In an operation 1410, a final FST may be generated by composing recreated input string FST 1800 and modified and expanded input string FST 1700, performing project to output label, removing epsilons, and finding the shortest path. FIG. 19 depicts an input string FST 1900 (having nodes 0-27) composed (e.g., using output) from recreated input string FST 1800 and modified and expanded input string FST 1700, according to an implementation of the invention.

[0139] In an operation 1412, the parsing result may be output. For example, depth-first-search may be used to traverse the final FST.

[0140] For example, in XML format, the parsing result for the above may include:

NBest 1: (score = 3960000)

```
<text>
 <Text>
  text <text_body> lunch together </text_body> to <person_name> min tang </person_name>
 </Text>
</text>
```

NBest 2: (score = 3970000)

```
<text>
 <Text>
  text <text_body> lunch together </text_body> to <person_name> tang </person_name>
 </Text>
</text>
```

NBest 3: (score = 3970000)

```
<text>
 <Text>
  text <text_body> together </text_body> to <person_name> min tang </person_name>
 </Text>
</text>
```

[0141] In JSON format, the parsing result for the above may include:

NBest 1: (score = 3960000)

```
[{"Text":{"text_body":" lunch together ","person_name":" min tang "}}]
```

NBest 2: (score = 3970000)

```
[{"Text":{"text_body":" lunch together ","person_name":" tang "}}]
```

NBest 3: (score = 3970000)

```
[{"Text":{"text_body":" together ","person_name":" min tang "}}]
```

[0142] Other output formats may be used as well to represent and view parsing results. Furthermore, in an implementation, the wildcard could be also supported by using a customized matchers, such as a SigmaMatcher<M> function.

[0143] Dynamic Update

[0144] In an implementation, instead of matching a predefined information domain or using wildcards, users may wish to query dynamic data that regularly changes or is otherwise frequently updated. For example, dynamic data may include a phone book (e.g., personal contact lists), personal documents, and/or other information that may regularly change.

[0145] In an implementation, the use of dynamic data may be supported by defining a dynamic slot in the grammar. For example, a dynamic data FST may be generated based on dynamic data (similar to the manner in which FST parser generator 130 generates an FST parser). The dynamic data FST may be used to fill a dynamic slot in an FST parser.

[0146] Table 2 illustrates non-limiting examples of dynamic data entries in a phone book. Such dynamic data will typically have many more entries (e.g., an entry for each contact), but the illustrated dynamic data is illustrated with two entries for illustrative purposes. Furthermore, the dynamic data may include alternative or additional types of information. The following table is therefore included solely to illustrate aspects of the disclosure and will be described in conjunction with FIGS. 20A, 20B, 21A, and 21B that follow.

Phone_Book	Name	Weight
Person 1	min tang	1
Person 2	Obama	1

[0147] In an implementation, an exact match and/or a fuzzy match may be used when generating a dynamic data FST. FIG. 20A depicts a dynamic data FST 2000A (having nodes 0-2) using an exact match, according to an implementation of the invention. FIG. 20B depicts a dynamic data FST 2000B (having nodes 0-2) using fuzzy matching, according to an implementation of the invention.

[0148] In an implementation, a dynamic slot of an FST parser may be filled with dynamic data FST 2000A or dynamic data FST 2000B. FIG. 21A depicts an FST parser 2100A (having nodes 0-13) in which a dynamic slot is filled with dynamic data FST 2000A, which uses exact matching, according to an implementation of the invention. FIG. 21B depicts an FST parser 2100B

(having nodes 0-13) in which a dynamic slot is filled with dynamic data FST 2000B, which uses fuzzy matching, according to an implementation of the invention.

[0149] An example of an input and parsing output (in JSON format) based on use of dynamic updates using dynamic data includes:

input: text talk to me to min tang

output In JSON format:

NBest 1: (score = 2970100)

[{"Text":{"text_body":" talk to me ","person_name":" min tang "}}]

NBest 2: (score = 2980100)

[{"Text":{"text_body":" talk me ","person_name":" min tang "}}]

NBest 3: (score = 2980100)

[{"Text":{"text_body":" to me ","person_name":" min tang "}}]

[0150] Named Entity Extraction Support

[0151] In an implementation, FSTs generated herein may integrate character level information in order to help the system determine intent.

[0152] For example, for the input: “call (425)123-4567 please,” a human being is able to figure out the intent to make a phone call to (425)123-4567. However, in order for a computer to recognize the intent, absent named entity information, wildcards may be required to create a pattern such as “call *”. This is not an optimal solution, because the aggressive wildcard would capture noise, such as “please” in the foregoing input example. To avoid abusing the usage of wildcard, we apply Named Entity Extraction (“NER”) process before parsing.

[0153] For example, assuming that the NER process returns the following NBest result:

NBest 1: call (425)123-4567

NBest 2: call <__vbtPN__> (425)123-4567 </__vbtPN__>

[0154] An input query FST may be created based on the above NBest result, and then parsed.

Using such processing, an output may include, for example:

NBest 1:

[{"CallByNumber":{"PhoneNumber":{"__vbtPN__":" (425)123-4567 "}}}]

[0155] Error Tolerance String Matching / Approximate String Matching

[0156] Errors in speech recognition may be caused by various error-introducing conditions. Such conditions may include, for example, homonyms, compound words, text normalization, and/or other issues that may cause errors in speech recognition. In the written language, for example, the strings "call (425)123-4567" and "call 4 2 5 1 2 3 4 5 6 7" are two different strings, but they share the same pronunciations (and could have different associated meanings or intent).

[0157] In an implementation, to help address the foregoing issues, the system may perform FST parsing at the phoneme level. By performing intent-recognition at the phoneme level, the system may handle sound-alike, homonym, compound-words, text normalization issues, and/or other error-introducing conditions. By injecting an error model (phonetic confusion matrix), the system may be able to make the parsing process robust to ASR errors (to some extent).

[0158] Table 3 illustrates a non-limiting example of a phoneme confusion matrix that is trained from speech data. In Table 3, ε stands for "null," which indicates that a corresponding phoneme is deleted or inserted; Pij is the penalty for phoneme i recognized as phoneme j, which case P_{ii} should be 0.

probability	ε	Phoneme 1	Phoneme 2	...	Phoneme i	...	Phoneme n
ε	P_{00}	P_{01}	P_{02}		P_{0i}		P_{0n}
Phoneme 1	P_{10}	P_{11}	P_{12}		P_{1i}		P_{1n}
Phoneme 2	P_{20}	P_{21}	P_{22}		P_{2i}		P_{2n}
...							
Phoneme i	P_{i0}	P_{i1}	P_{i2}		P_{ii}		P_{in}
...							

Phoneme n	P_{n0}	P_{n1}	P_{n2}		P_{ni}		P_{nn}

[0159] To allow any number of phonetic errors may be too computational demanding because. To restrict the size of possible candidates and limit the search space, the system may limit the number of allowed phonetic errors.

[0160] FIG. 22 depicts an FST 2200 (having nodes 0-2) having two phonemes and allowing up to two errors (including insertions, deletions, and replacements), with a penalty of 100 corresponding to each error, according to an implementation of the invention.

[0161] FIG. 23 depicts a flow diagram 2300 of FST parsing at the phoneme level, according to an implementation of the invention. In an implementation, a Phoneme Error Model 2302 may be generated using an input phoneme sequence/graph. Phoneme Error Model 2302 may include an FST that allows a given phoneme to transduce to another given phoneme, with a transition penalty (e.g., Pij defined in Table 3).

[0162] In an implementation, a Dictionary Model 2304 may be generated that includes an FST that defines each pronunciation of words in a given path. In an implementation, a Word-level Parser 2306 may be generated that includes an FST parser described herein, which is used to generate an output used for semantic queries.

[0163] Examples of System Architectures and Configurations

[0164] In an implementation, various system architectures may be used. For instance, some or all instructions of IRSI application 120 or other instructions may execute on different components of system 100. In particular, voice recognition (e.g., speech-to-text), one or more functions/operations of IRSI application 120, and/or other functions/operations described herein may be performed at computing device 110 and/or computing device 160.

[0165] For instance, computing devices 110 may include server devices and computing devices 160 may include user devices that connect to the server devices. Other architectures may be used as well.

[0166] Although illustrated in FIG. 1 as a single component, computer system 104 may include a plurality of individual components (e.g., computer devices) each programmed with at least some of the functions described herein. In this manner, some components of computer system 104 may perform some functions while other components may perform other

functions, as would be appreciated. The one or more processors 112 may each include one or more physical processors that are programmed by computer program instructions. The various instructions described herein are exemplary only. Other configurations and numbers of instructions may be used, so long as the processor(s) 112 are programmed to perform the functions described herein.

[0167] Furthermore, it should be appreciated that although the various instructions are illustrated in FIG. 1 as being co-located within a single processing unit, in implementations in which processor(s) 112 includes multiple processing units, one or more instructions may be executed remotely from the other instructions.

[0168] The description of the functionality provided by the different instructions described herein is for illustrative purposes, and is not intended to be limiting, as any of instructions may provide more or less functionality than is described. For example, one or more of the instructions may be eliminated, and some or all of its functionality may be provided by other ones of the instructions. As another example, processor(s) 112 may be programmed by one or more additional instructions that may perform some or all of the functionality attributed herein to one of the instructions.

[0169] The various instructions described herein may be stored in a storage device 114, which may comprise random access memory (RAM), read only memory (ROM), and/or other memory. The storage device may store the computer program instructions (e.g., the aforementioned instructions) to be executed by processor 112 as well as data that may be manipulated by processor 112. The storage device may comprise floppy disks, hard disks, optical disks, tapes, or other storage media for storing computer-executable instructions and/or data.

[0170] The various components illustrated in FIG. 1 may be coupled to at least one other component via a network, which may include any one or more of, for instance, the Internet, an intranet, a PAN (Personal Area Network), a LAN (Local Area Network), a WAN (Wide Area Network), a SAN (Storage Area Network), a MAN (Metropolitan Area Network), a wireless network, a cellular communications network, a Public Switched Telephone Network, and/or other network. In FIG. 1 and other drawing Figures, different numbers of entities than depicted may be used. Furthermore, according to various implementations, the components

described herein may be implemented in hardware and/or software that configure hardware.

[0171] The various information domains may be stored in one or more databases such as, for example, an Oracle™ relational database sold commercially by Oracle Corporation. Other databases, such as Informix™, DB2 (Database 2) or other data storage, including file-based, or query formats, platforms, or resources such as OLAP (On Line Analytical Processing), SQL (Structured Query Language), a SAN (storage area network), Microsoft Access™ or others may also be used, incorporated, or accessed. The database may comprise one or more such databases that reside in one or more physical devices and in one or more physical locations. The database may store a plurality of types of data and/or files and associated data or file descriptions, administrative information, or any other data.

[0172] Computing devices 110, 160 may each include a server computing device, a desktop computer, a mobile device that is generally portable (e.g., a laptop computer, a tablet computer, a "smartphone," etc.), or other computing device that is programmed by IRSI 120 and/or other computer program instructions.

[0173] Other implementations, uses and advantages of the invention will be apparent to those skilled in the art from consideration of the specification and practice of the invention disclosed herein. The specification should be considered exemplary only, and the scope of the invention is accordingly intended to be limited only by the following claims.

CLAIMS

What is claimed is:

1. A computer implemented method for integrating domain information and semantic parsing to generate meaning representations from natural language input, the method being implemented on a computer system having one or more physical processors programmed with computer program instructions to perform the method, the method comprising:

receiving, by the computer system, a natural language input comprising at least a first input token;

obtaining, by the computer system, a semantic grammar;

determining, by the computer system, a semantic structure of the natural language input based on the semantic grammar;

retrieving, by the computer system, from an information domain, a plurality of domain tokens that match the first token of the natural language input; and

generating, by the computer system, a meaning representation based on the semantic structure and the plurality of domain tokens, wherein the meaning representation is used to execute a computer executable action.

2. The method of claim 1, wherein the plurality of domain tokens are structured into a domain information FST parser that includes at least a first FST path comprising a first set of domain tokens and a second FST path comprising a second set of domain tokens, and wherein generating the meaning representation further comprises:

selecting the first FST path based on a first score of the first FST path and a second score of the second FST path, wherein the first FST path is used to generate the meaning representation.

3. The method of claim 2, wherein the first score is based on a first sum of weights of each domain token among the first set of domain tokens and the second score is based on a second sum of weights of each domain token among the second set of domain tokens, and wherein a given weight for a domain token is based on a level of frequency that the domain token appears in the information domain.

4. The method of claim 2, wherein retrieving the plurality of domain tokens comprises:

initializing an input FST based on the first token and the semantic structure; and

composing the input FST based on the first FST path and the second FST path, wherein the first FST path and the second FST path are integrated with the input FST, and wherein the first FST is selected from the input FST.

5. The method of claim 4, wherein selecting the first FST path comprises selecting a shortest path in the input FST.

6. The method of claim 4, wherein retrieving the plurality of domain tokens comprises:

performing fuzzy or exact matching between domain tokens from the information domain and the first token, wherein the plurality of domain tokens comprises fuzzy or exact matches to the first token.

7. The method of claim 1, the method further comprising:

identifying a second token that is relevant to the first token and the information domain, wherein the second token is not initially included in the natural language input; and

adding the second token to the meaning representation.

8. The method of claim 1, wherein the natural language input comprises at least a second token, the method further comprising:

determining that the second token is not relevant to the information domain; and

omitting the second token from the meaning representation responsive to the determination that the second token is not relevant.

9. The method of claim 1, the method further comprising:

obtaining a phoneme confusion matrix comprising at least two similar sounding words that are disambiguated based on previous training from one or more user utterances; and

disambiguating the first token based on the phoneme confusion matrix.

10. The method of claim 1, the method further comprising:

obtaining one or more dynamic data tokens from a dynamic data source; and

integrating the one or more dynamic data tokens with the plurality of tokens from the information domain, wherein the meaning representation is determined based on the integrated dynamic data tokens.

11. The method of claim 10, wherein the plurality of domain tokens are structured into a domain information FST parser that includes at least a first FST path comprising a first set of domain tokens and a second FST path comprising a second set of domain tokens, and wherein integrating the one or more dynamic data tokens comprises:

generating a dynamic FST based on the one or more dynamic data tokens; and

inserting the dynamic FST into a slot of the domain information FST parser reserved for dynamic data.

12. The method of claim 1, wherein the computer executable action comprises an execution of: a natural language-based search request or a natural language-based command.

13. The method of claim 1, wherein the information domain comprises a plurality of entries of searchable information, and wherein retrieving the plurality of domain tokens that match the first token comprises:

determining at least one entry, which includes the plurality of domain tokens, that is likely being searched for based on the first token.

14. A system for integrating domain information and semantic parsing to generate meaning representations from natural language input, the system comprising:

a computer system comprising one or more physical processors programmed with computer program instructions to:

receive a natural language input comprising at least a first input token;

obtain a semantic grammar;

determine a semantic structure of the natural language input based on the semantic grammar;

retrieve from an information domain, a plurality of domain tokens that match the first token of the natural language input; and

generate a meaning representation based on the semantic structure and the plurality of domain tokens, wherein the meaning representation is used to execute a computer executable action.

15. The system of claim 14, wherein the plurality of domain tokens are structured into a domain information FST parser that includes at least a first FST path comprising a first set of domain tokens and a second FST path comprising a second set of domain tokens, and wherein to generate the meaning representation, the computer system is further programmed to:

select the first FST path based on a first score of the first FST path and a second score of the second FST path, wherein the first FST path is used to generate the meaning representation.

16. The system of claim 15, wherein the first score is based on a first sum of weights of each domain token among the first set of domain tokens and the second score is based on a second sum of weights of each domain token among the second set of domain tokens, and wherein a given weight for a domain token is based on a level of frequency that the domain token appears in the information domain.

17. The system of claim 16, wherein to retrieve the plurality of domain tokens, the computer system is further programmed to:

initialize an input FST based on the first token and the semantic structure; and

compose the input FST based on the first FST path and the second FST path, wherein the first FST path and the second FST path are integrated with the input FST, and wherein the first FST is selected from the input FST.

18. The system of claim 17, wherein to select the first FST path, the computer system is further programmed to: select a shortest path in the input FST.

19. The system of claim 17, wherein to retrieve the plurality of domain tokens, the computer system is further programmed to:

perform fuzzy or exact matching between domain tokens from the information domain and the first token, wherein the plurality of domain tokens comprises fuzzy or exact matches to the first token.

20. The system of claim 14, wherein the computer system is further programmed to:

identify a second token that is relevant to the first token and the information domain, wherein the second token is not initially included in the natural language input; and

add the second token to the meaning representation.

21. The system of claim 14, wherein the natural language input comprises at least a second token, and wherein the computer system is further programmed to:

determine that the second token is not relevant to the information domain; and

omit the second token from the meaning representation responsive to the determination that the second token is not relevant.

22. The system of claim 14, wherein the computer system is further programmed to:

obtain a phoneme confusion matrix comprising at least two similar sounding words that are disambiguated based on previous training from one or more user utterances; and

disambiguate the first token based on the phoneme confusion matrix.

23. The system of claim 14, wherein the computer system is further programmed to:

obtain one or more dynamic data tokens from a dynamic data source; and

integrate the one or more dynamic data tokens with the plurality of tokens from the information domain, wherein the meaning representation is determined based on the integrated dynamic data tokens.

24. The system of claim 23, wherein the plurality of domain tokens are structured into a domain information FST parser that includes at least a first FST path comprising a first set of domain tokens and a second FST path comprising a second set of domain tokens, and wherein to integrate the one or more dynamic data tokens, the computer system is further programmed to:

generate a dynamic FST based on the one or more dynamic data tokens; and

insert the dynamic FST into a slot of the domain information FST parser reserved for dynamic data.

25. The system of claim 14, wherein the computer executable action comprises an execution of: a natural language-based search request or a natural language-based command.

26. The system of claim 14, wherein the information domain comprises a plurality of entries of searchable information, and wherein to retrieve the plurality of domain tokens that match the first token, the computer system is further programmed to:

determine at least one entry, which includes the plurality of domain tokens, that is likely being searched for based on the first token.

27. A method of generating an information domain FST parser for integrating domain information with semantic parsing to generate a meaning representation, the method being implemented on a computer system having one or more physical processors programmed with computer program instructions to perform the method, the method comprising:

obtaining, by the computer system, a plurality of entries of an information domain, wherein each entry comprises one or more domain tokens;

determining, by the computer system, a weight for each domain token, irrespective of whether a given domain token appears in more than one entry, wherein the weight indicates a level of frequency in which the given token appears in the information domain;

defining, by the computer system, for a given entry, a plurality of combinations of the one or more domain tokens, wherein each combination of the one or more domain tokens

represent a different order of the one or more domain tokens and include some or all of the one or more domain tokens for the given entry;

determining, by the computer system, a score for each combination, wherein the score is based on the weight for each domain token involved in the combination; and

storing, by the computer system, the plurality of combinations for each of the plurality of entries.

28. The method of claim 27, wherein the plurality of combinations for each of the plurality of entries are collectively stored as an information domain FST parser, and wherein each combination represents an FST path of the information domain FST parser.

29. The method of claim 28, wherein the score for each combination comprises a path score for each FST path.

30. The method of claim 28, the method further comprising: optimizing the information domain FST parser.

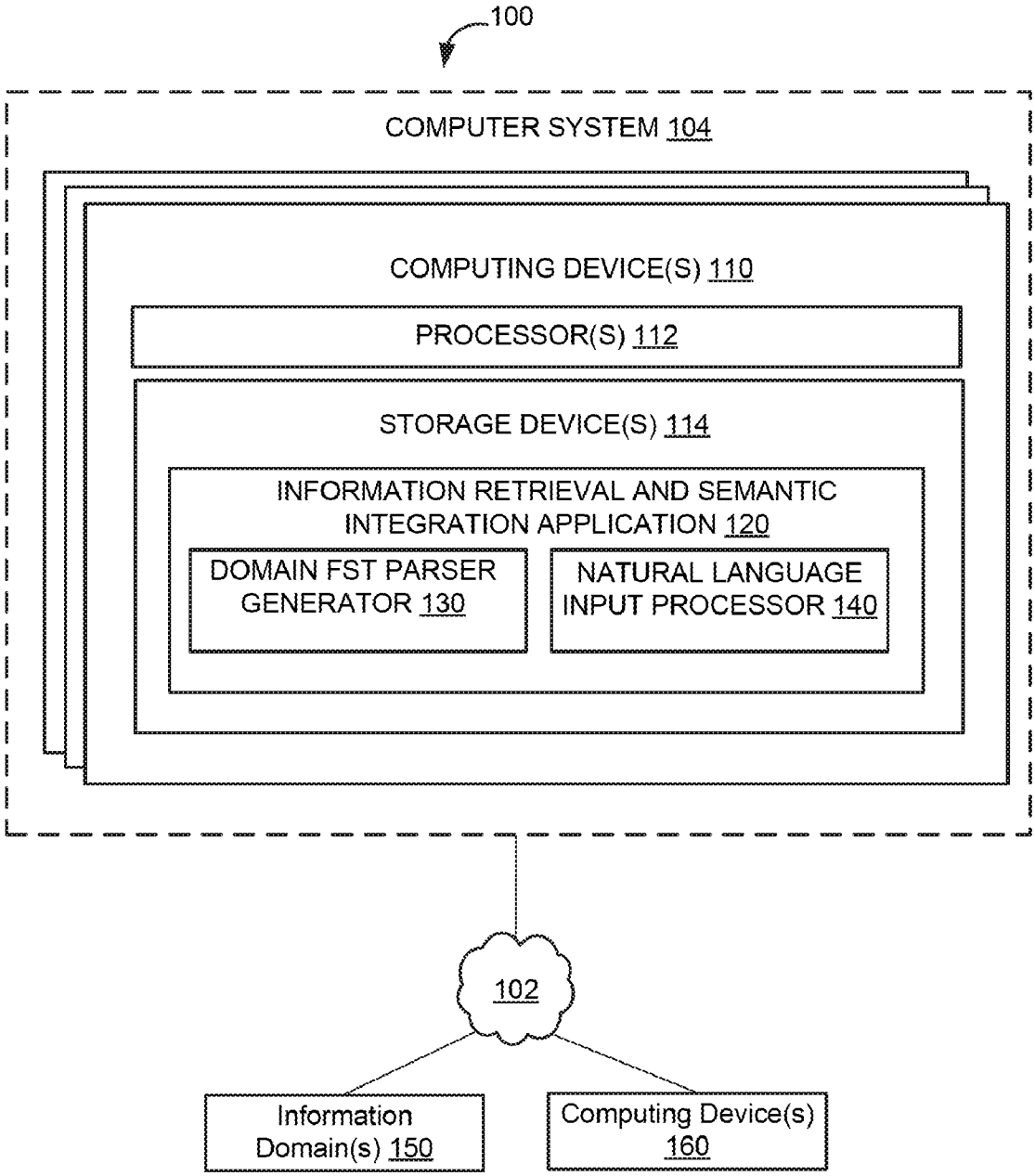
100
COMPUTER SYSTEM 104
COMPUTING DEVICE(S) 110
PROCESSOR(S) 112
STORAGE DEVICE(S) 114
INFORMATION RETRIEVAL AND SEMANTIC INTEGRATION APPLICATION 120
DOMAIN FST PARSER GENERATOR 130
NATURAL LANGUAGE INPUT PROCESSOR 140
102
Information Domain(s) 150
Computing Device(s) 160

FIG. 1

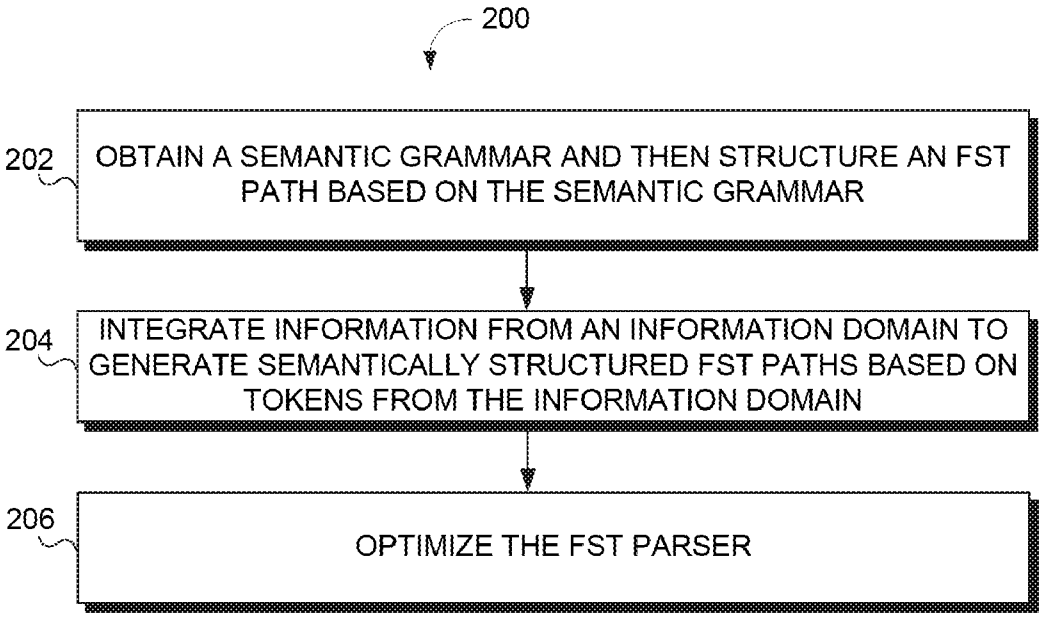
200
202
OBTAIN A SEMANTIC GRAMMAR AND THEN STRUCTURE AN FST PATH BASED ON THE SEMANTIC GRAMMAR
204
INTEGRATE INFORMATION FROM AN INFORMATION DOMAIN TO GENERATE SEMANTICALLY STRUCTURED FST PATHS BASED ON TOKENS FROM THE INFORMATION DOMAIN
206
OPTIMIZE THE FST PARSER

FIG. 2

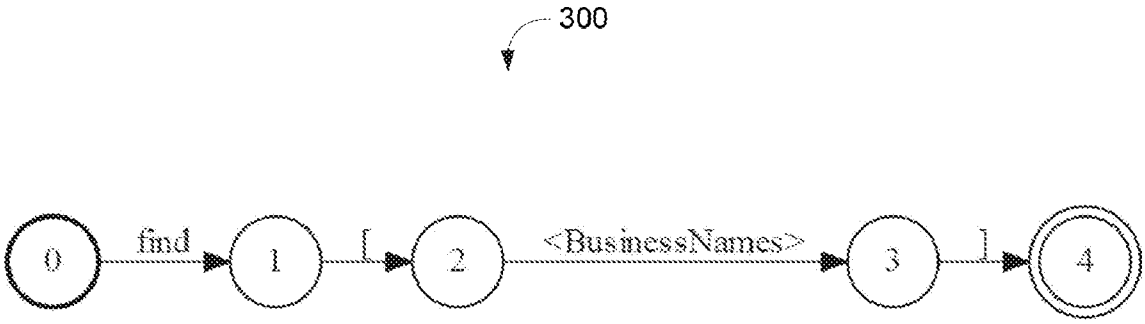
300
0
find
1
[
2
<BusinessNames>
3
]
4

FIG. 3

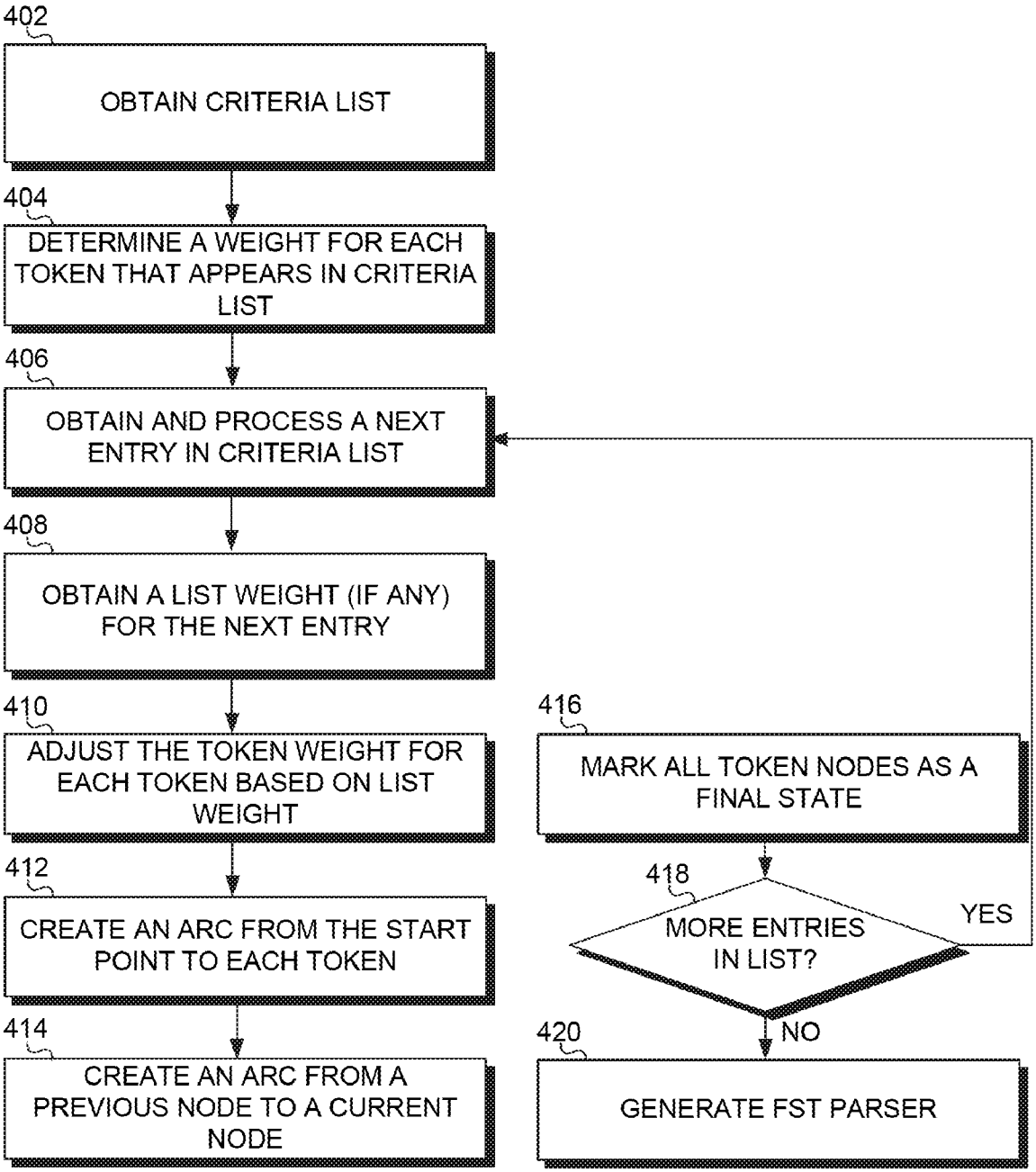
402
OBTAIN CRITERIA LIST
404
DETERMINE A WEIGHT FOR EACH TOKEN THAT APPEARS IN CRITERIA LIST
406
OBTAIN AND PROCESS A NEXT ENTRY IN CRITERIA LIST
408
OBTAIN A LIST WEIGHT (IF ANY) FOR THE NEXT ENTRY
410
ADJUST THE TOKEN WEIGHT FOR EACH TOKEN BASED ON LIST WEIGHT
412
CREATE AN ARC FROM THE START POINT TO EACH TOKEN
414
CREATE AN ARC FROM A PREVIOUS NODE TO A CURRENT NODE
416
MARK ALL TOKEN NODES AS A FINAL STATE
418
MORE ENTRIES IN LIST?
YES
NO
420
GENERATE FST PARSER

FIG. 4

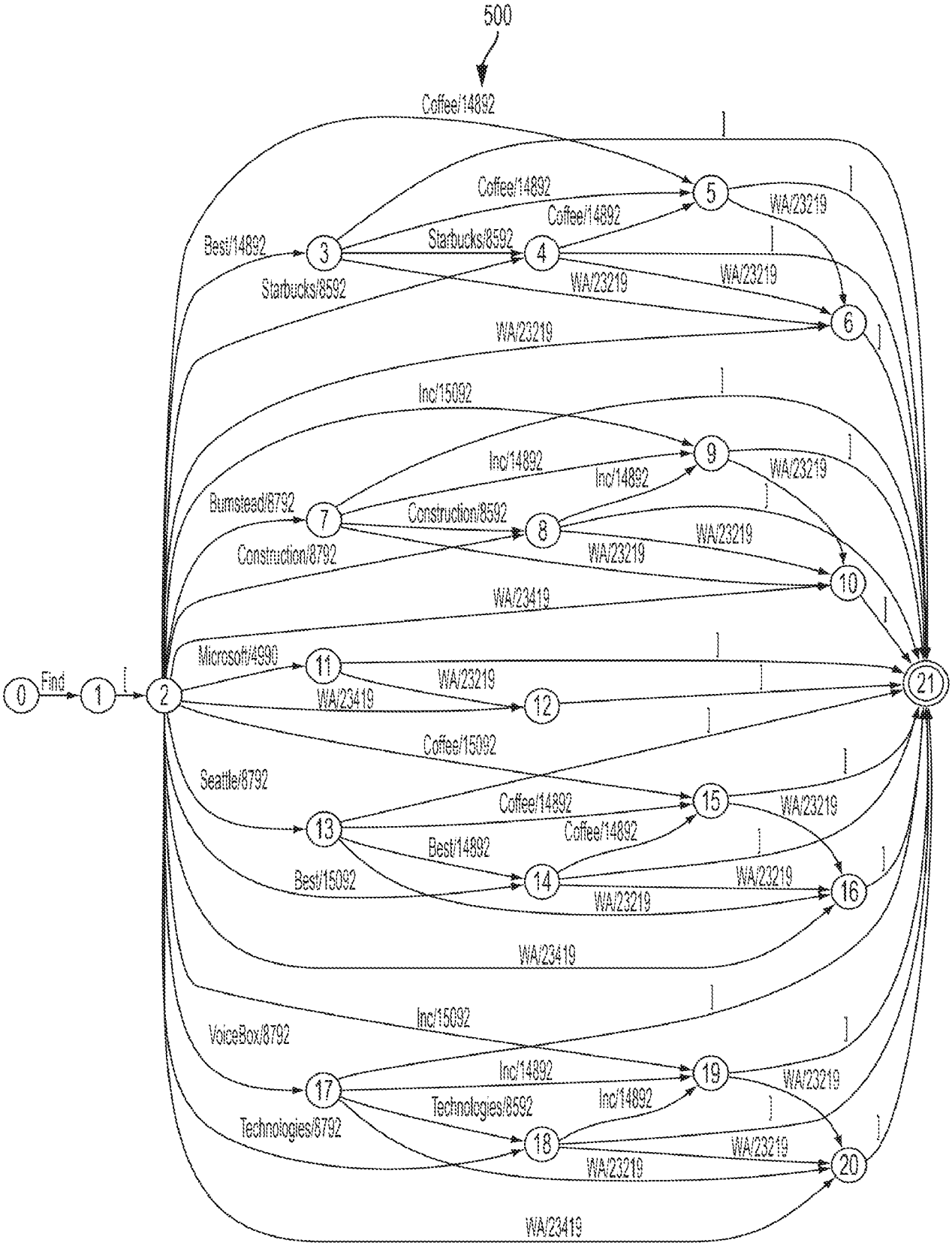
500
Coffee/14892
Coffee/14892
Coffee/14892
Best/14892
Starbucks/8592
Starbucks/8592
WA/23219
WA/23219
WA/23219
WA/23219
Inc/15092
Inc/14892
Inc/14892
WA/23219
Burnstead/8792
Construction/8592
Construction/8792
WA/23219
WA/23219
WA/23419
Microsoft/4990
WA/23219
WA/23419
Find
Coffee/15092
Seattle/8792
Coffee/14892
Coffee/14892
WA/23219
Best/14892
Best/15092
WA/23219
WA/23219
WA/23419
Inc/15092
VoiceBox/8792
Inc/14892
Inc/14892
WA/23219
Technologies/8592
Technologies/8792
WA/23219
WA/23219
WA/23419
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21

FIG. 5

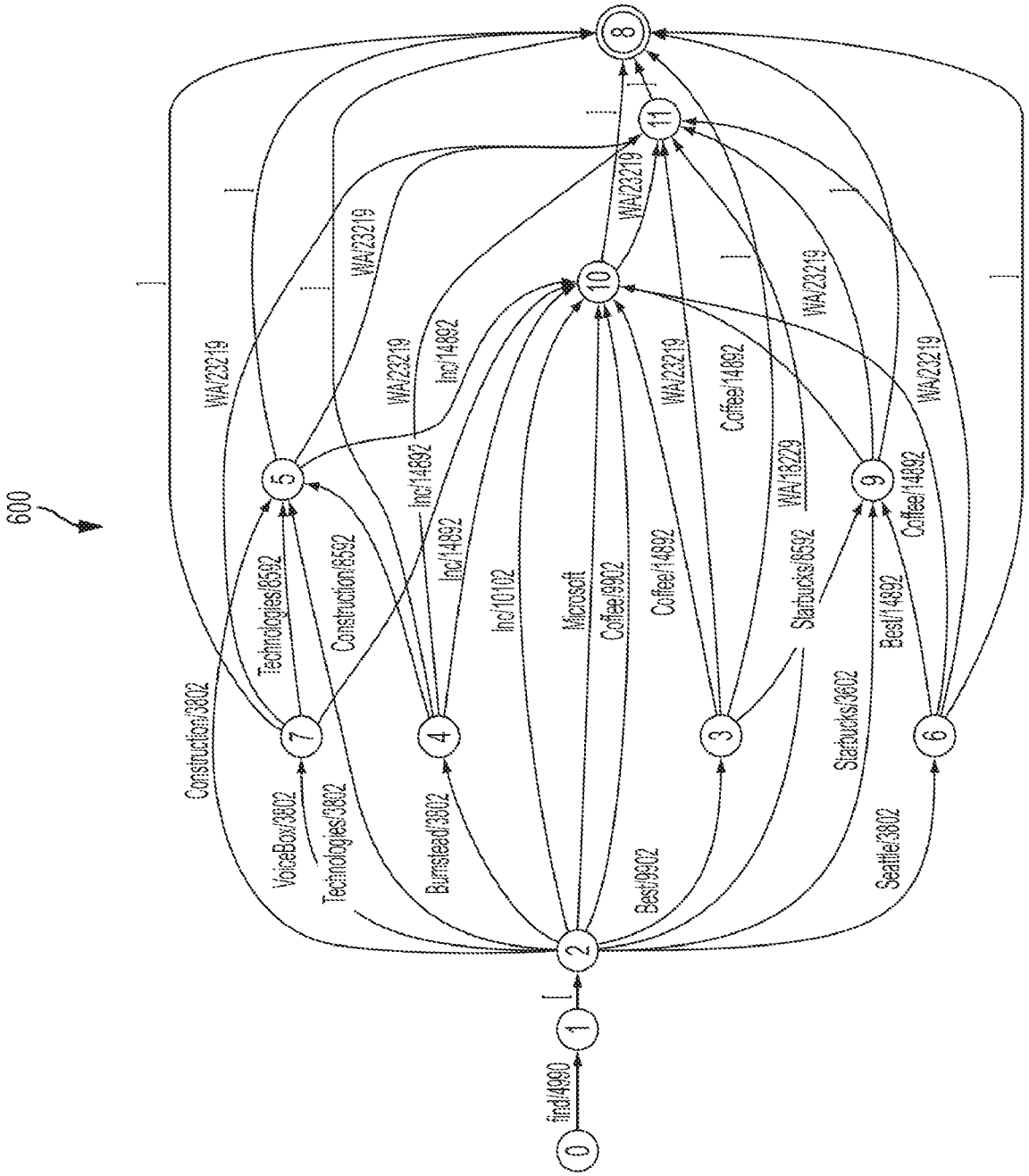
600
find/4980
0
1
2
Best/98032
Seattle/3802
Starbucks/98032
Best/14892
Coffee/14892
3
Starbucks/8592
WA/18229
Coffee/14892
4
Burnstead/3802
Inc/14892
Inc/101102
Microsoft
Coffee/99012
Coffee/14892
5
VoiceBox/3802
Technologies/3812
Technologies/8592
Construction/8592
Construction/3802
6
7
8
9
10
11
WA/23219

FIG. 6

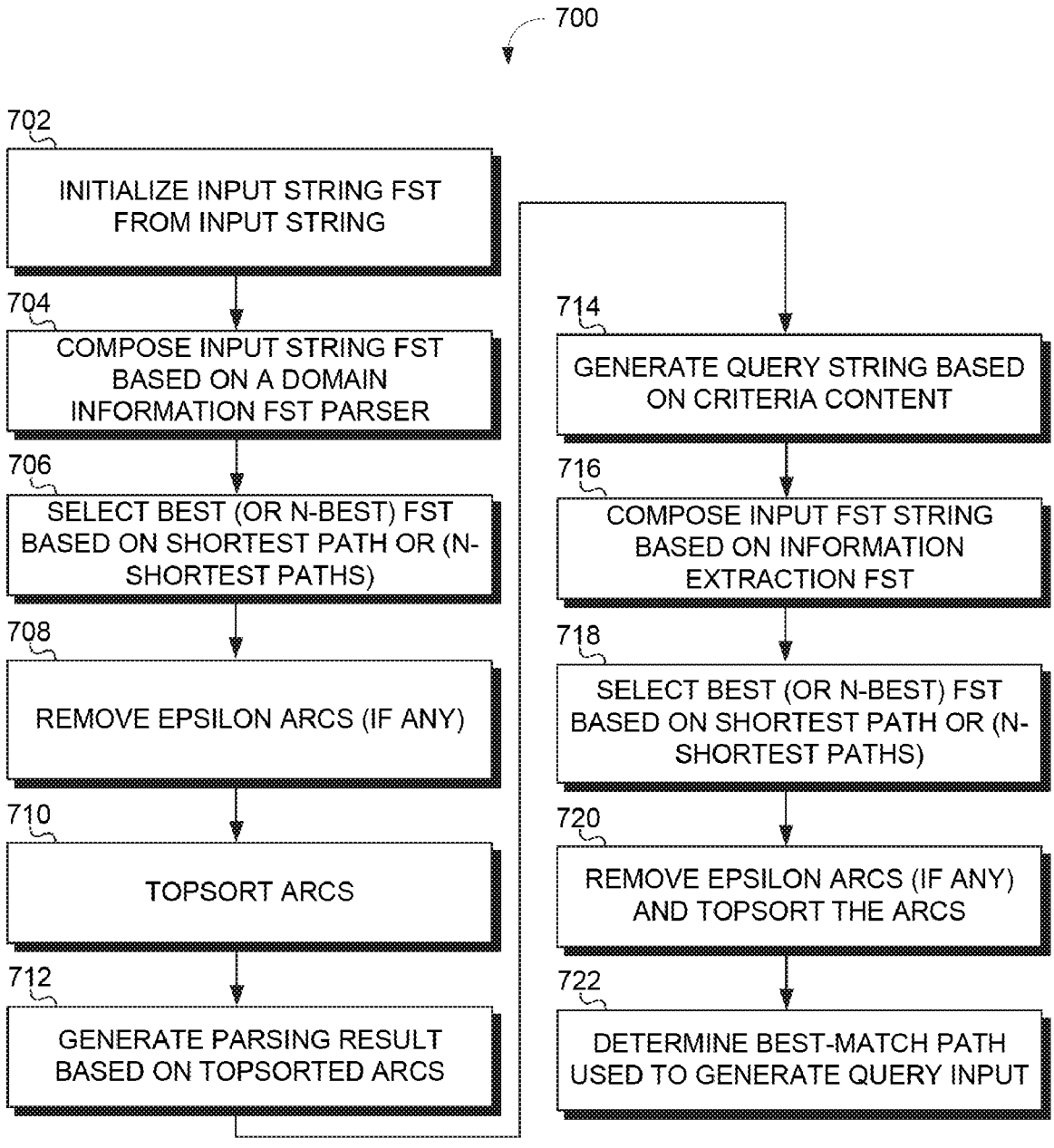
700
702
INITIALIZE INPUT STRING FST FROM INPUT STRING
704
COMPOSE INPUT STRING FST BASED ON A DOMAIN INFORMATION FST PARSER
706
SELECT BEST (OR N-BEST) FST BASED ON SHORTEST PATH OR (N-SHORTEST PATHS)
708
REMOVE EPSILON ARCS (IF ANY)
710
TOPSORT ARCS
712
GENERATE PARSING RESULT BASED ON TOPSORTED ARCS
714
GENERATE QUERY STRING BASED ON CRITERIA CONTENT
716
COMPOSE INPUT FST STRING BASED ON INFORMATION EXTRACTION FST
718
SELECT BEST (OR N-BEST) FST BASED ON SHORTEST PATH OR (N-SHORTEST PATHS)
720
REMOVE EPSILON ARCS (IF ANY) AND TOPSORT THE ARCS
722
DETERMINE BEST-MATCH PATH USED TO GENERATE QUERY INPUT

FIG. 7

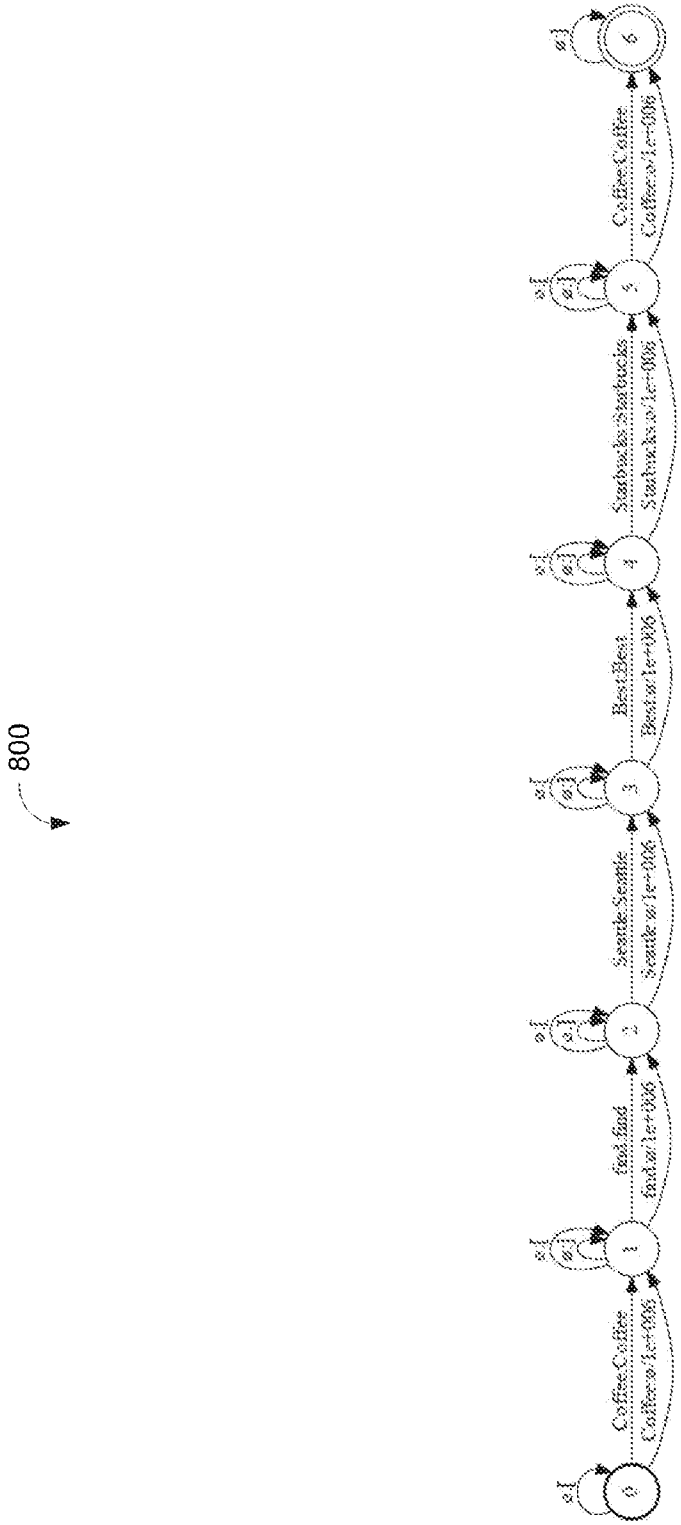
800
0
1
2
3
4
5
6
Coffee:Coffee
Coffee:w/1e+006
find:find
find:w/1e+006
Seattle:Seattle
Seattle:w/1e+006
Best:Best
Best:w/1e+006
Starbucks:Starbucks
Starbucks:w/1e+006
Coffee:Coffee
Coffee:w/1e+006

FIG. 8

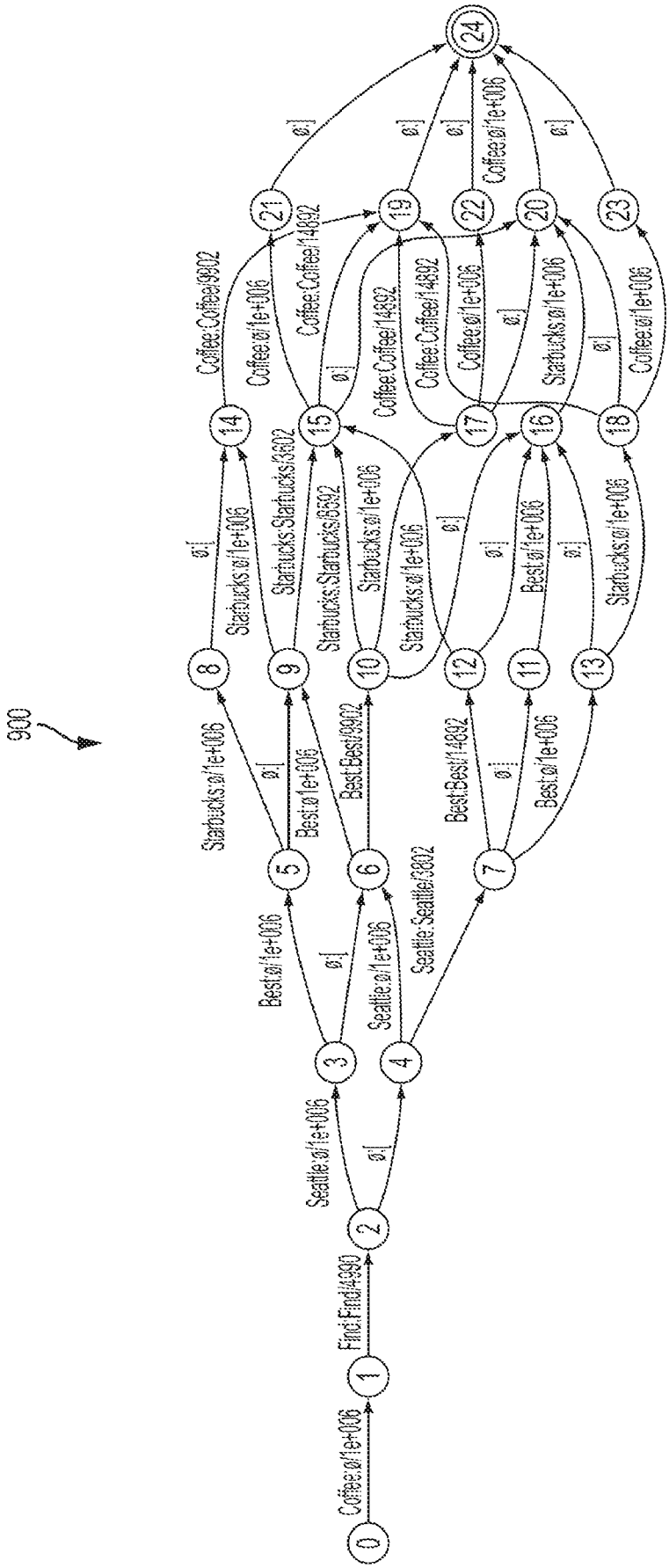
900
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
Coffee:ø/1e+006
Find:Find/4990
Seattle:ø/1e+006
Seattle:ø/1e+006
Seattle:Seattle/3802
Best:ø/1e+006
Best:Best/9902
Best:Best/14892
Best:ø/1e+006
Starbucks:ø/1e+006
Starbucks:Starbucks/6592
Starbucks:Starbucks/3802
Starbucks:ø/1e+006
Starbucks:ø/1e+006
Coffee:ø/1e+006
Coffee:Coffee/14892
Coffee:Coffee/14892
Coffee:ø/1e+006
Coffee:Coffee/9902
Coffee:ø/1e+006
Starbucks:ø/1e+006
Coffee:ø/1e+006
Coffee:ø/1e+006
ø|

FIG. 9

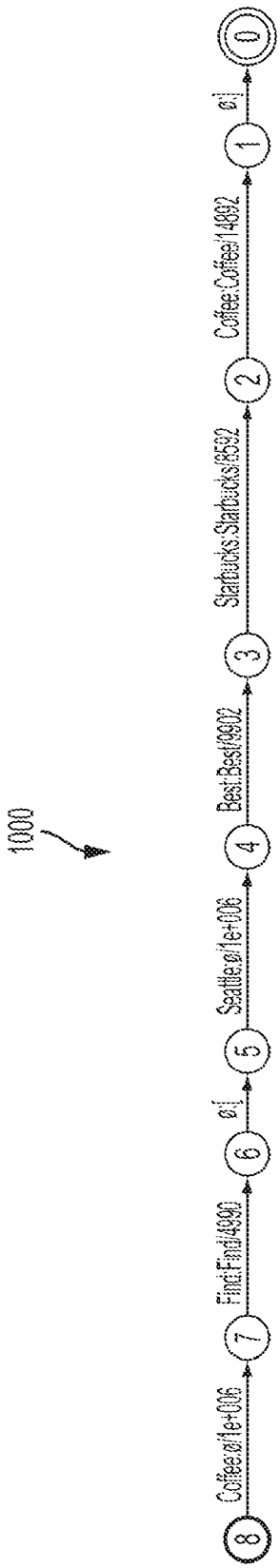
1000
8
Coffee:ø/1e+006
7
Find:Find/4990
6
ø1
5
Seattle:ø/1e+006
4
Best:Best/9902
3
Starbucks:Starbucks/8592
2
Coffee:Coffee/14892
1
ø1
0

FIG. 10

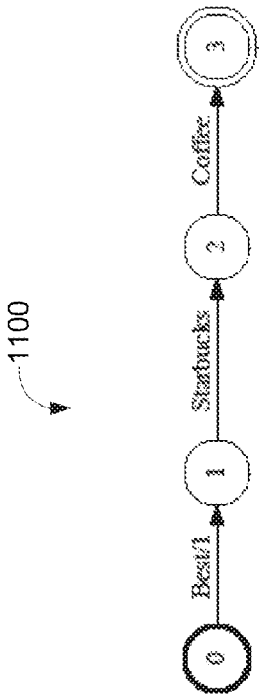

FIG. 11

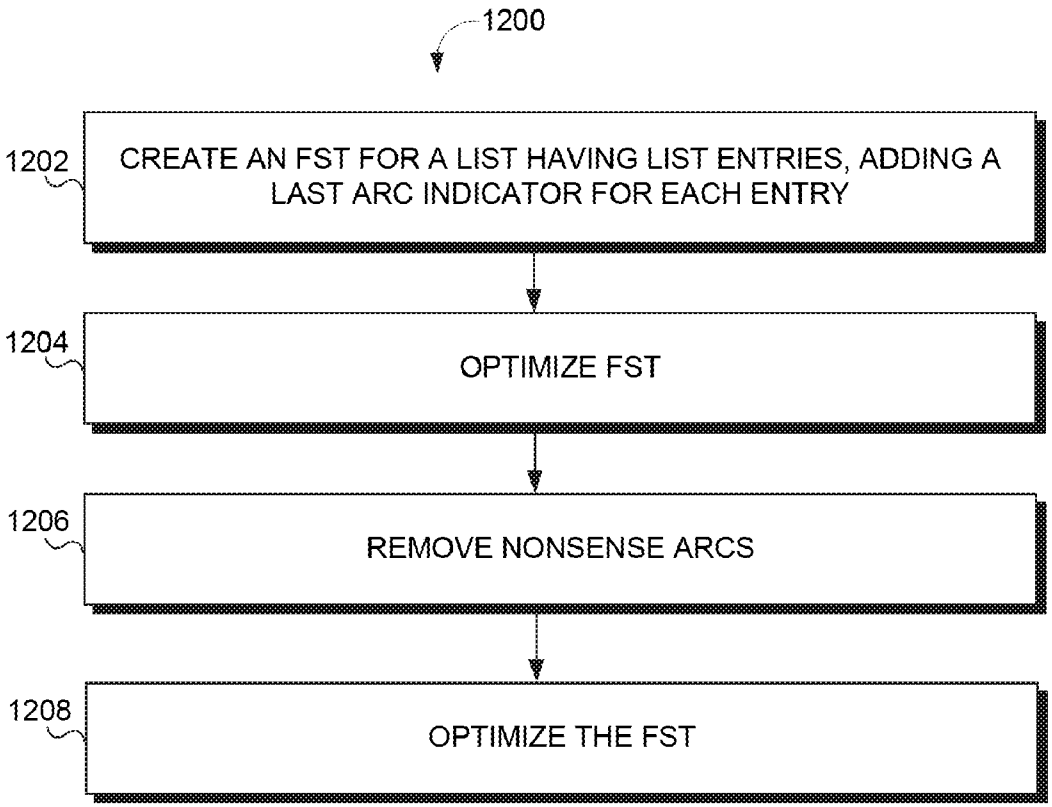
1200
1202
CREATE AN FST FOR A LIST HAVING LIST ENTRIES, ADDING A LAST ARC INDICATOR FOR EACH ENTRY
1204
OPTIMIZE FST
1206
REMOVE NONSENSE ARCS
1208
OPTIMIZE THE FST

FIG. 12

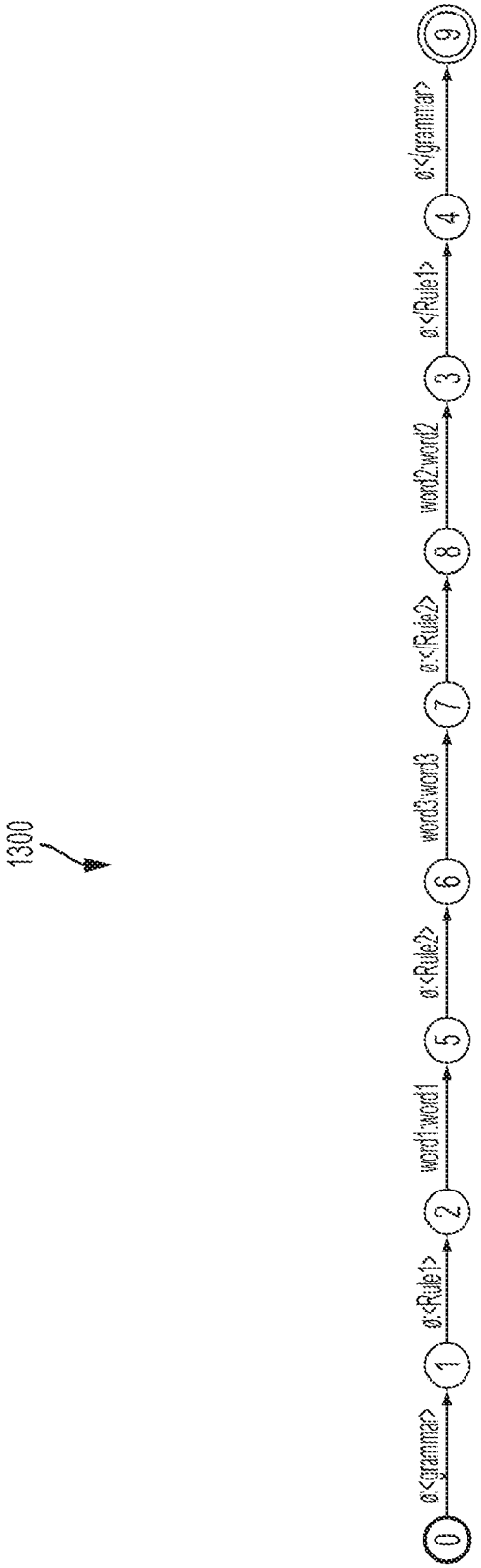
1300
0
ε:<grammar>
1
ε:<Rule1>
2
word1:word1
5
ε:<Rule2>
6
word3:word3
7
ε:</Rule2>
8
word2:word2
3
ε:</Rule1>
4
ε:</grammar>
9

FIG. 13

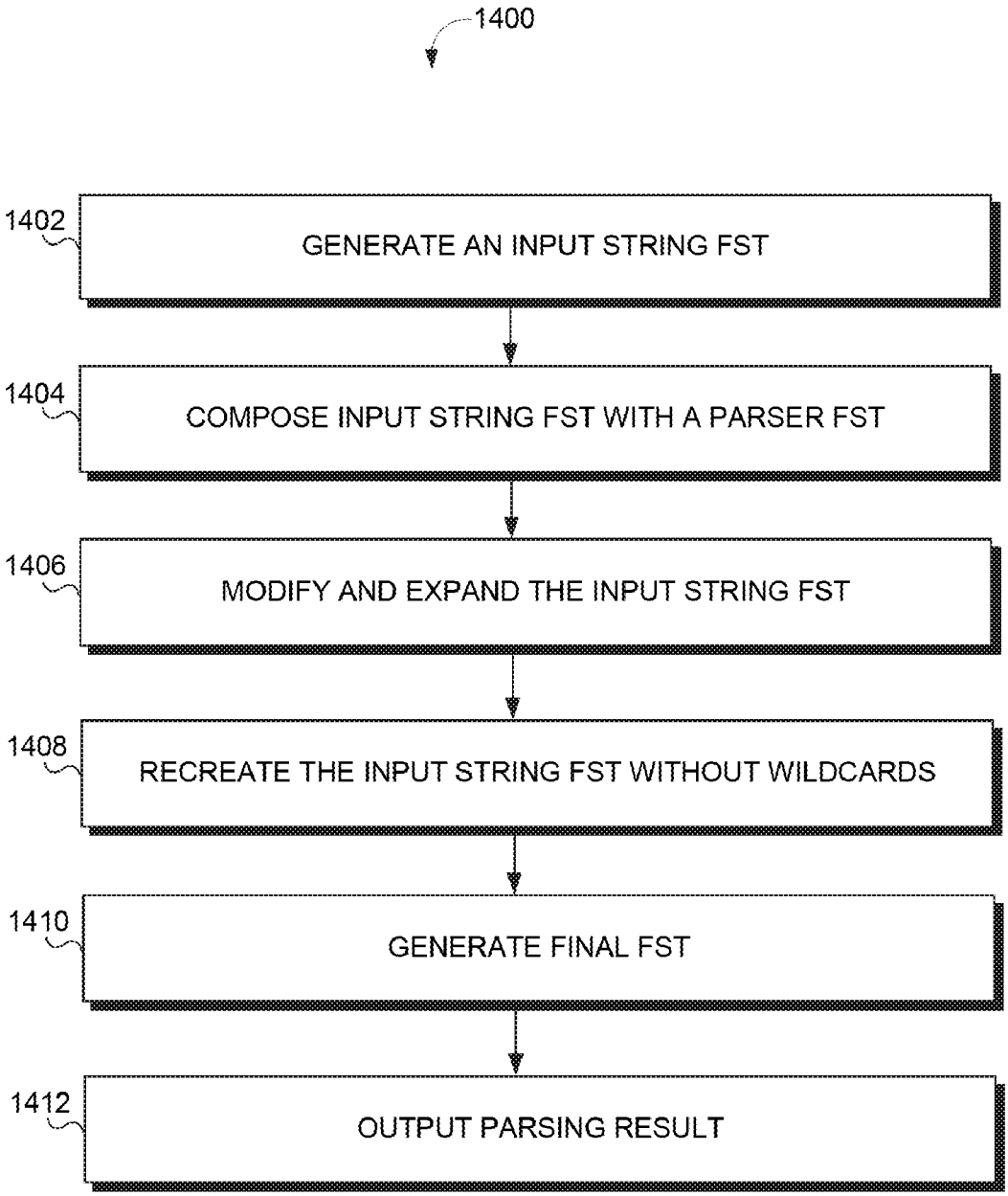
1400
1402
GENERATE AN INPUT STRING FST
1404
COMPOSE INPUT STRING FST WITH A PARSER FST
1406
MODIFY AND EXPAND THE INPUT STRING FST
1408
RECREATE THE INPUT STRING FST WITHOUT WILDCARDS
1410
GENERATE FINAL FST
1412
OUTPUT PARSING RESULT

FIG. 14

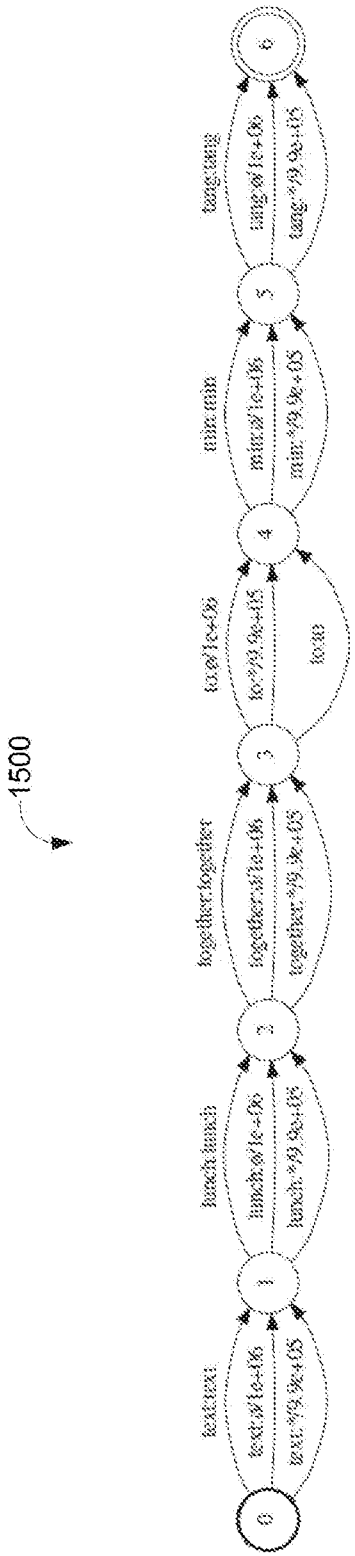
1500
0
1
2
3
4
5
6
text:text
text:*99.9e+05
text:1e+06
lunch:lunch
lunch:*99.9e+05
lunch:1e+06
together:together
together:*99.9e+05
together:1e+06
to:to
to:*99.9e+05
to:1e+06
min:min
min:*99.9e+05
min:1e+06
tang:tang
tang:*99.9e+05
tang:1e+06

FIG. 15

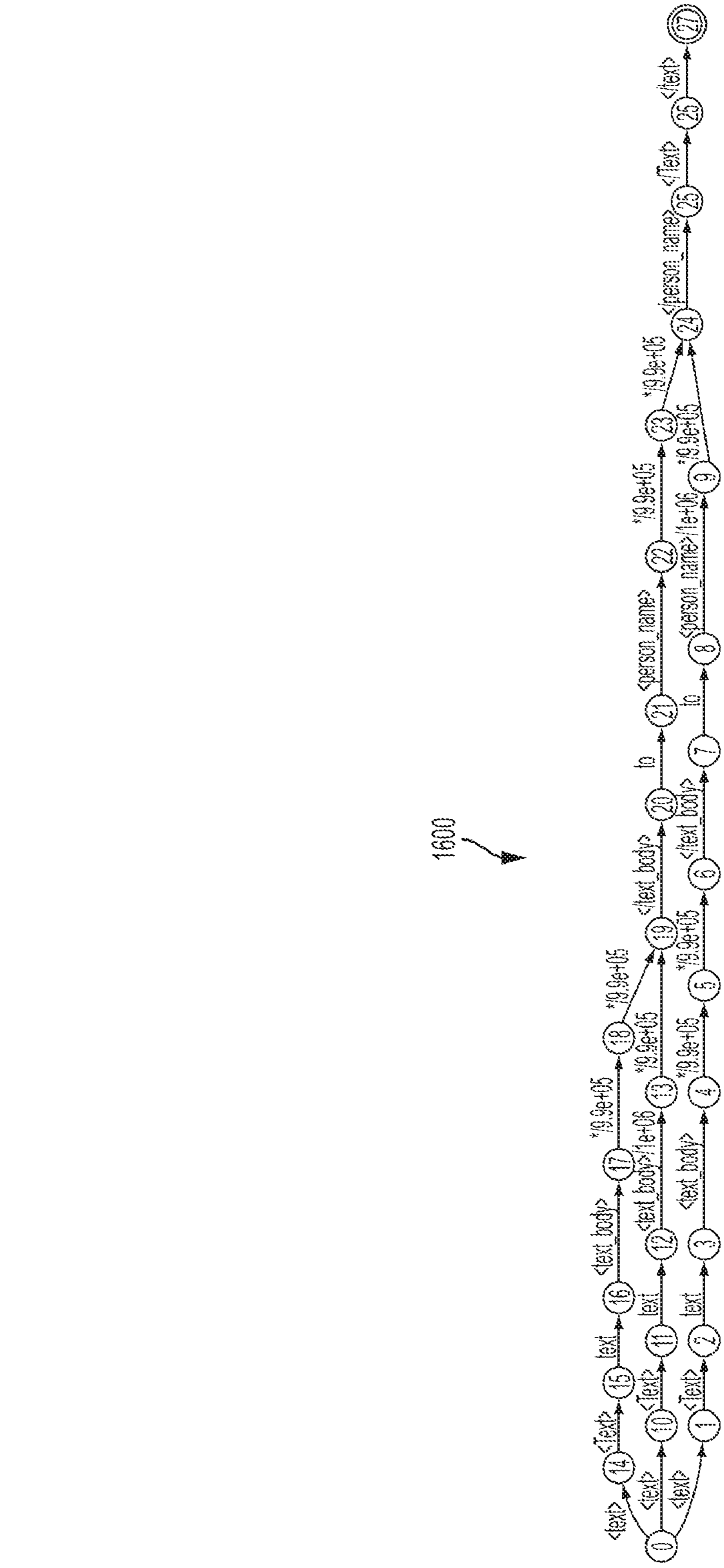

FIG. 16

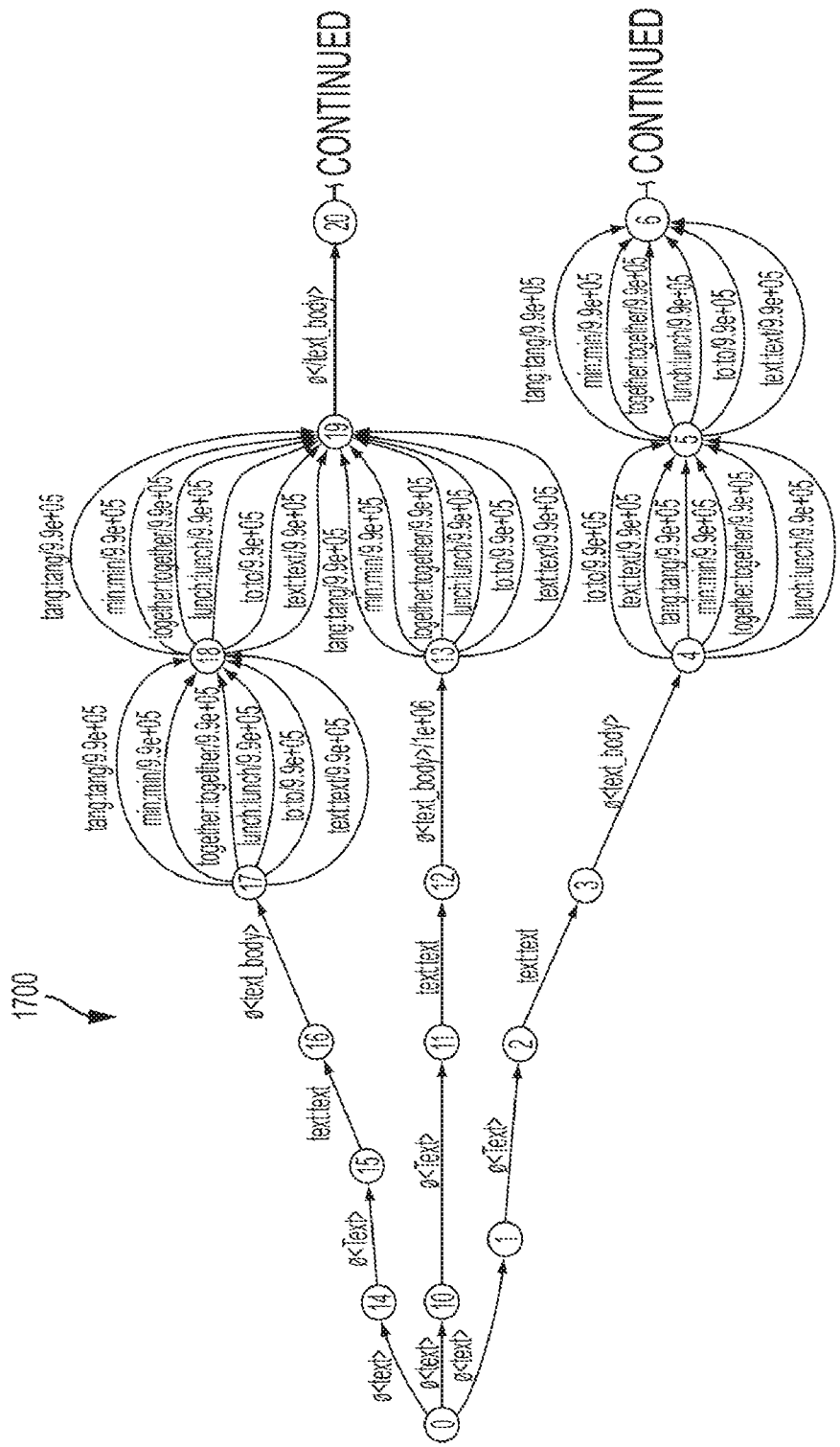
1700
0
1
2
3
4
5
6
10
11
12
13
14
15
16
17
18
19
20
ø<text>
ø<Text>
text:text
ø<text_body>
ø<text_body>/1e+06
lang:lang/9.9e+05
min:min/9.9e+05
together:together/9.9e+05
lunch:lunch/9.9e+05
to:to/9.9e+05
text:text/9.9e+05
CONTINUED

FIG. 17

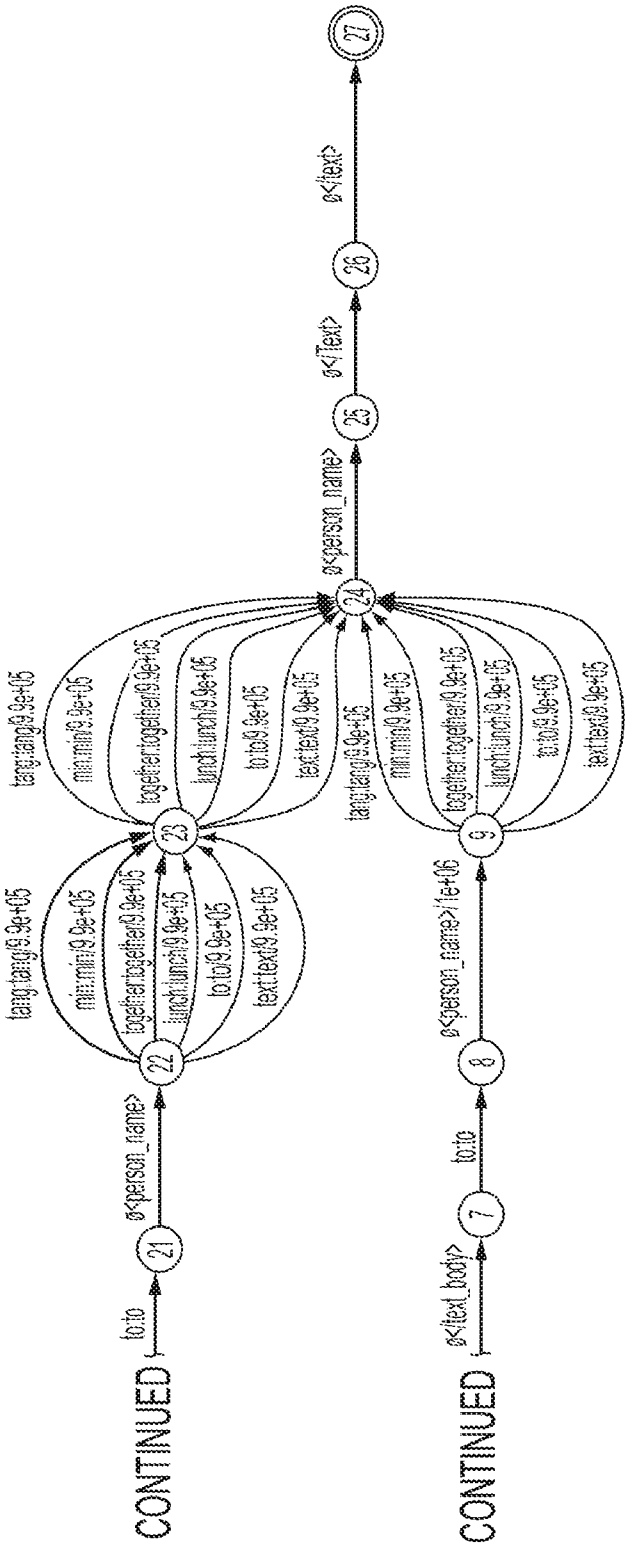

FIG. 17
CONTINUED

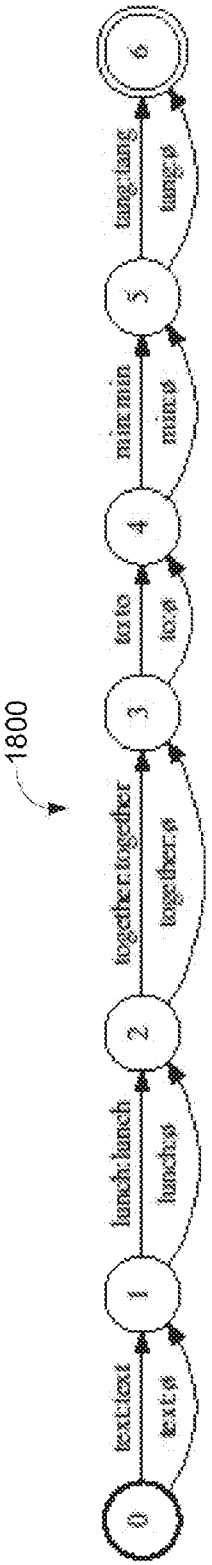
1800
0
1
2
3
4
5
6
text:text
exit:ε
lunch:lunch
lunch:ε
together:together
together:ε
into:into
into:ε
mine:mine
mine:ε
tag:tag
tag:ε

FIG. 18

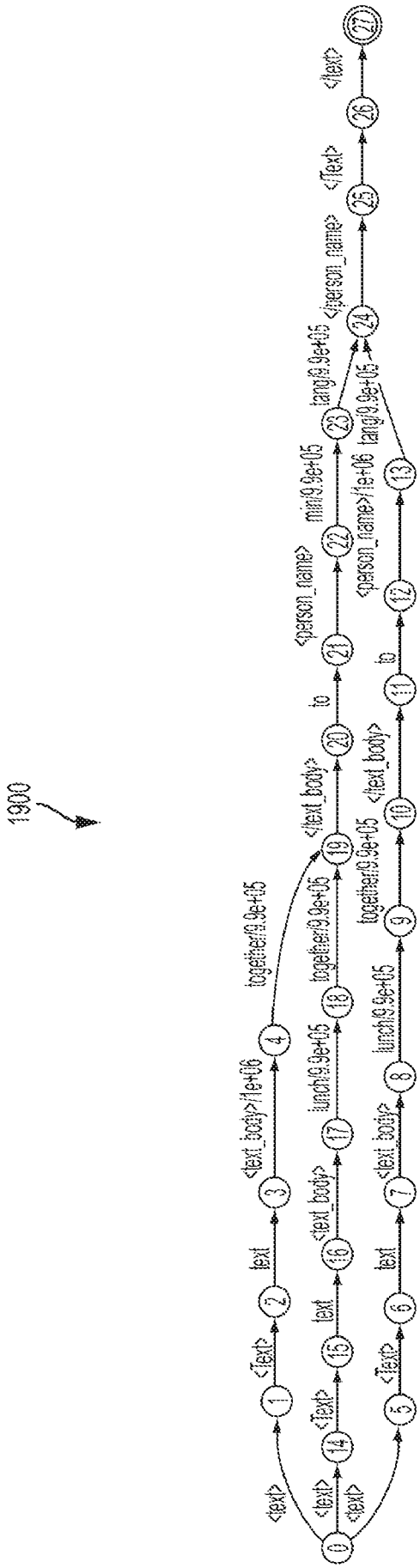
1900
0
<text>
1
<Text>
2
text
3
<text_body>/1e+06
4
together/0.9e+05
<text_body>
19
to
20
<person_name>
21
<person_name>
22
min/0.9e+05
23
tang/0.9e+05
<person_name>
24
</person_name>
25
</Text>
26
</text>
27
<text>
14
<Text>
15
text
16
<text_body>
17
lunch/0.9e+05
18
together/0.9e+05
<text_body>
10
<text_body>
11
to
12
<person_name>/1e+06
tang/0.9e+05
13
<text>
5
<Text>
6
text
7
<text_body>
8
lunch/0.9e+05
together/0.9e+05
9

FIG. 19

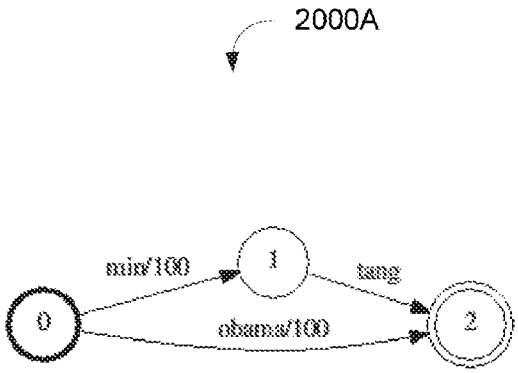

FIG. 20A

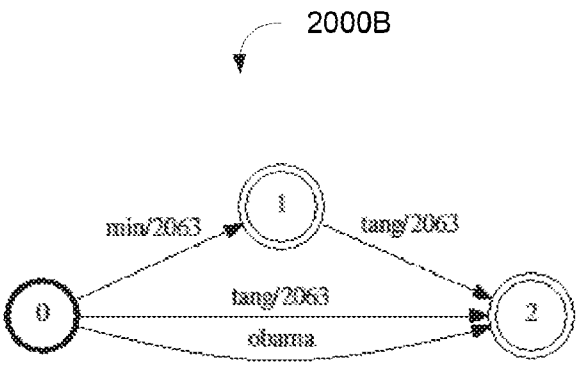

FIG. 20B

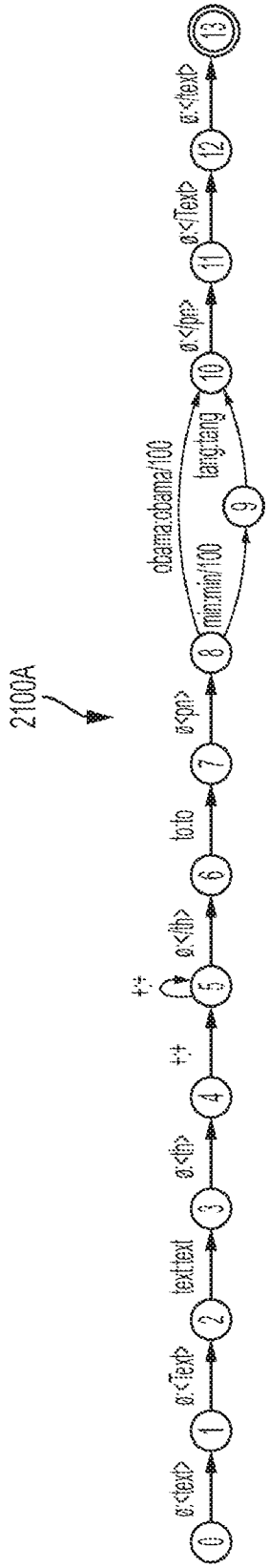

FIG. 21 A

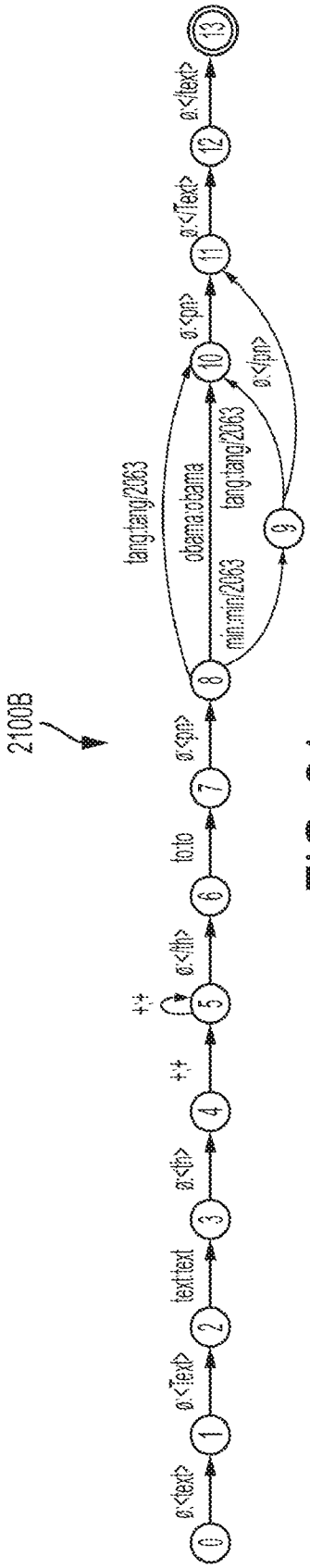

FIG. 21 B

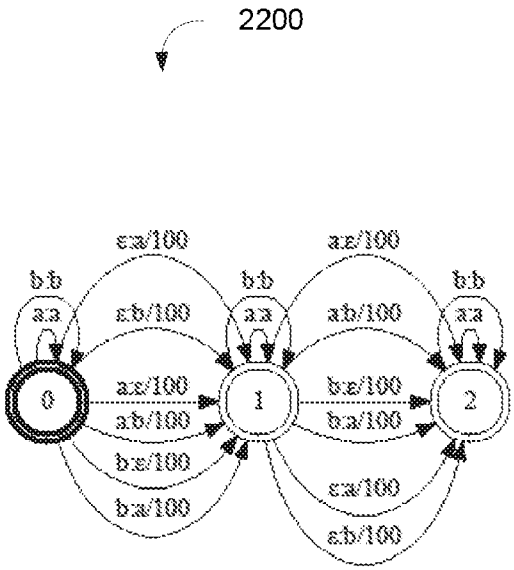
2200
ε:a/100
a:ε/100
b:b
a:a
ε:b/100
a:b/100
0
1
2
a:ε/100
a:b/100
b:ε/100
b:a/100
b:ε/100
b:a/100
ε:a/100
ε:b/100

FIG. 22

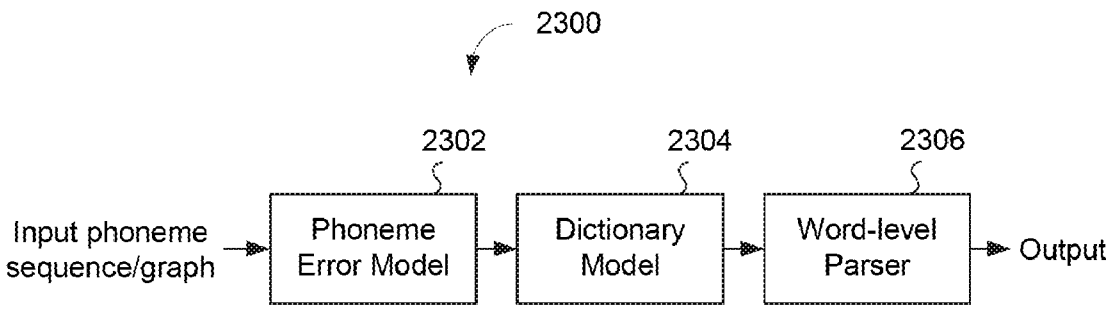
2300
2302
2304
2306
Input phoneme sequence/graph
Phoneme Error Model
Dictionary Model
Word-level Parser
Output

FIG. 23

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2015/050263

A. CLASSIFICATION OF SUBJECT MATTER
IPC(8) - G06F 17/27 (2015.01)
CPC - G06F 17/271 (2015.10)
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)
IPC(8) - G06F 17/27, G06F 17/28, G06F 17/30 (2015.01)
USPC - 1/1, 704/9, 707/999.006, 707/E17.014

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
CPC - G06F 17/271, G06F 17/2785, G06F 17/28, G06F 17/3043, G06F 17/3061 (2015.10) (keyword delimited)

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
Orbit, Google Patents, Google Scholar, Google.
Search terms used: integrate domain information, semantic parsing, generate meaning representations from natural language input, processor

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 2008/0104071 A1 (PRAGADA et al) 01 May 2008 (01.05.2008), entire document	1-30
Y	US 2014/0236575 A1 (MICROSOFT CORPORATION) 21 August 2014 (21.08.2014), entire document	1-30
Y	US 2011/0119049 A1 (YLONEN) 19 May 2011 (19.05.2011), entire document	1-26
Y	US 2003/0187643 A1 (VAN THONG et al) 02 October 2003 (02.10.2003), entire document	9, 22
Y	US 6,816,830 B1 (KEMPE) 09 November 2004 (09.11.2004), entire document	2-6, 11, 15-19, 24, 27-30
Y	US 2008/0294437 A1 (NAKANO et al) 27 November 2008 (27.11.2008), entire document	3, 11, 16-19, 24, 27-30
Y	US 2012/0046935 A1 (NAGAO) 23 February 2012 (23.02.2012), entire document	2-6, 15-19, 29
Y	US 6,073,098 A (BUCHSBAUM et al) 06 June 2000 (06.06.2000), entire document	5, 18
A	US 2009/0248605 A1 (MITCHELL et al) 01 October 2009 (01.10.2009), entire document	1-30
A	US 2007/0112555 A1 (LAVI et al) 17 May 2007 (17.05.2007), entire document	1-30
A	US 5,265,065 A (TURTLE) 23 November 1993 (23.11.1993), entire document	1-30
A	US 5,794,050 A (DAHLGREN et al) 11 August 1998 (11.08.1998), entire document	1-30

☐ 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 another 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	Date of mailing of the international search report
31 October 2015	20 JAN 2016
Name and mailing address of the ISA/ Mail Stop PCT, Attn: ISA/US, Commissioner for Patents P.O. Box 1450, Alexandria, Virginia 22313-1450 Facsimile No. 571-273-8300	Authorized officer Blaine Copenheaver PCT Helpdesk: 571-272-4300 PCT OSP: 571-272-7774

Form PCT/ISA/210 (second sheet) (January 2015)