А.В. ЧЕУСОВ

**ЛЕКСИЧЕСКИЙ АНАЛИЗ ТЕКСТА
В ПРОМЫШЛЕННЫХ СИСТЕМАХ ЕГО ОБРАБОТКИ**

An approach to lexical analysis of the natural language processing systems is suggested.

Актуальность систем обработки естественного языка

Естественный язык (ЕЯ) является универсальным средством описания действительности и коммуникации с вычислительной системой, и в условиях современной информационной индустрии возрастает его роль как основного средства ее интеллектуализации.

Эффективное решение задач автоматизации документооборота, информационного поиска, инженерии знаний и подобные проблемы требуют разработки «развитых» лингвистических процессоров (ЛП), реализующих следующие основные этапы обработки произвольного текста ЕЯ, производя его лексический, лексико-грамматический, синтаксический, семантический анализ.

Основными характеристиками таких ЛП должны стать:

- 1) способность «работать» с произвольным текстом ЕЯ;
- 2) высокая точность и скорость обработки текста каждым функциональным модулем ЛП;
- 3) наличие лингвистической базы знаний (ЛБЗ), адекватно отображающей «основное лингвистическое поведение» текста на всех уровнях глубины ЕЯ;
- 4) модульность архитектуры ЛБЗ;
- 5) обеспечение пользователю доступа к ЛБЗ в конкретных приложениях.

Придание лингвистическому процессору таких характеристик требует решения целого ряда математических, алгоритмических и лингвистических проблем, а также задач программирования и тестирования как ЛП в целом, так и его ЛБЗ в частности.

Нам удалось построить промышленный ЛП, в значительной степени удовлетворяющий требованиям 1) - 5) на всех этапах обработки входного текста ЕЯ. Так, в основу разработки первого функционального модуля (лексического анализа) были положены следующие решения.

Лексический анализ

Начальной задачей автоматического анализа текста является распознавание в нем границ слов, т. е. преобразование входного текста в список слов, знаков препинания, формул, телефонных номеров, дат и т. п. Этот процесс также называют лексическим анализом текста.

В языках программирования (ЯП) для решения этой задачи часто применяют инструмент *flex* [1], основанный на конечных автоматах (finite state automaton, или FSA [2, 3]). Однако такой подход неэффективен для анализа естественного языка, поскольку типов лексем в ЕЯ значительно больше, чем в ЯП. Возможное отсутствие необходимого шаблона приведет к неправильному разбиению токена (под токеном понимается последовательность символов входного текста между двумя соседними пробелами) на слова. В то же время во многих практических задачах нет необходимости в разбиении сложных токенов на подтокены. Например, математические и химические формулы удобнее рассматривать как одно слово. Наконец, при разработке ЕЯ-систем в целях облегчения тестирования их функциональных модулей желательно иметь возможность создавать соответствующие правила постепенно, начиная с простых и наиболее часто встречающихся случаев и заканчивая более сложными. То же относится и к модулю лексического анализа. Имеет смысл начинать обработку с наиболее простых токенов, оставляя без изменения сложные. И затем по мере необходимости обрабатывать все более и более сложные. С одной стороны, такая техника позволяет быстрее достичь приемлемого результата, а с другой - дает возможность легко выделить шаблоны токенов, для которых правила разбора еще не описаны, что, в свою очередь, позволяет оценить вероятность возможных ошибок для различных предметных областей, представленных соответствующим корпусом текстов [4]. Алгоритмы, основанные на FSA, такой возможности не предоставляют. Этого недостатка не лишены и системы, основанные на взвешенных конечных автоматах (weighted finite state automaton, или WFSA [5]), использующиеся, например, в [6]. Особенно остро проблема сложных токенов стоит при анализе документов из предметных областей, не представленных в тренировочном корпусе (training corpus [4]).

Некоторые исследователи [7, 8] для лексического анализа ЕЯ используют конечные преобразователи (finite state transducers, или FST) [9], которые лише-

ны описанного выше недостатка. Однако при создании правил в виде FST для анализа плохо форматированных документов (например, отсутствие пробела после знака препинания) быстро растет само количество правил, что приводит к сильной недетерминированности результирующего автомата и большому количеству генерируемых вариантов анализа (в некоторых случаях количество таких вариантов растет экспоненциально от длины входного токена), что приводит к замедлению последующих этапов анализа текста.

Нотация записи правил и алгоритм

Для решения поставленной задачи с учетом описанных выше проблем предлагается следующий алгоритм. Для выделения из текста слов разбиваем его на токены. Затем отдельно обрабатываем каждый токен согласно упорядоченному набору правил $R = (R_1, R_2, \dots, R_n)$, где каждое правило - упорядоченный список пар <расширенное регулярное выражение [10], команда>, т. е.

$$\forall i, 1 \leq i \leq n, R_i \equiv ((E_{i,1}, C_{i,1}), (E_{i,2}, C_{i,2}), \dots, (E_{i,M_i}, C_{i,M_i})), \quad (1)$$

где $\forall i, \forall j, 1 \leq i \leq n, 1 \leq j \leq M_i, C_{i,j} \in \{ _rec, _put \}$ - команда обработки части токена, $E_{i,j}$ - регулярное выражение.

Каждый полученный из текста токен обрабатывается следующим образом. Начиная с первого, ищем правило, подходящее токену, т. е. ищем такое правило i , чтобы токен удовлетворял регулярному выражению

$$E_i = \sum_{j=1}^{M_i} E_{i,j}. \quad (2)$$

(Под сложением здесь понимается операция конкатенации.)

Операцию поиска можно выполнить за время, линейно зависящее от длины токена, если преобразовать ERE в детерминированный конечный автомат [2]. Если подходящее правило найдено, то по принципу «самый левый самый длинный» [10] находим части токена, соответствующие компонентам $E_{i,j}$. Эта операция также может быть выполнена за линейное время, в [11] описан один из алгоритмов, реализующий эту функциональность согласно указанной оценке. Затем для каждого выделенного компонента проверяем соответствующую ему команду $C_{i,j}$. Если это команда $_put$, добавляем выделенный фрагмент к результирующему списку слов; если команда $_rec$, то обрабатываем часть токена рекурсивно. Когда токен не удовлетворяет ни одному из правил, он добавляется к результирующему списку слов без изменения. В этом случае можно сказать, что мы обнаружили токен «неизвестного» типа. Такая информация может быть полезна на этапе создания и тестирования правил.

Нетрудно убедиться, что трудоемкость этого алгоритма $O(nk_2)$, где n - количество правил, а k - длина токена. Оценка показывает, что описанный алгоритм недостаточно эффективен для систем, ориентированных на обработку больших объемов информации (именно это характерно для промышленных ЕЯ-систем), и необходимы дальнейшие исследования по его оптимизации.

Оптимизация

Алгоритм можно значительно улучшить, усовершенствовав поиск по заданному токену подходящего для него правила из R . Для этого построим детерминированный автомат Мура [2] $\langle I, O, S, qf, g \rangle$, где I - конечное непустое множество алфавитно-цифровых и специальных символов; $O = \{ \lambda, 1, 2, \dots, n \}$, где n - количество правил; S - конечное непустое множество состояний автомата; $q \in S$ - его начальное состояние; $f: S^* I \rightarrow S$ - функция переходов автомата; $g: S \rightarrow O$ - функция его выходов; * - операция декартова произведения. Функция g отображает состояние автомата в минимальный номер правила, которому удовлетворяет входная анализируемая цепочка символов, или в λ , если цепочка не удов-

летворяет ни одному из правил. Таким образом, мы за линейное время находим нужное правило. Очевидно, что трудоемкость полученного алгоритма $O(k)$.

Как видно из описания алгоритма, подтокены обрабатываются рекурсивно. Однако эксперименты на реальных текстах и правилах показали, что необходимости в большой глубине рекурсии нет, т. е. эту глубину можно искусственно ограничить некоторой константой, при этом результат работы алгоритма практически не меняется. Таким образом, получаем алгоритм с трудоемкостью $O(k)$. Такую же трудоемкость имеет алгоритм, положенный в основу названной ранее утилиты *flex*.

Как известно, детерминизация конечного автомата приводит к экспоненциальному росту количества состояний [2]. Поэтому при большом количестве правил быстро растет количество состояний автомата Мура, что приводит к большому расходу памяти. На практике еще большей проблемой (кстати, свойственной многим алгоритмам, основанным на конечных автоматах) является быстро растущее количество переходов в конечном автомате. Для решения этих проблем можно разбить правила на группы и строить автомат Мура для каждой группы отдельно, при этом уменьшая суммарное количество переходов и расход памяти. В то же время это приводит к необходимости искать подходящее правило во всех группах последовательно, что, очевидно, замедляет работу алгоритма. Однако такой подход позволяет достичь разумного компромисса между скоростью и потребляемой памятью.

Также нужно заметить, что сама частота срабатывания правил на реальных текстах может сыграть большую роль в оптимизации алгоритма. Так, при анализе текстов на английском и русском языках наиболее часто срабатывает правило `[:alnum:]+_put`, суть которого состоит в том, чтобы без дополнительной обработки возвращать в качестве результата токен, состоящий только из алфавитно-цифровых символов. Работу алгоритма можно значительно ускорить, если перегруппировать правила таким образом, чтобы наиболее часто встречающиеся из них находились в начале списка и при разбиении правил попадали в первую группу. В общем виде для решения этой задачи необходимо:

- построить отношение частичного порядка на правилах следующим образом: $R_i < R_j$, если $i < j$ $L(E_i) - L(E_j) \neq \emptyset$, где L - язык регулярного выражения [3]; $E_i = E_{i_1} \circ E_{i_2} \circ \dots \circ E_{i_{m_i}}$; $\circ$ - операция конкатенации;
- из отсортированных последовательностей выбрать такую, в которой наиболее частотные правила находятся как можно ближе к началу списка.

При выполнении данных условий среднее количество анализируемых групп уменьшается, поскольку, если подходящее правило найдено в первой из них, остальные не рассматриваются.

Построение отношения частичного порядка является достаточно трудоемкой операцией, поэтому проверять пересечение языков регулярных множеств правил имеет смысл тогда, когда вероятность хотя бы одного из правил пары выше определенного порога.

Представленный алгоритм лексического анализа для систем обработки естественного языка обладает определенными преимуществами по отношению к существующим алгоритмам подобного назначения. В частности, к его достоинствам можно отнести следующие: возможность постепенного написания правил от наиболее простых к сложным; линейная зависимость скорости работы алгоритма от длины обрабатываемого токена; гибкость алгоритма, достигаемая за счет расширения поля «команда». Отметим также, что описанный алгоритм положен в основу разработки модуля лексического анализа промышленного лингвистического процессора таких известных в области решения инноваци-

онных задач информационных систем, как Knowledgeist™, CoBrain™, Gold Fire Research™, Gold Fire Innovation™ [12], в значительной степени обеспечив ему высокие показатели эффективности обработки текста.

1. Nicol G.T. Flex: The Lexical Scanner Generator for Version 2.3.7. Free Software Foundation. 1993.
2. Брауер В. Введение в теорию конечных автоматов. М., 1997.
3. Kam T., Villa T., Brayton R., Sangiovanni-Vincentelli A. Synthesis Of Finite State Machines: Functional Optimization. Kluwer Academic Publishers. 1997.
4. Bider D., Conrad S., Reppen R. Corpus Linguistics. Investigating Language Structure and Use. Cambridge, 1998.
5. Pereira F., Riley V., Sproat R. // ARPA Workshop on Human Language Technology. 1994. P. 249.
6. Piskorski J., Neumann G. An Intelligent Text Extraction and Navigation System. 1999.
7. Karttunen L., Chanod J.-P., Grefenstette G., Schiller A. // Natural Language Engineering. Cambridge, 1997. P. 1.
8. Karttunen L. Directed Replacement. Grenoble, 1996.
9. Roche E., Schabes Y. Introduction to Finite-State Devices in Natural Language Processing Technical Report. Mitsubishi Electric Research Laboratories. 1996.
10. Friedl J. Mastering Regular Expressions. O'Really, 1997.
11. Laurikari V. Efficient submatch addressing for regular expressions. Master's thesis. Helsinki, 2001.
12. www.invention-machine.com/prodserv/GFIN.cfm

Поступила в редакцию 09.04.2004.

Алексей Владимирович Чеусов - инженер-программист ЗАО «НучСофт».