Выводы. Предложена и обоснована методика по выбору параметров зондирующих импульсов и режима измерения сигнала при проведении исследований на спектрометре ЯМР. На числовом примере показано, что компьютерные технологии позволяют практически реализовать предложенные технические решения. Применение подобных решений повысит точность и достоверность получаемых результатов исследований, проводимых с применением спектрометров ЯМР.

Дальнейшая задача, которая должна быть решена с целью повышения уровня автоматизации измерений на спектрометре ЯМР – это создание программно-аппаратных средств, позволяющих в автоматическом режиме по определенному алгоритму изменять временные параметры между зондирующими импульсами, регистрировать амплитуды эхо-сигналов и вычислять необходимые параметры исследуемого продукта.

Решение подобных задач позволит впоследствии создать необходимые предпосылки по внедрению автоматизированных измерительных систем по проведению подобных экспериментов, что в значительной степени сократит как время проведения исследований, так и повысит их эффективность.

Список литературы: **1**. *Дьяков А.Г., Даниленко А.Ф.* Информационно-измерительная система установки ЯМР // Вестник НТУ "ХПИ". – Харьков: НТУ "ХПИ", 2003. – Вып. 19. – С. 69–72. **2**. *Дьяков А.Г., Даниленко А.Ф.* Система управления спектрометром ЯМР // Вестник НТУ "ХПИ". – Харьков: НТУ "ХПИ", 2004. – Вып. 26. – С. 119–123. **3**. *Дьяков А.Г., Даниленко А.Ф.* Повышение точности измерений в ЯМР спектрометре // Вестник НТУ "ХПИ". – Харьков: НТУ "ХПИ", 2005. – Вып. 46. – С. 83–86. **4**. *Олсон Г., Пиани Д.* Цифровые системы автоматизация процесса управления. – СПб.: Невский диалект, 2002. – 254 с. **5.** *Вашман А.А., Прошин И.С.* Ядерная магнитная релаксация и ее применение в химической физике. – М.: Наука, 1979. – 236 с. **6.** *Хартман К.* и др. Планирование эксперимента в исследовании технологических процессов. – М.: Мир, 1977. –- 522 с. **7.** *Эрнст Р., Боденхаузен Дж. и др.* ЯМР в одном и двух измерениях. – М.: Мир, 1990. – 711 с. **8.** *Фаррар Т., Беккер Э.* Импульсная и Фурье-спектроскопия ЯМР. – М.: Мир, 1973. – 164 с.

Поступила в редакцию 05.10.2006

УДК 519.766

Г.Ф. ДЮБКО, канд. техн. наук**,** ХНУРЭ (г. Харьков),
Е.Л. ЛЕЩИНСКАЯ, ХНУРЭ (г. Харьков)

ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ВЫВОДА В ТИПИЗИРОВАННЫХ "ПЛЕКС-ГРАММАТИКАХ"

Обґрунтована необхідність використання спеціальних методів для опису мов, що подають графові структури даних (графових мов): UML, SYSML, VHDL, WWF та ін. Проаналізовані існуючі підходи для подання формальних моделей таких мов, їх переваги та недоліки. Запропоновано оригінальний підхід для формального опису підкласу графових мов (мов подання діаграм руху даних) на основі типізованих плекс-граматик. Розроблені алгоритми породження та розбору фраз, що належать таким мовам. Доведено перевагу використання отриманих результатів у порівнянні з відомими підходами.

A need to use special methods to describe languages, that represent graph-alike data structures, such as UML, SYSML, VHDL, WWF is proved. We call them graph languages. Known approaches to represent formal models of such graph languages are investigated, their advantages and disadvantage are marked. A novel approach to represent formal models of subclass of graph languages (data-flow diagram language) is introduced. This approach is based on typified plex-grammars. Generative and recognizing algorithms for such class of languages are developed. Advantages of obtained results application (compared to classic approaches) are argued.

Постановка проблемы. Хорошо известен широко распространенный формальный подход к описанию синтаксиса некоторого языка, т.е. правил построения конструкций этого языка, на основе аппарата формальных грамматик (ФГ) [1]. В классическом своем варианте аппарат ФГ применяется для описания текстов – линейных цепочек символов. С точки зрения грамматики такого вида существует всего два бинарных отношения между элементами её алфавита: «предшествует» и «следует за». На практике же представляет интерес использования такого формального аппарата для описания структуры языка, между элементами алфавита которого возможно произвольное число отношений любой размерности. Использование такого аппарата значительно упростит представление правил построения конструкций языков, описывающих древовидные и графовые структуры, а значит и создание новых языков такого типа, разработку для них синтаксических анализаторов, компиляторов, интерпретаторов, трансляторов. К множеству таких языков можно отнести унифицированный язык моделирования UML, язык описания цифровых электронных систем VHDL, язык описания систем широкого класса SYSML, язык визуального программирования на базе Windows Workflow Foundation [2] и пр. Все эти языки – неотъемлемая часть систем автоматизированного проектирования и моделирования. **Целью данной статьи** является описание разработанной авторами формальной грамматики для представления правил построения конструкций языков, описывающих графовые структуры данных, а также

представление базовых алгоритмов порождения и разбора фраз на таких языках с использованием правил этой грамматики.

Анализ публикаций. В современной литературе по синтаксическому анализу известна такая форма описания графовых языков, как плекс[1]-грамматики [3] или графовые грамматики [4]. Ключевыми понятиями такой грамматики являются нейп[2] и схема из нейпов – плекс. Как и классическая форма грамматики Хомского [1], обычные плекс-грамматики не содержат никакой семантической информации об элементах языка, лишь описывая его структуру (в продукции плекс-грамматики структура задается перечислением точек соединения, которым просто присваиваются порядковые номера). Преимуществом такого подхода является возможность успешно представлять с помощью плекс-грамматик широкий класс в основном графических языков. Но, как показано в работе [5], алгоритмы разбора, разработанные для таких плекс-грамматик, являются очень неоптимальными – экспоненциальными в худшем случае ($O(|G|^m)$, где m – максимальное количество узлов в правой части продукции плекс-грамматики, на основании которой проводится анализ графа G, $|G|$ – количество узлов в анализируемом графе). Поэтому, несмотря на простоту представления языка с помощью такого формального аппарата, с его помощью невозможно решение задач, основным требованием к которым является скорость отклика системы. Доказательством этого может являться работа [6] по разработке эффективных методов распознавания рукописного текста, в которой авторы утверждают, что отказываются от плекс-грамматик только из-за желания создать систему реального времени.

Постановка задачи. Одним из путей решения такой проблемы может стать добавление специфической информации в продукции грамматики и использование этих дополнительных знаний при разрешении неопределенностей в момент работы алгоритмов разбора. Это, конечно же, сужает множество представимых грамматикой языков, зато позволяет сделать время разбора приемлемым для использования в практических целях. В данной работе выбрано подмножество языков представления flow-chart диаграмм движения данных. Эти диаграммы удобно использовать при проектировании структуры программных и цифровых аппаратных систем в терминах сети взаимодействующих активных блоков. Каждый типовой блок выполняет в системе свою функцию, обладает набором типизированных входов и выходов. Блоки могут быть как примитивные (черный ящик с известным интерфейсом и логикой работы, но неизвестной реализацией), так и композитные (в таких блоках определен не только интерфейс, но и внутренняя структура – подсхема блока определенного типа). Примитивные блоки можно

[1] Plex – в переводе с английского означает «сеть», «сплетение»
[2] Нейп – от английского Nape – n-attached point

считать терминальными символами плекс-грамматики, а составные, соответственно, – нетерминальными. Используя алгоритмы разбора конструкций такого языка можно доказывать правильность построенной схемы, производить поиск типовых участков схемы, которые возможно заменять на пользовательские непримитивные блоки, что ведет к улучшению структуризации схемы, упрощению поиска ошибок, повышению возможности повторного использования.

Цель статьи. Назовем разновидность грамматики для представления диаграмм движения данных типизированной плекс-грамматикой. Целью статьи является предложение её формального описания и алгоритмов порождения и распознавания на её основе конструкций flow-chart языков.

Формальное представление типизированной плекс-грамматики. Пусть *терминальный символ грамматики* (элементарный нейп) определяется как двойка:

<NapeType, Interfaces>,
где NapeType – идентификатор типа узла
Interfaces – множество интерфейсов узла, представленных в виде троек <Kind, InterfaceName, SimpleType>, где
Kind – вид интерфейса (входной/выходной)
InterfaceName – имя интерфейса
SimpleType – тип интерфейса (любой из предопределенных типов данных).

Структуру нейпа можно формально описать в виде следующих порождающих правил грамматики:

Nape → PrimitiveNape | NonePrimitiveNape

Primitive Nape {
PrimitiveNape → NapeType **(** InterfacesList **)**
InterfacesList → Interface | Interface **,** InterfacesList
Interface → Kind InterfaceName **:** SimpleType
Kind → in | out
}

InterfaceName, SimpleType – это идентификаторы.

Нетерминальный символ грамматики (составной нейп) нуждается в описании его интерфейсной и структурной частей. Представить его можно как продукцию плекс-грамматики:

NonePrimitiveNape {
NonePrimitiveNape → NapeType **(** InterfacesList **) ::=**
NodesList **(** EdgesList **) ;**

NodesList → Node | Node**,** NodesList;
Node → NodeName**:** NapeType;
EdgesList → Edge | Edge**;** EdgesList;
Edge → EdgeName **[**PointsList**];**
PointsList → Point | Point**,** PointsList;
Point → Kind NodeName**.**InterfaceName
}

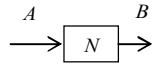
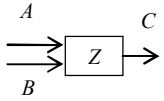
Для упрощения различия между примитивными и непримитивными нейпами будем обозначать в продукции имена примитивных нейпов строчными буквами, а непримитивных – прописными.

Список PointsList может содержать разное число элементов, минимум один. Один элемент в списке PointsList, а также отсутствие точек in или out при описании сигнала означает, что сигнал является интерфейсным (входным или выходным), т.е. противоположный конец (один или несколько) заранее не известен и будет служить для связи с интерфейсными точками внешних нейпов.

В табл. приведен пример правил простой типизированной плекс-грамматики. *n* и *z* – терминальные нейпы, *K* и *W* – нетерминальные. Аксиомой является нетерминальный нейп *W*.

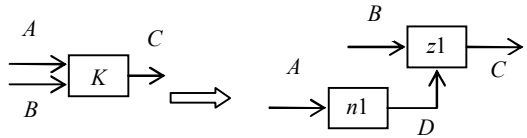
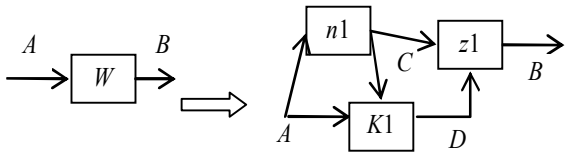
Таблица

Пример простой типизированной плекс-грамматики

В виде графа	В виде текстовых правил
1	2
Терминальные символы	
	*N* (in *A*: int, out *B*:float)
	*Z* (in *A*: float, in *B*:float, out *C*: float)

Продолжение таблицы

Пример простой типизированной плекс-грамматики

1	2
Нетерминальные символы	
	*K* (in *A*:int, in *B*:float, out *C*:float)::= n1: *N*, *z*1: *Z* (*A*:int[in *n*1.*A*]; *B*:float[in *z*1.*A*]; *C*:float[out *z*1.*C*:float]; *D*:float[out n1.*B*, in *z*1.*B*])
	*W* (in *A*:int, out *B*:float)::= *n*1:*N*, *K*1:*K*, *z*1:*Z* (*A*:int[in *n*1.*A*, *K*1.*A*]; *B*:float [out *z*1.*C*]; *C*:float[out *n*1.*B*, in *K*1.*B*, *z*1.*A*]; *D*:float [out *K*1.*C*, in *z*1.*B*])

Задача развертки в типизированной плекс-грамматике. Развертка позволяет порождать различные предложения из аксиомы грамматики, поэтому представляет интерес формальное определение механизма развертки в типизированной плекс-граматике. Развертка предполагает эквивалентное преобразование некоторого плекса путем замены его непримитивных нейпов на порождаемые ими плексы.

Обозначим плекс до преобразования *P*, подвергающийся развертке нейп – *N*, срабатывающее правило *R,* порожденный разверткой плекс – *P*1. Тогда нейпы и сигналы порожденного плекса *P*1 как функции от *P*, *R* и *N* выражаются следующими правилами:

Nodes (P1) = Nodes(P) \/ Map(Nodes(R)) / N;

Edges(P1) =
Edges(P)\/Map(Edges(R))\/Comb(Interfaces(N),Interfaces(R))/Interfaces(N);

Interfaces(P1) = Interfaces(P);

Здесь назначение вспомогательных функций Map и Comb следующее:

Map – функция, отображающая имена нейпов из правила *R* на плекс *P*1 для избежания дублирования имен вновь добавляемых нейпов с существующими;

Comb – функция, комбинирующая дуги расширяемого нейпа *N* с соответсвующими им интерфейсными дугами правила *R*.

Условием срабатывания процедуры развертки в плексе является наличие в нем непримитивного нейпа, известного типа.

Правила преобразования на псевдокоде можно выразить следующим образом:

1. Для каждого непримитивного нейпа *N* в плексе *S*:
1.1. Добавить его в очередь *Q* на разворачивание.
2. Для каждого нейпа *N* очереди *Q*.
2.1. Для каждого нейпа RN_i из правой части продукции *R,* соответсвующей нейпу *N*
2.1.1. Сгенерировать новое уникальное в пределах схемы *S* имя для RN_i , запомнить его в списке замен *RL*[3]. Имя генерируется на основе типа узла.
2.1.2. Добавить RN_i с указанием типа узла в схему.
2.1.3. Если RN_i непримитивен, то поставить его в очередь *Q* на разворачивание.
2.2. Для каждой неинтерфейсной дуги *E* правой части продукции *R*, расширяющей *N*
2.2.1. Сформировать новое уникальное в пределах плекса *S* имя для дуги *E*.
2.2.2. Заменить все имена нейпов дуги *E* в соответсвии со списком замен *RL*.
2.2.3. Добавить дугу *E* в схему *S*.
2.3. Для каждой интерфейсной дуги *E* узла *N* из схемы *S*.
2.3.1. Удалить из дуги *E* все точки соединения с расширяемым нейпом *N*.
2.3.2. Добавить в дугу *E* все точки из соответсвующей ей интерфесной дуги *E'* правой части продукции *R* с учетом замен имен из списка *RL*.
2.4. Удалить нейп *N* из плекса *S*.

При таком подходе можно говорить о левостороннем выводе с точки зрения порядка следования нетерминальных нейпов в продукции плекс-грамматики. Последовательность обработки (разворота) вновь добавляемых в схему узлов соответствует порядку их добавления, а он, соответственно,

[3] от англ. ReplaceList

зависит от порядка следования нейпов в правой части продукции разворачиваемого узла.

Оценим наихудшее время выполнения полной развертки из аксиомы грамматики до схемы, не содержащей непримитивных нейпов:

Шаг 1 выполняется 1 раз.

Шаг 2 выполняется до опустошения очереди *Q*. Оценка максимального числа элементов, которые могут в неё попасть, возможна только для нерекурсивной плекс-грамматики. Построим дерево, корень которого аксиома плекс-грамматики. Потомки первого уровня – непримитивные нейпы правой части продукции этой грамматики, соответсвующей узлу типа аксиомы. Если таких продукций несколько, то берется продукция с максимальным числом непримитивных нейпов. Потомки второго уровня, соответственно, строятся по аналогичному правилу (берутся продукции, соответствующие типам каждого из узлов предыдущего уровня). Общее число элементов, попадающих в очередь *Q* за время работы алгоритма, равно количеству узлов построенного по указанным выше правилам дерева. Обозначим его *QN* (Quee Nodes).

Шаг 2.1. Если обозначить как *Np* (Number of points) число нейпов в правой части продукции с максимальным количеством узлов, то шаг 2.1 будет выполняться *Np* или менее раз. Т.к. оценивается верхняя граница производительности, то можно принять за *Np* число раз исполнения шага 2.1 на каждой итерации цикла 2.

Шаги 2.2. Если обозначить как *MNIE* (Max None Interface points) максимальное число неинтерфейсных соединений среди продукций грамматики, то (аналогично оценке 2.1) шаги 2.2 будут выполняться *MNIE* раз.

Шаг 2.2.2. Примем за *MEN* (Max Nodes per Edge) максимальное число узлов в соединении плекс-продукции. Тогда шаг 2.2.2 на каждой итерации 2.2 выполняется *MEN* раз. Эта оценка дана с учетом того, что список замен *RL* организован как хеш-таблица и поиск в нем осуществляется за время O(1).

Шаг 2.3. Если обозначить как *MIE* (Max Interface points) максимальное число интерфейсных соединений среди продукций грамматики, то шаг 2.3 будет выполняться *MIE* раз на каждой итерации.

Шаг 2.3.1 и 2.3.2. На каждой итерации цикла 2.3 будут выполняться по *MEN* раз

Шаг 2.4. На каждой итерации цикла 2 будет выполняться 1 раз.

Всего число шагов исполнения алгоритма равно:
$QN*(Np + MNIE + MNIE*MEN + MNIE + MIE*2*MEN + 1) =$
$= QN*Np + QN*MNIE + QN*MNIE*MEN + MN*MNIE +$
$+ QN*MIE*2*MEN + QN*1.$

Максимальный вклад в оценку алгоритма дают слагаемые:
$QN*Np$,
$QN*MNIE*MEN$,
$QN*MIE*2*MEN$.

Поскольку число интерфейсных дуг обычно много меньше числа неинтерфейсных дуг в плекс-продукции, то третье слагаемое значительно меньше второго. Остается сравнить *QN*Np* и *QN*MNIE*MEN*. Поскольку число соединений в схеме обычно превосходит число узлов, то первое произведение много меньше второго, поэтому верхнюю оценку алгоритма развертки в плекс-грамматике можно оценить как O(*QN*MNIE*MEN*).

Такую оценку можно считать кубической.

Пример развертки нейпа *K*1 в правиле, соответствующем нейпу *W*, в приведенной выше грамматике, показан на рис.1

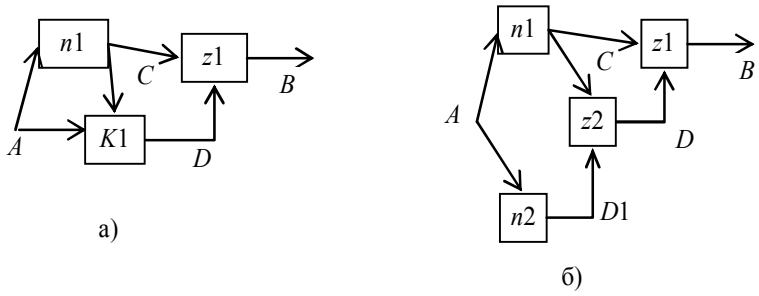


Рис. 1. Плекс а) "до" и б) "после" развертки нейпа *K*1

Как видно из рис. 1, сигнал *A* исходного плекса скомбинирован с интерфейсным сигналом *A* сработавшего правила *K*, а сигнал *C* исходного плекса скомбинирован с интерфейсным сигналом *B* сработавшего правила *K*.

Задача свертки в плекс-грамматике. Задача свертки позволяет в случае удачи доказывать принадлежность анализируемой фразы языку, порождаемому грамматикой.

Свертка заключается в последовательной замене подграфов в плексе на соответствующие им непримитивные нейпы. Таким образом, граф упрощается вплоть до одной вершины и интерфейсных дуг.

Обозначим плекс до преобразования *P*, срабатывающее правило *R*, формируемый срабатывающим правилом узел – *N*, результирующий плекс – *P*1. Тогда нейпы и сигналы порожденного плекса *P*1 как функции от *P*, *R* и *N* выражаются следующими правилами:

Nodes (P1) = (Nodes(P) / Map(Nodes(R))) \/ N

Edges(P1) = (Edges(P) / Map(Edges(R))) \/
Comb(Map(Interfaces(N)),Interfaces(R))

Interfaces(P1) = Interfaces(P)

Как и в классическом варианте грамматики при свертке необходимо определить наличие правой части срабатывающего правила. Эта задача

известна в классической теории графов как задача поиска подграфа в графе. Она является NP-полной и в самом общем случае требует для своего решения полного перебора [7]. В модифицированной для типизированной плекс-грамматики форме оптимизация достигается благодаря использованию дополнительной информации (тип узла, тип связи, тип точки соединения). Свертка считается возможной при выполнении следующих условий:

– найдено отображение набора вершин правой части правила *R* на вершины графа *P*;

– найдено отображение на внутренние связи правила *R* некоторого подмножества дуг графа *P*;

– найдено отображение внешних связей множества вершин правила *R* на подмножество дуг графа *P*;

– объединение внутренних и внешних связей отображенных вершин правила *R* – это полное множество дуг, связанных с этими вершинами графа *P*.

Возможны два подхода к свертке. При первом из них все потенциальные шаблоны упорядочиваются по количеству входящих в них узлов и поиск на каждой итерации осуществляется от более крупных к более мелким (известный прием, позволяющий избежать некоторых видов конфликтов [8]) и так вплоть до нахождения аксиомы. Изначально был разработан алгоритм свертки, решающий задачу поиска отображения правой части продукции на некоторый подграф плекса *P*, используя сокращенную матрицу смежности, содержащую в качестве *строк множество точек выхода* связей искомого шаблона, а в качестве *столбцов – множество точек входа* связей искомого шаблона. Для продукции такая матрица может быть построена заранее в момент создания продукции. Для сворачиваемого плекса – после определения предполагаемого множества узлов, подвергаемых свертке. Тогда сложность задачи сравнения двух плексов можно оценить как квадратичную, равную

DubleCompl = *OutPoints*(*Pattern*)**InPoints*(*Pattern*),

где *Pattern* – плекс правой части срабатывающего правила;

OutPoints(*Plex*) – множество исходящих точек сигналов плекса;

InPoints(*Plex*) – множество входящих точек сигналов плекса;

а, следовательно, сложность задачи поиска правой части продукции на всех комбинациях вершин графа *P* (в худшем случае) составляла:

$$\text{AllCompl}=DubleCompl* \prod_{tp \in Pattern.Types} A_{SchemeNodesByType(tp)}^{PatternNodesByType(tp)},$$

где Pattern.Types – множество различных типов нейпов искомого шаблона;

$PaternNodesByType(tp)$ – количество узлов типа *tp* в искомом шаблоне;

$SchemeNodesByType(tp)$ – количество узлов типа *tp* в плексе, в котором производится поиск шаблона;

$A_{SchemeNodesByType(tp)}^{PaternNodesByType(tp)} = A_n^k = n \cdot (n-1) \cdot \ldots \cdot (n-(k-1))$ – число размещений из *n* по *k*,

$n = SchemeNodesByType(tp)$, $k = PaternNodesByType(tp)$.

Такая производительность неприемлема для поиска шаблонов в плексах больших размеров, наиболее часто встречающихся в реальных задачах. В последней модификации алгоритма предлагается производить поиск шаблона в плексе «целенаправленно»: если известно отображение некоторого узла шаблона *N* на узел плекса *N'*, то логичным будет производить поиск связанных с *N* по шаблону узлов, отображая их на узлы, связанные с *N'* (с учетом типа узла и типа точки соединения). Кроме того, изначально рекурсивный алгоритм поиска с возвратом был преобразован в итеративный, что делает его независимым от аппаратного размера стека исполняющей машины и позволяет анализировать плексы любой сложности. Этот алгоритм и приведен ниже:

1. По аксиоме сформировать целевой плекс *PG* из одного узла.
2. В стек-план действий поместить действие **"удовлетворить шаблон"** *PG*. Сбросить статус контекста в "Ok".
3. Пока в стеке-плане есть действия, повторять шаг 4.
   3.1. Извлечь и обработать действие из стека-плана.
4. Если статус контекста Ok, то свертка удалась.

Под статусом понимается статус завершения предыдущего действия (далее Status): {Ok, Err}.

Предусмотрена обработка следующих типов действий: "**удовлетворение шаблона**", "**удовлетворение сигнала"**.

Каждое действие может исполняться как за один раз, так и откладываться ("засыпать", отодвигаться по стеку-плану действий вниз) до определенного момента, когда будут доступны все данные, чтобы либо удовлетворить его, либо отвергнуть как невозможное. Для осуществления возвратов ведется журнал действий *HJ* (History Journal), в котором фиксируются закрепления принятых соответствий узлов шаблона и анализируемого плекса.

Алгоритм обработки действия ***"удовлетворить шаблон"***.

Сохраняемая с действием контекстная информация:

- имя действия *AN* (Action Name);
- родительское действие *P* (Parent);
- шаблон *G* (Graph);
- анализируемый плекс *PL* (Plex);
- планируемый к удовлетворению интерфейс *PS* (Planned Signal);
- множество *SS* (Signals Set) потенциальных сигналов для удовлетворения *PS*;
- номер *i* текущего рассматриваемого сигнала из потенциального множества *SS*;
- начальный входной сигнал *FIS* (First Input Signal) – сигнал шаблона, с которого предполагается начать его сопоставление;

- потенциальный начальный входной сигнал *PFIS* (Potential First Input Signal) – сигнал плекса, с которым предполагается сопоставить *FIS*;
- режим исполнения Mode {New – исполняется в первый раз, EdgeSat – удовлетворение связи}.

1. Если исполняется первый раз (Context.Mode = New), то
   1.1. Сформировать уникальное имя действия *AN*.
   1.2. Если установлены *FIS* и *PFIS*, то
      1.2.1. Установить *PS* = *FIS,* добавить *PFIS* в множество *SS*.
   1.3. Иначе для первого входного интерфейсного сигнала *IS* (Interface Signal) шаблона *G* установить:
      1.3.1. *PS* =*IS*, а для всех интерфейсных сигналов такого типа в анализируемом плексе *PL* – добавить их в *SS*.
   1.4. *i*=1.
2. Иначе
   2.1. Если Status = Ok, то "конец" – шаблон удовлетворен.
   2.2. Иначе, если Status = Err, то откатить все действия из *HJ*, родителем которых является AN. *i*++.
3. Если $i <= |SS|$, то добавить в план действия текущее действие и действие "удовлетворить сигнал" (Параметры: шаблон-сигнал *PS;* потенциальный сигнал-пара *SS*[*i*]*;* шаблон-плекс, чей сигнал удовлетворяется, *G;* вызвавшее действие *AN*). Конец текущего этапа, анализ отложен до возвращения к данному действию.
4. Иначе Status = Err, конец – шаблон не удовлетворен.

Алгоритм обработки действия ***"удовлетворить сигнал"***.

Сохраняемая с действием контекстная информация:

- имя действия *AN* (Action Name);
- родительское действие *P* (Parent);
- шаблон-плекс *G* (Graph), которому принадлежит сигнал;
- анализируемый плекс *PL* (Plex);
- целевой сигнал *AS* (Aimed signal);
- потенциальный сигнал *PS* (Potential Signal);
- планируемый к удовлетворению список интерфейсных точек сигнала *NS* (Nodes Set), каждая точка задается типом узла и именем интерфейса;
- номер текущей неудовлетворенной в *NS* точки – *i*;
- множество *PP* (Potential points) потенциальных точек сигнала *PS*, удовлетворяющих *NS*[*i*];
- номер *j* текущей рассматриваемой точки из *PP*;
- множество *RP* (Reserved Points) точек сигнала *PS*, одна из которых может принадлежать сворачиваемому интерфейсному сигналу *NS*[*i*].InterfaceName (в случае, когда *NS*[*i*] не присутствует в плексе явно, а предположительно, будет получена сверткой от точки из *RP*);

– номер k текущей рассматриваемой точки из RP;

– план удовлетворения интерфейсов IP (Interface satisfaction Plan) текущей закрепленной точки $NS[i]$;

– номер m текущего удовлетворяемого интерфейса из IP;

– режим исполнения Mode {NodeReduce-свертка, EdgeSat – удовлетворение связи, New – исполняется первый раз}.

1. Если исполняется первый раз (*Context.Mode* = New), то составить план удовлетворения точек NS (вначале out, затем in, не учитывая уже закрепленные), $i = 1$. Сформировать уникальное имя действия AN.

2. Иначе

2.1. Если *Mode* = NodeReduce, то

2.1.1. Если *Status* = Ok, то произвести свертку согласно записям из HJ о закреплении всех узлов правила, соответствующего $NS[i]$. Поместить соответствующее действие в HJ, снять все пометки о "резервировании" в PS, Добавить в PP вновь созданный узел, j++. Пометить его как "зарезервированный", перейти к 3.6.

2.1.2. Иначе (Status = Err). Пометить $RP[k]$ как "зарезервированную", k++, перейти к 3.4.

2.2. Иначе, если *Mode* = EdgeSat, то

2.2.1. Если Status = Ok, то m++; перейти к 3.7.

2.2.2. Иначе (Status = Err). Снять пометку о "резервировании" с $PP[j]$. Если $PP[j]$ – оригинальная вершина, то j++, перейти к 3.2, иначе ($PP[j]$ – свернутая вершина), k++, перейти к 3.4.

3. Если $i <= |NS|$, то

3.1. Из свободных точек сигнала PS составить множество PP точек, потенциальных пар $NS[i]$, j=1.

3.2. Если $j <= |PP|$, то пометить $PP[j]$ как "зарезервированную".

3.3. Иначе из свободных точек сигнала PS составить множество RP, такое, что $\{\forall\, v \in RP \mid v \in In(NS[i].InterfaceName)\}$ – любая точка v может явиться точкой входа интерфейсного сигнала NS[i].InterfaceName в случае свертки от v до $NS[i]$, $k = 1$.

3.4. Если $k <= |RP|$, то *Mode* = NodeReduce, добавить в план действий текущее действие и действие "удовлетворить шаблон" (тип шаблона = $NS[i]$, начальный входной сигнал = $NS[i]$.InterfaceName, потенциальный парный сигнал = PS). При сопоставлении этих сигналов уже не будут учитываться закрепленные ранее точки PS; конец.

3.5. Иначе Status=Err, i--. Если $i > 0$, то восстановить состояние действия из AStatus[i], отменить по HJ все действия, родителем или инициатором которых является AN, перейти к 2.2.2, иначе конец.

3.6. Составить план удовлетворения интерфейсов IP, для всех связей $NS[i]$, кроме out сигналов плекса-шаблона G и целевого сигнала AS, $m = 1$.

3.7. Если $m <= |IP|$, то добавить в план действия текущее действие и действие "удовлетворить сигнал" (Целевой сигнал = $IP[m]$, Потенциальный сигнал = $PP[j]$. InterfaceName[m], плекс-шаблон G), *Mode* = EdgeSat, конец.

3.8. Иначе добавить в HJ запись о закреплении пары ($NS[i]$, $PP[j]$), Запомнить состояние текущего действия в AStatus[i].

3.9. i++, сформировать уникальное имя действия AN, перейти к 2.

4. Иначе Status = Ok, конец.

Оценим наихудшее время выполнения свертки из исходного плекса до аксиомы грамматики или признания невозможности этого.

Если предположить, что свертка происходит без единого возврата, то затраченное время будет пропорционально числу интерфейсных точек плекса, обозначим его PIP (Plex Interface Points). Этот вывод сделан на основании того, что поиск шаблона в плексе сводится к поиску соответствия сигналов, а два сигнала считаются соответствующими, если у них соответствуют точки входа и точки выхода, образующие сигнал. Количество же возвратов зависит от числа неопределенностей, встречающихся во время работы алгоритма. Признаком наличия таких неопределенностей является отличный от единицы размер множеств PP и RP, т.е. один сигнал связывает несколько входов или выходов однотипных нейпов. Такой случай встречается и в приведенном в данной статье примере плекс-грамматики: входящий сигнал A правила W имеет в ситуации полной развертки две *in* точки типа N (вход A). То же самое можно сказать и про сигнал C того же правила (две *in* точки типа Z, вход A) см. рис.1.б.

Чтобы оценить наихудшее время работы такого алгоритма необходимо:

– найти сигнал с наибольшим числом неопределенностей любого типа. Обозначим число таких неопределенностей MNA (Max Number of Ambiguities);

– максимальное число узлов в соединении плекс-продукции. Это число еще при оценке шага 2.2.2 алгоритма развертки было обозначено MEN (Max Nodes per Edge);

– вычислить высоту дерева, построенного еще при оценке шага 2 алгоритма развертки, обозначим её HT (height of tree).

Тогда наибольшее число различных вариантов соединений, которые необходимо будет проверить можно вычислить по формуле MNA^{HT}. Следовательно, если на каждом шаге размер соединения будет составлять MEN узлов, то наибольшее число действий, которые окажутся в стеке в течение работы алгоритма свертки можно оценить как $O(MEN \cdot MNA^{HT})$. Хотя такой показатель и является экспоненциальным, но в реальности он

значительно меньше времени работы алгоритма свертки в классической плекс-грамматике. К примеру, для плекса на рис. 1.б продолжительность свертки оценивается в 24 шага (MEN = 3; MNA = 2; HT = 3), в то время как для алгорима из [9] она бы оценивалась в 125 шагов. На самом деле данный плекс сворачивается за 12 шагов.

Выводы. Разработанные математические основы позволяют эффективно использовать типизированные плекс-грамматики в задачах анализа и преобразования семантически нагруженных графовых структур, представляющих собой диаграммы движения данных.

Список литературы: 1. Formal syntax and semantics of programming languages: a laboratory based approach / *Kenneth Slonneger, Barry L. Kurtz*. – Addison-Wesley PC, 1995. – 637 p. **2.** *Колдовский В.* Знакомство с Windows Workflow Foundation – Компьютерное Обозрение, 2006, http://itc.ua/article.phtml?ID=23217 **3.** *Карпов В.* Теория и технология программирования. Основы построения трансляторов. – СПб.: БХВ-Петербург, 2005. – 272 с. **4.** *Rekers, J., Schürr A.* Definiting and Parsing Visual Languages with Layered Graph Grammars // Journal of Visual Languages and Computing. – 1997. – Vol.10. – P. 27 – 55. **5.** *Keukelaar J. H. D.* Topics in Soft Computing: Doctoral Dissertation – Royal Institute of Technology, Department of Numerical Analysis and Computer Science. – Stockholm, 2002. – 176 p. **6.** *Kam-Fai Chan, Dit-Yan Yeung* Elastic structural matching for recognithing on-line handwritten alphanumeric characters: Technical report HKUST-CS98-07 / Department of Computer Science, The Hong Kong University of Science & Technology, 1998. – 29 p. **7.** *Алексеев В.Б., Носов В.А.* NP-полные задачи и их полиномиальные варианты. Обзор // Обозрение промышленной и прикладной математики. – 1997. – Т.4. – №2. – С. 165 – 193. **8.** *Льюис Ф., Розенкранц Д., Стирнз Р.* Теоретические основы проектирования компиляторов – М.: Мир, 1990. – 654 с. **9.** *Da Qian Zhang, Kang Zhang, Jiannong Cao* A Context-sensitive Graph Grammar Formalism For the Specification of Visual Lenguages // The computer journal. – 2001. – Vol. 44. – № 3. – 15 p.

Поступила в редакцию 30.09.2006

УДК 621.391

Е.Г. ЖИЛЯКОВ, д-р техн. наук, БелГУ (г. Белгород, Россия),
С.П. БЕЛОВ, канд. техн. наук, БелГУ (г. Белгород, Россия),
Е.И. ПРОХОРЕНКО

ОБ ОДНОМ СПОСОБЕ ОБНАРУЖЕНИЯ ПАУЗ В РЕЧЕВЫХ ДАННЫХ

У статті пропонується один з шляхів рішення задачі зменшення об'єму бітового представлення мовних даних при їх передачі і зберіганні в інформаційно-телекомунікаційних системах. Це новий спосіб виявлення і кодування пауз, заснований на обліку відмінностей в розподілі енергетичних частотних складових звуків мови і сигналу у паузі.

In article is offered one of the ways of decision a problem of reduction of volume of bit presentation voice data. This is the new method founded on highlighting and coding in input an voice a signal a pause based on the account of differences in distributing power frequency constituents of sounds of speech and signal in a pause.

Постановка задачи. В настоящее время наблюдается непрерывное увеличение объемов речевых данных в общем потоке информации, циркулирующем в информационно-телекоммуникационных системах [1, 2]. В связи с этим возникает необходимость решения задачи обеспечения высокой эффективности их хранения и передачи. Одним из путей решения данной задачи может стать разработка новых методов сжатия (кодирования) речевых данных, применение которых позволит значительно сократить объем их битового представления, и, как следствие, уменьшить скорость их передачи по каналам связи, а также ресурсы памяти при их записи на физические носители. Данная работа посвящена рассмотрению именно этих вопросов.

Анализ литературы. При решении задачи кодирования речевых данных целесообразно учитывать ряд особенностей речевого обмена. Речь, как известно [3, 4], обладает смысловой и сигнальной избыточностью и допускает определенный уровень потерь и искажений при осуществлении преобразований. В зависимости от избыточности цифровых данных и длительности обрабатываемых сегментов речи, допустимый уровень потерь может составлять от 1 до 50% времени активности диктора [5]. Различные методы устранения избыточности в процессе преобразования речевых данных представляют широкий диапазон возможных уменьшений объема их битового представления.

Особенностью речевых сигналов является высокая доля пауз, составляющая в среднем, например, при телефонных переговорах, 56 % от длительности диалога [6]. Речь содержит множество кратких перерывов длительностью от 5 до 200 мс, существующих как внутри слов, так и между