21 мая 2008 г.

П. В. Егоров

ЯЗЫКИ ПРОГРАММИРОВАНИЯ С РАСШИРЯЕМЫМ СИНТАКСИСОМ. СИНТАКСИЧЕСКИЙ АНАЛИЗ

1. Введение

Все разработчики языков программирования (ЯП) стоят перед выбором: сделать язык максимально богатым, включив в него поддержку многих парадигм и технологий программирования, либо ограничиться небольшим ядром языка, поместив всё остальное в различные библиотеки. Типичными примерами первого подхода являются языки Perl и C++. Примерами второго подхода – языки семейства Pascal. У каждого из этих подходов есть как достоинства, так и недостатки.

В данной работе рассматривается способ решения этого противоречия – – расширяемый синтаксис языка программирования. Основная идея языка программирования с расширяемым синтаксисом заключается в том, чтобы с одной стороны можно ограничиваться небольшим подмножеством языка везде, где этого достаточно, а в модулях, требующих интенсивной работы с некоторой технологией или парадигмой, подключать соответствующее расширение синтаксиса. Таким образом, язык можно сделать максимально мощным, но при этом позволить программисту полностью контролировать богатство языка в каждый момент времени. Эта идея была озвучена довольно давно.

Существуют впечатляющие примеры успешных, гибко расширяющихся программных продуктов, например, интернет пейджер Miranda IM, или среда разработки Eclipse. Богатство функциональных возможностей в них сочетаются с гибкостью добавления новых возможностей. Разрабатывая и подключая к ядру программы дополнительные модули можно расширять возможности программы. Благодаря опубликованному описанию интерфейса создавать дополнительные модули могут сторонние разработчики и вообще любые желающие.

Для того чтобы лучше проиллюстрировать выгоды, которые получат программисты от такого подхода к созданию трансляторов, рассмотрим пример использования гипотетического языка с расширяемым синтаксисом.

Пример 1.1. *Использование ЯП с расширяемым синтаксисом.*

```
syntax Sql;

string name = "Иван";
SqlQuery q = sql(select count(*) from Persons where name=$name);
int count = connection.Execute(q);
```

Сначала командой `syntax Sql` подключается расширение синтаксиса, добавляющее поддержку технологии sql-запросов. Это позволяет писать sql запросы прямо в исходном коде. Благодаря этому среда разработки может подсвечивать синтаксис sql запроса, а компилятор проверять его корректность непосредственно на этапе компиляции. Основные выгоды от использования ЯП с расширяемым синтаксисом можно сформулировать следующим образом:

1. Для решения каждого класса специфических задач можно создать расширение с максимально удобным синтаксисом.

2. Многие проверки переходят с этапа выполнения на этап компиляции, увеличивая скорость обнаружения ошибок. В данном примере так случилось с проверкой корректности sql-запроса.

Важно подчеркнуть, что различные расширения ЯП могут создаваться группами разработчиков, независимыми от группы, создавшей ядро языка. Этот факт может привести к качественному увеличению скорости развития языков и парадигм программирования. Раньше, для проверки практикой какой-то новой концепции создавались целые новые языки. Так, например, дизайн по контракту был реализован в специально созданном для этого языке Eiffel. Проверяемые исключения впервые были опробованы в языке Java (к слову, не очень удачно). Если бы можно было вводить поддержку своих идей в свой любимый язык программирования, новые концепции стали изобретаться бы и отбраковываться гораздо быстрее и эффективнее.

Итак, преимущества довольно внушительные, и это подтверждается многочисленными попытками создать язык с расширяемым синтаксисом. В качестве примеров таких языков можно назвать, например, язык Haskel с его метапрограммированием ([2]), и новые, развивающиеся языки Nemerle ([3, 4]) и Boo с их системой макросов. Сама же идея того, что будущее программной инженерии за гибкими, расширяемыми языками была озвучена довольно давно, например в статье Extensible Programming for the 21st Century (Расширяемое программирование для 21-ого века) [5].

В данной работе будет рассмотрен несколько иной подход к созданию расширяемых языков - несколько более низкоуровненевый, но лишённый всех ограничений, связанных с использованием макросов.

2. Расширение

Первым делом нужно определиться с термином «расширение». Пусть есть некоторый транслируемый язык L_0. Для начала зафиксируем неформальное определение. Язык L_1 можно называть расширением L_0, если он сохраняет все возможности расширяемого языка, добавляя какие-то новые возможности. Наилучшим вариантом является сохранение обратной совместимости расширения с исходным языком.

Будем называть язык L_1 и его транслятор T_1 **совместимым расширением** языка L_0 и его транслятора T_0, если любая корректная программа на языке L_0 будет корректной и в языке L_1, а кроме того, результат трансляции этой программы в языке L_1 останется тем же, что и в языке L_0.

Обычно процесс трансляции разбивают на несколько последовательных этапов. Создание ЯП с расширяемым синтаксисом означает превращение каждого такого этапа в расширяемый. При этом контролировать сохранение обратной совместимости можно на каждом этапе по отдельности: если обратная совместимость будет сохраняться при расширении на каждом из этапов трансляции, то всё расширение транслятора целиком также будет обладать обратной совместимостью с исходным языком. Сегодня выделение в качестве двух первых этапов трансляции лексического и синтаксического анализов стало стандартом де-факто. В этой работе будут рассмотрены подходы к созданию расширяемой версий синтаксического анализатора, сохраняющего совместимость при расширении.

3. Синтаксический анализ

Обозначим через CFG множество всех контекстно-свободных однозначных приведенных грамматик. Далее мы будем иметь дело только с такими грамматиками. Обозначим символом Γ некоторое подмножество CFG и зафиксируем его. Оно будет использоваться на протяжении всей статьи.

Ниже будут использоваться следующие общепринятые обозначения. ([1, стр. 27]) Нетерминальные символы грамматики будут обозначаться прописными латинскими буквами. Строчными латинскими буквами из конца алфавита будут обозначатся слова над алфавитом терминальных символов. Терминальные символы – строчными латинскими буквами из начала алфавита. Слова из терминальных и нетерминальных символов – греческими маленькими буквами.

Определение 3.1. *Пусть L – некоторый язык. Тогда синтаксическим анализатором G будем называть произвольную функцию, определенную на L.*

Таким образом на данном этапе мы не накладываем никаких ограничений на то, что является результатом работы синтаксического анализатора.

Определение 3.2. *Пусть $w \in L(G), s$ – синтаксический анализатор языка $L(G)$. Тогда $s(w)$ будем называть внутренним представлением слова w в грамматике G.*

Далее дадим более формальное определение обратной совместимости для синтаксических анализаторов.

Определение 3.3. *Пусть $G_1, G_2 \in \Gamma : L(G_1) \subseteq L(G_2)$. Кроме того пусть s_1, s_2 – синтаксические анализаторы языков $L(G_1)$ и $L(G_2)$. Тогда s_2 будем называть совместимым с s_1, если $s_2\big|_{L(G_1)} = s_1$*

3.1. Расширяющие преобразования грамматик

Этап синтаксического анализа управляется грамматикой соответствующего языка. Соответственно, расширение синтаксического анализатора может заключаться в расширении соответствующей грамматики.

Определение 3.4. *Преобразование грамматик $\phi : \Gamma \to \Gamma$ называется расширяющим преобразованием грамматик из Γ, если*

$$\forall G \in \Gamma \Rightarrow L(G) \subseteq L(\phi(G))$$

.

Перобразование $\phi : CFG \to CFG$ такое, что его сужение на Γ является расширяющим также будем называть расширяющим.

Далее будет дано определение множества AE удобных для исследования расширяющих преобразований грамматик. В этой работе мы ограничимся только преобразованиями из этого класса AE.

Определение 3.5. *Определим множество преобразований Add как множество всех преобразований $\phi : CFG \to CFG$, добавляющих в грамматику новое правило. Другими словами таких, что выполняются следующие условия:*

$$\phi(V, \Sigma, P, S) = (V, \Sigma_1, P_1, S),$$
$$\Sigma \subseteq \Sigma_1, P \subset P_1, |P_1 \setminus P| = 1.$$

Преобразование из Add может увеличить терминальный алфавит, но не увеличивает нетерминальный.

Лемма 3.1. *Любая грамматика $G = (V, \Sigma, P, S)$ из Γ может быть получена из пустой грамматики $G_0 = (\{S\}, \Sigma, \emptyset, S)$, последовательным применением преобразований из Add.*

Определение 3.6. *Определим множество преобразований Extract как множество всех преобразований $\phi : CFG \to CFG$ таких, что выполняются следующие условия:*

1. $\phi(V, \Sigma, P, S) = (V_1, \Sigma, P_1, S)$
2. $V_1 = V \cup \{B\}, B \notin V$
3. $P_1 = (P \setminus \{A \to \alpha\beta\gamma\}) \cup \{A \to \alpha B\gamma, B \to \beta\}$, *для некоторого правила* $A \to \alpha\beta\gamma$ *из* P.

Другими словами, после применения преобразования из $Extract$, одно некоторое правило $A \to \alpha\beta\gamma$ заменяется двумя правилами:

$$A \to \alpha B\gamma$$

$$B \to \beta$$

Преобразования из $Extract$ расширяют нетерминальный алфавит, но не расширяют терминальный. Кроме того они не изменяют языка грамматики.

Определение 3.7. $AE = Add \cup Extract$

Таким образом в множестве AE преобразования $Extract$ играют роль подготовки грамматики к расширению языка, а преобразования Add собственно расширяют язык.

Далее мы будем работать только с расширениями, полученными композицией конечного количества преобразований из AE:

Определение 3.8. *Пусть T – класс некоторых расширяющих преобразований в Γ. Обозначим*

$$T^\Gamma = \{\phi \in \langle T \rangle | \phi(\Gamma) \subseteq \Gamma\}$$

Довольно легко заметить, что элементы AE^Γ – это расширяющие преобразования грамматик из Γ.

Однако ограничивая себя только преобразованиями из AE, возможно, мы слишком сильно ограничиваем свои возможности по расширению языка. Поэтому вопрос о выразительной силе преобразований из AE требует дополнительных исследований. Следующие два определения являются попыткой формализовать вопрос о выразительной силе систем расширяющих преобразований грамматик.

Определение 3.9. *$T \subseteq \{\phi : CFG \to CFG\}$ называется исчерпывающей в Γ системой расширяющих преобразований если выполняется условие*

$$\forall G_1 \in \Gamma, \forall L_2 \in L(\Gamma) : L(G_1) \subseteq L_2 \Rightarrow \exists \phi \in T^{\Gamma} : L(\phi(G_1)) = L_2.$$

Исчерпывающая система преобразований идеальна для расширения грамматик. С её помощью можно из любой грамматики сделать любую грамматику, распознающую произвольное расширение языка исходной грамматики.

Определение 3.10. *Система преобразований $T \subseteq \{\phi : CFG \to CFG\}$ называется полной в Γ, если выполняются условия*

$$\forall L_1, L_2 \in L(\Gamma) : L_1 \subseteq L_2 \Rightarrow$$

$$\exists G_1 \in \Gamma, \exists \phi \in T^{\Gamma} : L(G_1) = L_1, L(\phi(G_1)) = L_2.$$

Легко видеть, что второе определение более слабое: любая исчерпывающая система преобразований является полной. Менее формально суть полноты заключается в следующем: если знать заранее какое расширение может потребоваться, то для исходного языка можно подобрать удачно расширяемую грамматику.

Далее будет представлена серия предложений в которых система AE исследуется на полноту и исчерпываемость для разных классов грамматик. Но сначала сформулируем несколько простых лемм, которыми в дальнейшем нам придётся несколько раз воспользоваться.

Лемма 3.2. *Пусть $G \in \Gamma, \phi \in AE^{\Gamma}, w \in L(G)$. Рассмотрим вывод w в G:*

$$S \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow w$$

Тогда существует следующий вывод w в $\phi(G)$:

$$S \Rightarrow \alpha_{0,1} \Rightarrow \alpha_{0,2} \Rightarrow \cdots \Rightarrow \alpha_1 \Rightarrow \alpha_{1,1} \Rightarrow \alpha_{1,2} \Rightarrow \cdots \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow w$$

Лемма 3.3. *AE не является исчерпывающей в классе $LL(1)$ грамматик.*

Доказательство. Рассмотрим грамматику G_1:

$$S \to AB; A \to ab; B \to ac$$

Она порождает язык $L_1 = \{abac\}$. Рассмотрим язык $L_2 = \{abac, acab\}$, для которого, очевидно, существует $LL(1)$-граматика. Ниже докажем, что никаким преобразованием из $AE^{LL(1)}$ невозможно привести G_1 к произвольной

$LL(1)$-грамматике G_2 такой, что $L(G_2) = L_2$. Доказательство будем вести от противного.

Допустим существует такое преобразование $\phi \in AE^{LL(1)}$, что $\phi(G_1) = G_2$. Рассмотрим вывод слова $abac$ в G_1:

$$S \Rightarrow AB \Rightarrow abB \Rightarrow abac$$

Согласно лемме 3.2, вывод слова $abac$ в G_2 может быть такой:

$$S \Rightarrow \cdots \Rightarrow AB \Rightarrow \cdots \Rightarrow abB \Rightarrow \cdots \Rightarrow abac$$

Значит в G_2 $B \Rightarrow^* ac$

Кроме этого, поскольку $acab$ так же как и $abac$ начинается с буквы a и в силу принадлежности G_2 к классу $LL(1)$-грамматик, вывод слова $acab$ в G_2 может иметь вид:

$$S \Rightarrow \cdots \Rightarrow A\beta \Rightarrow \cdots \Rightarrow w\beta \Rightarrow \cdots \Rightarrow acab$$

Тут w состоит только из терминалов. Рассмотрим все возможные значения w: $\epsilon, a, ac, aca, aca, acab$. Учитывая то, что из B выводится ac, из слова AB может быть выведено одно из следующих слов: $ac, aac, acac, acaac, acabac$. Кроме того AB само выводится из аксиомы. Получаем что одно из перечисленных выше слов содержится в $L_2 = L(G_2)$, что противоречит условию. Значит исходная посылка неверна и действительно не существует преобразования из AE переводящего G_1 в G_2.

Обозначим через FL множество всех грамматик из Γ, порождающих конечные непустые языки.

Предложение 3.1. *Система преобразований AE является исчерпывающей в FL.*

Доказательство. Пусть $L_2 \setminus L_1 = \{w_1, \ldots, w_n\}$. Добавим с помощью преобразований из Add в грамматику G_1 n правил вида $S \to w_i$.

Предложение 3.2. *AE является полной, но не исчерпывающей в $LL(1) \cap FL$.*

Доказательство. То, что AE не является исчерпывающей, следует из примера, построенного при доказательстве замечания 3.3.

Обозначим через R множество $LL(1)$-грамматик, в которых все правила имеют вид $A \to a_1 \ldots a_k$ или $A \to a_1 \ldots a_k T$, где $T \in V$, а $a_1, \ldots a_l \in \Sigma$. Для

любого слова w из терминальных символов, грамматика с одним правилом вывода вида $S \to w$ принадлежит множеству R.

Построим алгоритм расширения произвольной грамматики $G_1 \in R$ до грамматики $G_2 \in R$ так, что $L(G_2)$ содержит на одно наперёд заданное слово w больше, чем $L(G_1)$. Найдём правило вывода, с наиболее длинным общим префиксом с w. Обозначим найденное правило $A \to u\alpha$, где u общий префикс: $w = us$. Удалим это правило из грамматики, и добавим три следующих правила: $A \to uB$, $B \to \alpha$ и $B \to s$, где B – это некоторый новый нетерминал.

Таким алгоритмом можно построить грамматику языка L_1 из R, а потом дополнить её до грамматики языка L_2.

Стоит заметить, что класс языков, описываемых грамматиками класса $LL(1) \cap FL$ в точности совпадает с классом всех конечных языков. И действительно, для любого конечного языка несложно построить $LL(1)$-грамматику. Однако полнота и исчерпываемость - это свойства системы преобразований в классе грамматик, а не в классе языков. Поэтому ничего странного в том, что AE в классе FL является исчерпывающей, а в классе $LL(1) \cap FL$ нет, несмотря на то, что классы языков этих классов грамматик совпадают.

Обозначим через RL – множество праволинейных грамматик ([1, стр. 111]) из Γ, в которых начальный символ не встречается в правых частях правил вывода.

Очевидно, любую праволинейных грамматику можно легко преобразовать так, чтобы она стала принадлежать RL.

Предложение 3.3. *AE является исчерпывающей в классе RL-грамматик.*

Доказательство. Для любой грамматики G_1 из класса RL, и любого регулярного языка L_2, таких что $L(G_1) \subseteq L_2$ язык $L_3 = L_2 \setminus L(G_1)$ является регулярным. По грамматике G_1 построим соответствующий НДКА A_1. Возмем любую грамматику G_3 из класса RL для языка L_3 и построим по ней соответствующий НДКА A_3.

Объединим начальные вершины A_1 и A_3. Легко показать, что мы получим автомат, распознающий язык $L(G_2)$. По этому автомату легко построить соответствующую регулярную грамматику.

Легко показать, что для любого принимаемого автоматом слова, существует единственный путь от начального состояния до одного из конечных. Это означает, что соответствующая грамматика будет однозначной. Можно показать, что в автомате не будет недостижимых и тупиковых вершин, а значит грамматика будет приведённой. Можно также показать, что степень захода начального состояния – 0. Всё это означает, что грамматика будет действительно из RL.

Осталось показать, что автомат A_1 с помощью последовательности преобразований из AE можно достроить до A_3. Но это легко сделать с помощью серии преобразований из *Add*.

Гипотеза 3.1. *Система преобразований AE является полной в $LL(1)$.*

Эту гипотезу автору не удалось ни доказать ни опровергнуть. Однако если она верна, то это означает, что система преобразований AE может использоваться для расширения грамматик довольно сложных языков.

3.2. Внутреннее представление

В задаче трансляции синтаксический анализатор занимается не только распознаванием корректных программ, но и построением некоторого внутреннего представления программы в качестве результата своей работы. Этот подраздел будет посвещен поиску удобного внутреннего представления и алгоритма синтаксического анализа, построенного на основе классического $LL(1)$-анализатора.

Для начала попробуем использовать в качестве внутреннего представления классическое дерево вывода. Для некоторого слова w языка $L(G)$ обозначим через $s(w)$ дерево вывода w в G.

Ниже приведено два примера, поясняющих, почему дерево вывода является не самым удачным внутренним представлением в рамках задачи расширения синтаксиса.

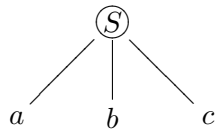
Пример 3.1. *Простейшее нарушение совместимости из-за цепных узлов, то есть узлов ровно с одним сыном.*

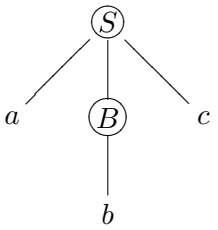
Рассмотрим две грамматики:

$$G_1 : S \to abc$$

$$G_2 : S \to aBd; B \to b$$

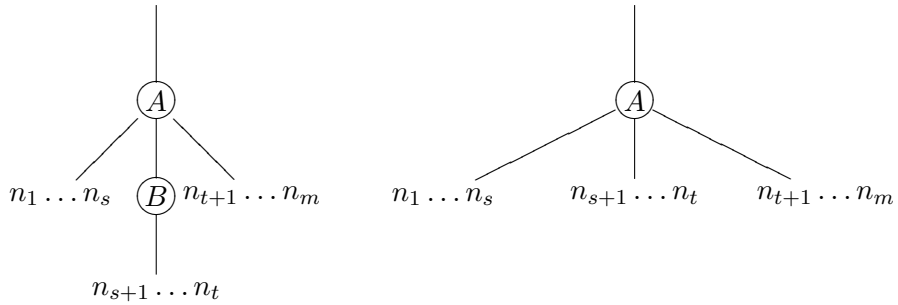
Очевидно, G_2 получается из G_1 применением преобразования из *Extract*. Однако видно, что $s_1(abc) \neq s_2(abc)$.





Таким образом расширение грамматики с помощью преобразований из AE приводит к несовместимости исходного и расширенного синтаксических анализаторов. Значит такой выбор устройства синтаксических анализаторов не подходит для конструирования расширяемых трансляторов.

Вырезание цепных узлов, показанным ниже образом решает этот класс проблем.



Пример 3.2. *Нарушение совместимости из-за ε-листьев.*

Ещё раз рассмотрим две грамматики:

$$G_1 : S \to a$$

$$G_2 : S \to aA; A \to \varepsilon$$

Аналогично G_2 получается из G_1 применением преобразования из $Extract$. Слово a содержится в обоих языках. А деревья вывода этого слова в G_1 и G_2 будут различными, даже если цепные узлы, раскрытые по правилу $A \to \varepsilon$ будут вырезаны из дерева вывода в G_2.

Далее описано внутреннее представление, в основе которого лежит классическое дерево вывода, и устройство соответствующего синтаксического анализатора, лишённое этих двух недостатков.

3.3. Размеченная грамматика

Пусть есть некоторая грамматика $G = (V, \Sigma, P, S)$ и p – это некоторое правило вывода из P. Введём несколько вспомогательных обозначений.

Через $len(p)$ будем обозначать длину правой части правила вывода p.

Через $p(i)$, при $1 \leq i \leq len(p)$, будем обозначать i-ый символ правой части правила вывода p.

Через $p(0)$ будем обозначать символ левой части правила вывода p.

Определение 3.11. *Разметкой грамматики G будем называть функцию m, которая каждому правилу вывода грамматики G будет ставить в соответствие кортеж, состоящий из нулей и единиц, длиной в количество символов в правой части своего аргумента. Другми словами для любого правила p $m(p)$ есть кортеж*

$$(m_1, m_2, \dots m_{len(p)}), \qquad \text{где} m_i \in \{0, 1\}$$

Кортеж $m(p)$ будем называть разметкой правила p.

Для удобства введём слудеющее обозначение для элементов кортежа:
$m(p) = (m(p, 1), m(p, 2), \dots m(p, len(p)))$.

Значение $m(p, i)$ будем называть разметкой i-ого символа правой части правила p.

Определение 3.12. *Определим множество размеченных грамматик Γ_M как множество всевозможных пар (G, m), где $G \in \Gamma$, а m – разметка G. Саму пару (G, m) будем называть размеченной грамматикой.*

Разметку правила вывода будем обозначать с помощью верхнего индекса у символов правой части правила вывода. Например для правила вывода $p : A \to BCde$ и разметки $m(p) = (0, 1, 0, 1)$, правило вывода будем обозначать так: $p : A \to B^0C^1d^0e^1$

Разметка нам потребуется для того, чтобы вырезать некоторые узлы из дерева вывода.

Каждому узлу кроме корня в классическом дереве вывода можно сопоставить значение разметки правила, в результате применения которого появился данный узел. Такое сопоставление будем называть разметкой узлов дерева вывода. Для единообразия доопределим разметку на корне дерева единицей. Чтобы пояснить идею разметки дерева вывода, обратимся к примеру:

Пусть дана следующая грамматика:

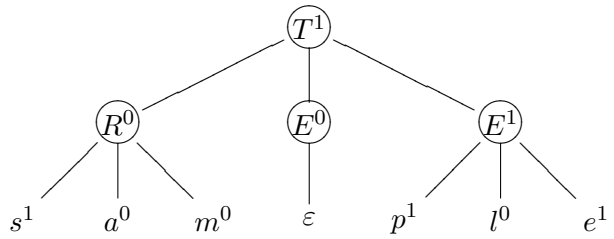
$$T \to R^0E^0E^1$$

$$R \to s^1a^0m^0$$

$$E \to \varepsilon$$

$$E \to p^1l^0e^1$$

Соответствующее дерево вывода слова «sample» в данной грамматике будет выглядеть так:



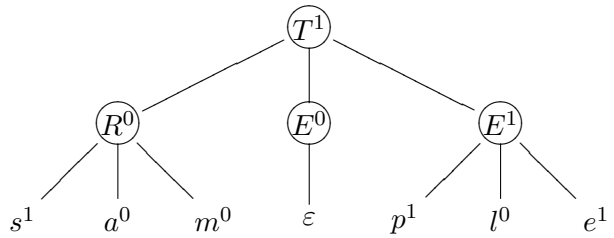
Определение 3.13. *Вспомогательным деревом вывода слова w в грамматике G, назовём дерево, полученное из обычного дерева вывода слова w в грамматике G, в результате удаления всех листьев, помеченных ε.*

Определение 3.14. *Сокращённым деревом вывода слова w в грамматике G, с разметкой m назовём дерево, полученное из вспомогательного дерева вывода слова w в грамматике G, путём вырезания из дерева всех узлов, размеченных нулями. Ниже дано определение вырезания узла из дерева.*

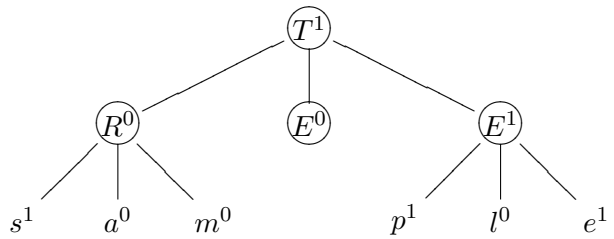
Определение 3.15. *Пусть у узла p_1 есть сыновья $n_1, n_2, \ldots n_k$. А у узла $n_i, (1 \leq i \leq k)$ есть сыновья $c_1, c_2, \ldots c_m$. Вырезанием узла n_i из дерева, называется операция замены в списке сыновей p_1 узла n_i на список своих сыновей $c_1, c_2, \ldots c_m$.*

Пример 3.3. *Сокращённое дерево вывода для дерева вывода из предыдущего примера.*

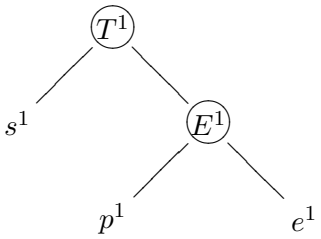
Исходное дерево вывода:



Вспомогательное дерево вывода:



Сокращённое дерево вывода:



Лемма 3.4. *Вырезание узлов при сокращении дерева можно выполнять в произвольном порядке.*

Для того, чтобы двигаться дальше, нужно ввести дополнительный термин «генератор синтаксических анализаторов».

Определение 3.16. *Пусть CT – это множество всех сокращённых деревьев вывода. Определим $ct\langle G, m\rangle$ как синтаксический анализатор, который слову ставит в соответствие его сокращённое дерево вывода в грамматике G с разметкой m.*

Будем обозначать ct функцию, которая размеченной грамматике ставит в соответствие синтаксический анализатор $ct\langle G, m\rangle$.

Приступим к изучению свойств функции ct.

Определение 3.17. *Пусть Φ – множество расширяющих преобразований грамматик из Γ. Будем говорить, что Φ – безопасная для ct система преобразований если для любой размеченной граматики (G_1, m_1) и любого преобразования $\phi \in \Phi$ существует такая разметка m_2 грамматики $G_2 = \phi(G_1)$, что выполняются следующие условия:*

1. $m_1\big|_{P_1 \cap P_2} = m_2\big|_{P_1 \cap P_2}$;
2. $ct\langle G_2, m_2\rangle\big|_{L(G_1)} = ct\langle G_1, m_1\rangle$.

Другими словами, каким бы образом безопасная система преобразований не изменила грамматику, всегда удастся доопределить разметку на новых правилах так, чтобы соответствующие синтаксические анализаторы оказались совместимы. Кроме того, первое условие нам гарантирует, что новая разметка будет отличаться от старой только на новых правилах. Это означает, что изменение разметки будет носить локальный характер.

Ниже будет показано, что система преобразований AE^Γ является безопасной системой для ct. Этот факт и является главным результатом данной работы.

Лемма 3.5. *Система преобразований AE является безопасной для ct.*

Доказательство. Возьмём произвольную размеченную грамматику (G_1, m_1) из Γ_M и ϕ из $AE = Add \cup Extract$. Пусть $G_1 = (V_1, \Sigma_1, P_1, S)$. Рассмотрим два случая.

Случай 1. $\phi \in Add$. Для любого слова w из языка $L(G_1)$ рассмотрим его дерево вывода в грамматике G_1. Очевидно, это дерево также будет являться деревом вывода слова w и в грамматике $G_2 = \phi(G_1) = (V_1, \Sigma_2, P_2, S)$. Следовательно, вспомогательное дерево вывода слова w в грамматике G_1 будет также являться вспомогательным деревом вывода w в G_2. Рассмотрим произвольную разметку m_2 грамматики G_2 такую, что $m_2\big|_{P_1} = m_1$. Поскольку во вспомогательном дереве вывода присутствуют лишь узлы, помеченные правилами вывода из P_1, на которых разметка сохранилась, то сокращённое дерево вывода w в (G_1, m_1) совпадёт с сокращённым деревом вывода w в (G_2, m_2), то есть $ct\langle G_1, m_1\rangle(w) = ct\langle G_2, m_2\rangle(w)$.

Случай 2. $\phi \in Extract$. Для любого слова w из языка $L(G_1)$ рассмотрим его вспомогательное дерево вывода слова w в грамматике G_1. Обозначим через $Nodes$ множество всех узлов этого дерева и введём на этом множестве три функции:

1. $Sym : Nodes \to \Sigma_1 \cup V_1$ такую, что узел $n \in Nodes$ помечен символом $Sym(n)$.

2. $Sons$, которая каждому узлу $n \in Nodes$ ставит в соответствие упорядоченный кортеж узлов, являющихся сыновьями n.

3. $Prod : \{n \in Nodes | Sym(n) \in V_1\} \to P_1$ такую, что $Prod(n)$ – это правило вывода, по которому был раскрыт узел n.

Пусть ϕ удаляет правило $q_0 : A \to \alpha\beta\gamma$ и добавляет вместо него правила $q_1 : A \to \alpha B\gamma$ и $q_2 : B \to \beta$.

Для каждого элемента $\bar{n}$ из множества $Nodes(q_0) = \{n \in Nodes | Prod(n) = q_0\} \neq \emptyset$ проделаем следующую операцию:

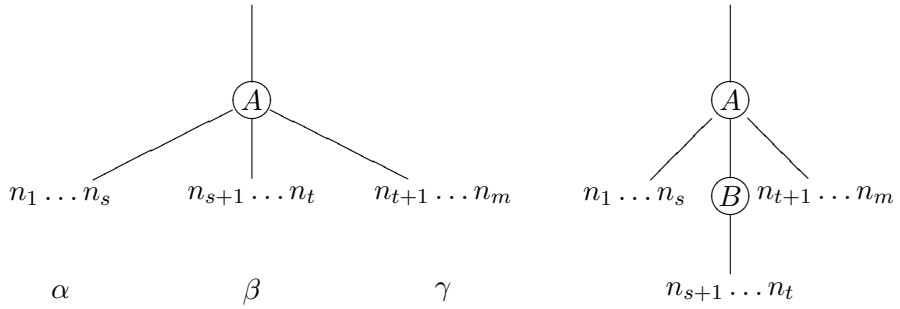
Пусть $Sons(\bar{n}) = (n_1, \dots n_m)$. Тогда существуют такие два целых числа s, t $(0 \leq s \leq t \leq m)$, что

$$Sym(n_1) \dots Sym(n_s) = \alpha$$

$$Sym(n_{s+1}) \dots Sym(n_t) = \beta$$

$$Sym(n_{t+1}) \dots Sym(n_m) = \gamma$$

Создадим новый узел $\dot{n}$, помеченный Y, и все узлы $n_{s+1} \dots n_t$, вместе со своими поддеревьями, сохраняя порядок, сделаем сыновьями $\dot{n}$. А у узла $\bar{n}$ удалим их из списка сыновей, а на их место добавим узел $\dot{n}$. Схематично эту операцию можно представить следующим образом:



Несложно заметить, что после того, как все узлы, раскрытые по правилу вывода q_0, будут преобразованы описанным образом, мы получим вспомогательное дерево вывода слова w в грамматике $G_2 = \phi(G_1) = (V_2, \Sigma_1, P_2, S)$. И действительно, слово, читающееся слева направо по листьям, помеченным терминалами, не изменилось. Узлы не из $Nodes(q_0)$ мы не изменяли и можно считать, что они были раскрыты по правилам вывода из $P_1 \setminus \{q_0\} \subset P_2$. Узлы из $Nodes(q_0)$ после изменения, можно считать раскрытыми по $q_1 \in P_2$, а вновь созданные узлы, можно считать расрытыми по $q_2 \in P_2$.

Рассмотрим произвольную разметку m_2 граматики G_2 такую, что

1. $m_2\big|_{P_2 \setminus \{q_1, q_2\}} = m_1\big|_{P_1 \setminus \{q_0\}}$
2. $\forall i \leq s \Rightarrow m_2(q_1, i) = m_1(q_0, i)$
3. $m_2(q_1, s+1) = 0$
4. $\forall i \geq 1 \Rightarrow m_2(q_1, s+1+i) = m_1(q_0, t+i)$
5. $m_2(q_2, i) = m_1(q_0, s+i)$

Рассмотрим вспомогательное дерево вывода w в G_1 и полученное из него вспомогательное дерево вывода w в G_2. При сокращении этих двух деревьев, после добавления искусственного корня, первым делом удалим все узлы, раскрытые по q_2 (это сделать можно по лемме 3.4). Очевидно, после этого оба дерева станут равны, а разметки совпадут на всех правилах, по которым был раскрыт хотя бы один узел, оставшийся в деревях. Но в этом случае, дальнейшее сокращение будет проходить одинаково в обоих деревьях. А это значит, что сокращённое дерево вывода w в (G_2, m_2) совпадает с сокращённым деревом вывода w в (G_1, m_1). Другими словами $ct\langle G_1, m_1\rangle(w) = ct\langle G_2, m_2\rangle(w)$.

Теорема 3.1. *Система преобразований AE^{Γ} является безопасной для ct.*

Доказательство. Возьмём произвольную размеченную грамматику (G_1, m_1) из Γ_M и произвольное преобразование $\phi = \phi_n \circ \cdots \circ \phi_2 \circ \phi_1$ из AE^{Γ}. Введём обозначения:

$$G_1 = (V_1, \Sigma_1, P_1, S);$$

$$G_{i+1} = (V_{i+1}, \Sigma_{i+1}, P_{i+1}, S) = \phi_i(G_i), i = 1..n.$$

Согласно лемме 3.5, существуют такие разметки $m_i : P_i \to \{0, 1\}, i = 2..n+1$, что

$$m_2\big|_{P_1 \cap P_2} = m_1$$

$$m_3\big|_{P_2 \cap P_3} = m_2$$

$$\cdots$$

$$m_{n+1}\big|_{P_n \cap P_{n+1}} = m_n$$

При этом для любого i из промежутка от 1 до n $ct\langle G_{i+1}, m_{i+1}\rangle$ совместим с $ct\langle G_i, m_i\rangle$.

Докажем, что $ct\langle G_{n+1}, m_{n+1}\rangle$ совместим с $ct\langle G_1, m_1\rangle$. Для этого достаточно показать, что $m_{n+1}\big|_{P_1 \cap P_{n+1}} = m_1$.

Заметим, что из определния m_i можно заключить, что $m_{n+1}\big|_{P_1 \cap P_2 \cap \ldots \cap P_n} = m_1$. Рассмотрим множество $P_1 \cap P_{n+1} \setminus (P_2 \cap \ldots \cap P_n)$. Докажем, что это множество пусто. Очевидно, что этого достаточно для завершения доказательства теоремы.

Предположим обратное. Тогда существует правило вывода $q \in P_1 \cap P_{n+1} \setminus (P_2 \cap \ldots \cap P_n)$. Обозначим $k = max\{i | 1 < i < n+1, q \notin P_i\}$. Тогда $q \in P_{k+1} \setminus P_k$.

Найдём любое слово $w \in L(G_1)$ такое, что в его выводе в грамматике G_1 присутствует применение правила q, преобразующее α в β:

$$S \Rightarrow \cdots \Rightarrow \alpha \Rightarrow \beta \Rightarrow \cdots \Rightarrow w$$

По лемме 3.2 в грамматике G_k, вывод слова w будет иметь вид:

$$S \Rightarrow \cdots \Rightarrow \alpha \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_r \Rightarrow \beta \Rightarrow \cdots \Rightarrow w$$

Причём $r > 0$, так как $q \notin P_k$.

По лемме 3.2 в грамматике G_{k+1}, вывод слова w будет иметь вид:

$$S \Rightarrow \cdots \Rightarrow \alpha \Rightarrow \bar{\alpha_1} \Rightarrow \bar{\alpha_2} \Rightarrow \cdots \Rightarrow \bar{\alpha}_{\bar{r}} \Rightarrow \beta \Rightarrow \cdots \Rightarrow w$$

Причём $\bar{r} \geq r > 0$.

Однако поскольку $q \in P_{k+1}$, то существует также другой, отличный от данного, вывод слова w в G_{k+1}:

$$S \Rightarrow \cdots \Rightarrow \alpha \Rightarrow \beta \Rightarrow \cdots \Rightarrow w$$

Что противоречит однозначности грамматики G_{k+1}. А значит действительно $P_1 \cap P_{n+1} \setminus (P_2 \cap \ldots \cap P_n) = \emptyset$.

4. Заключение

В качестве резюме, перечислены ключевые факты данной работы.

Была предложена простая, но при этом довольно выразительная система приобразований грамматик AE^{Γ}.

Были предложены концепции размеченной грамматики, синтаксического анализатора ct и сокращённых деревьев вывода. Синтаксический анализатор управляется размеченной грамматикой и выдаёт в качестве результата своей работы сокращённые деревья вывода.

Показано, что при применении к размеченной грамматике преобразований из AE^{Γ}, разметку у полученной в результате грамматики можно доопределить на новых правилах вывода так, чтобы синтаксический анализатор после применения к его грамматике преобразования из AE^{Γ} оставался совместимым с исходным синтаксическим анализатором.

Локальность изменения разметки при расширении понадобится, когда встанет вопрос об одновременном применении нескольких независимых преобразований одной и той же грамматики. Тот факт, что каждое из преобразований меняет разметку лишь локально, уменьшит количество конфликтов, в которых несколько преобразований претендует на изменение одних и тех же правил вывода.

Литература

1. Ахо А. Ульман Дж. Теория синтаксического анализа, перевода и компиляции. Том 1. Синтаксический анализ. М.: Мир, 1978.
2. Sheard T. Jones S. P. Template meta-programming for Haskell. In Proceedings of the Haskell workshop, pp. 1–16. ACM Press, 2002.
3. Moskal M. Olszta P. W. Skalski K. Nemerle. Introduction to a Functional .NET Language, www.nemerle.org.
4. Skalski K. Syntax-extending and type-reflecting macros in an object-oriented language. Master Thesis, Institute of Computer Science University of Wroclaw, 2005.
5. Wilson G. V. Extensible Programming for the 21st Century. ACM Queue, Programming languages, Vol. 2, No. 9 – Dec/Jan 2004–2005.