ЯЗЫКИ ПРОГРАММИРОВАНИЯ С РАСШИРЯЕМЫМ СИНТАКСИСОМ

Егоров П.В.
e-mail: xoposhiy@mail.ru

Чем легче и гибче язык программирования (ЯП) позволяет расширять свои возможности, тем он более удобен для применения в крупных проектах. Какие же способы расширения предлагают на данный момент наиболее популярные ЯП? Прежде всего, расширение с помощью библиотек функций. Кроме того, объектно-ориентированные ЯП предлагают возможность расширения типов. В данной работе рассматривается альтернативный способ — расширение синтаксиса ЯП. То есть фактически, расширение самого транслятора.

Существуют впечатляющие примеры успешных, гибко расширяющихся программных продуктов, например, интернет пейджер Miranda IM, или среда разработки Eclipse. Богатство функциональных возможностей в них сочетаются с большой гибкостью добавления новых возможностей. Расширение возможностей производится путём создания и подключения к ядру программы дополнительных модулей. Причём благодаря опубликованному интерфейсу создавать дополнительные модули могут сторонние разработчики и вообще любые желающие. Почему бы не попробовать такой же подход к конструированию трансляторов?

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

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

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

2. Многие проверки переносятся с этапа выполнения на более ранний этап компиляции, уменьшая тем самым стоимость исправления ошибок. В нашем примере, это случилось с проверкой корректности sql-запроса.

3. Всё это даёт возможность гибко поддерживать множество технологий и парадигм прямо на уровне синтаксиса языка.

4. При этом 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.

Преимущества достаточно внушительные, чтобы браться за дальнейшее развитие идеи. Рассмотрим один из подходов к задаче создания расширяемого транслятора.

Обычно процесс трансляции разбивают на несколько последовательных этапов. Создание ЯП с расширяемым синтаксисом означает превращение каждого такого этапа в расширяемый. Сегодня выделение в качестве двух первых этапов трансляции лексического и синтаксического анализов стало стандартом де-факто. Дело в том, что создание синтаксических и лексических анализаторов (СА и ЛА) является хорошо изученной, проработанной и автоматизированной задачей. В частности, существуют готовые инструменты для создания ЛА и СА по описанию грамматики языка с помощью форм Бекуса-Науера. Примерами могут служить продукты `LEX`, `YACC`, `ANTLR`. Таким образом, для разработки расширяемых алгоритмов ЛА и СА существует мощная теоретическая и алгоритмическая база.

Постановка задачи конструирования расширяемой версии ЛА выявляет следующие задачи: разработать алгоритмы расширения набора распознаваемых лексем на этапе выполнения транслятора; диагностирования конфликтов, возникающих при расширении; их автоматического разрешения в тех ситуациях, когда это возможно. Как показало исследование, конструирование приемлемого расширяемого ЛА на базе известных алгоритмов не представляет принципиальных теоретических сложностей и является задачей скорее технического плана, нежели теоретического.

Гораздо больший интерес представляет разработка алгоритма расширяемого СА. Учитывая, что обычно синтаксис языка задаётся его грамматикой, задачу СА можно определить так: руководствуясь описанием грамматики языка по последовательности лексем построить дерево разбора. Задачи при конструировании расширяемого СА стоят практически те же, что и при конструировании расширяемого ЛА: разработать алгоритмы модификации (расширения) грамматики распознаваемого языка на этапе выполнения транслятора; диагностирования конфликтов, возникающих при расширении; их автоматического разрешения в тех ситуациях, когда это возможно.

При расширении важно, чтобы расширенная грамматика оставалась совместимой с исходной. То есть расширенный СА должен преобразовы-

вать любые программы исходного языка точно в то же дерево разбора, что и исходный СА. Проще говоря, после расширения транслятора, старые программы должны продолжать работать. Вообще говоря, проверка этого условия в общем случае является алгоритмически неразрешимой задачей. Однако расширение грамматики можно производить серией последовательного применения безопасных, то есть не нарушающих условие совместимости, преобразований. Задача состоит в том, чтобы найти удобную систему таких безопасных преобразований.

Для исследования задачи, изначально был выбран наиболее простой класс грамматик, приемлемых для определения синтаксиса ЯП — класс LL(1) грамматик. Для описания грамматики языка, была разработана специальная нотация, представляющая собой модификацию форм Бекуса-Науера. Данная модификация к правилам вывода (которые фактически представляют собой инструкции СА по выводу) добавляла также инструкции по построению дерева разбора (результата работы СА). Затем для LL(1) грамматик, записанных в данной нотации, была найдена удобная система безопасных преобразований: каждое преобразование из этой системы с одной стороны заведомо преобразует любую LL(1)-грамматику в совместимую, а с другой стороны не выводит грамматику из класса LL(1).

При поиске системы безопасных преобразований главным был вопрос о её выразительной силе. Другими словами, на сколько сильно можно расширить синтаксис языка, последовательно применяя данные преобразования? Подступиться к этому вопросу с чисто теоретической стороны оказалось достаточно сложно. Поэтому для демонстрации выразительной силы найденной системы преобразований был разобран пример создания и расширения модельного языка.

В качестве такого модельного языка был выбран язык `Pascal-S` — упрощённый `Pascal`, разработанный Николаусом Виртом как раз для того, чтобы на его примере обучать построению компиляторов. Он отличается от классического языка `Pascal` отсутствием меток, указателей и вариантных записей. Все синтаксические конструкции были разбиты на ядро и 7 его расширений:

1. Ядро. Определение понятия блока (последовательности операторов, заключённой в `begin-end`). Определение базовых понятий: переменная, присваивание, вызов подпрограммы.

2. Поток исполнения. Определение операторов управления потоком исполнения: `if`, `while`, `for`, `repeat-until`, `case`.

3. Типы. Вспомогательные определения, используемые в последующих расширениях системы типов.

4. Массивы (зависит от расширения “Типы"). Определение типа данных “Массив” и операции доступа к элементу массива.

5. Записи (зависит от расширения “Типы"). Определение типа данных “Запись” и операции доступа к полю записи.

6. Арифметика. Определение синтаксиса арифметических выражений и операций.

7. Константы и типы. Определение блоков констант и типов.

8. Подпрограммы (зависит от расширения “Блоки"). Определение “процедур"и “функций".

Таким образом, имея изначально некий очень простой и строгий язык, определяемый в ядре, можно, используя найденную систему преобразований грамматики, расширить этот язык, добавив в него любую из существующих в `Pascal-S` синтаксических конструкций. В частности, можно добавить и все конструкции сразу, получив при этом язык `Pascal-S`. На самом деле точно так же можно было разработать некоторое расширение, которое добавляло бы в наш язык возможность, отсутствующую в `Pascal-S`, но содержательность примера от этого бы не изменилась.

Однако на пути от идеи ЯП с расширяемым синтаксисом до работоспособной реализации этой идеи находится ещё целый ряд задач. Из них хочется отметить следующие наиболее интересные с теоретической точки зрения направления:

Формализация вопроса о богатстве и выразительной силе системы безопасных преобразований. Последующий анализ найденной системы преобразований, основывающийся на данной формализации. Разработка способов диагностики конфликтов, возникающих при расширении ЛА и СА, а также способов их автоматического разрешения, в тех случаях, когда это возможно. Разработка технологии построения дальнейших этапов трансляции в ключе расширяемости. Что касается последнего направления, то некоторые результаты в этой области уже есть и реализованы в исследовательском продукте `Meta Programming System` компании `JetBrains`.

Список литературы

[1]. *A. Brugermann-Klein* Regular Expressions into Finite Automata. Proceedings of Latin ’92, 1992.

[2]. *N. Wirth* PASCAL-S: A Subset and its Implementation. Juny 1975.

[3]. Интернет страница проекта Meta Programming System компании JetBrains. http://www.jetbrains.com/mps/