ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ

Обзорная статья

УДК 004.4
DOI: 10.18101/2304-5728-2021-4-3-18

ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА В ЯЗЫКОВЫХ IT–ТЕХНОЛОГИЯХ

© Федорченко Людмила Николаевна
кандидат технических наук, старший научный сотрудник,
Санкт–Петербургский федеральный исследовательский центр
Российской академии наук (СПб ФИЦ РАН)
Россия, 199178, г. Санкт–Петербург, 14 Линия В.О., 39
lnf@iias.spb.su

Аннотация. В статье представлен перечень инструментальных средств, разработанных или находящихся в процессе разработки, используемых в системах программирования в лабораториях и на кафедрах СПбГУ. Перечислены основные технологии, применяемые при разработке инструментальных платформ для современных систем реализации языков программирования, которые сведены в таблицы. Приведены примеры правил грамматик для инструментальных систем CDL, Форт, ANTLR и образцы автоматически получаемого кода для инструмента *PEG in RUST* и интегрированной среды разработки *JetBrains Grammar–Kit.* Более подробно рассмотрена инструментальная система SynGT эквивалентных преобразований правил контекстно–свободных грамматик с целью их регуляризации. В таблицах даны характеристики тех инструментальных систем, которые находятся в открытом доступе и распространяются бесплатно. С момента освещаемой ситуации минуло 30 лет. Тем не менее, читатель наверняка найдет полезным эту навигационную схему, дающую ключ к изучению деятельности различных групп разработчиков в области программирования как в России, так и за рубежом.

Ключевые слова: системы построения трансляторов, предметно–ориентированные языки, контекстно–свободные грамматики в регулярной форме.

Благодарности

Работа выполнена при участии студента математико-механического факультета СПбГУ Небогатикова Ивана Юрьевича.

Для цитирования

Федорченко Л. Н. Инструментальные средства в языковых IT–технологиях // Вестник Бурятского государственного университета. Математика, информатика. 2021. № 4. С. 3–18.

Введение

В настоящее время языковые технологии активно включаются в различные сферы нашей жизни, а именно, в комплексные системы производственной сферы при формулировании целей и проблем для принятия решений, при сопоставлении образцов в анализе ДНК (дезоксирибонуклеиновая кислота), в задачах компьютерной безопасности и многих других. Это привело к развитию современных транслирующих систем (трансляторов, компиляторов, интерпретаторов), использующих принцип синтаксического управления для производства программ обработки данных и ориентированных на разнообразный набор вычислительных устройств, используемых не только в производстве, но и в быту.

Процесс обработки данных в такой технологии управляется синтаксической структурой этих данных. Управляющие структуры традиционно описываются посредством грамматических правил или графов, представляющих эти правила. Чаще всего для этой цели используются контекстно – свободные, автоматные грамматики или их обобщения, основанные на регулярной модели.

Актуальность разработки такой технологии обработки данных в системе построения трансляторов на современной основе вытекает из следующего:

- увеличение ассортимента вычислительных устройств и требований к функционированию программного обеспечения (ПО) в различных системах искусственного интеллекта;
- сокращение сроков разработки как следствие частого выпуска обновленных моделей, адаптируемых к модернизируемым вычислительным системам и конкурентности в рыночной экономике;
- огромное разнообразие языков (более 10 тыс. новых на текущую дату), применяемых не только в общесистемном ПО, но и во многих предметных областях, и в производстве;
- неоднородность спецификаций реализуемых языков (метаязыки БНФ (Бэкус–Наур Формы) и extended БНФ, двухуровневые и аффиксные грамматики, регулярные выражения и прочие);
- использование формальных методов на всех этапах разработки приложений позволяет автоматизировать процесс создания и сопровождения систем для сокращения сроков разработки и повышения надежности ПО;
- применение автоматизации в предварительной обработке синтаксиса реализуемого языка, как правило, обеспечивает существенное сокращение трудозатрат и времени разработки фазы синтаксического анализа, которая занимает до 25% ресурсов при создании языкового процессора.

1 Инструментарий, применяемый в разработке языковых процессоров

При разработке любой транслирующей системы, основанной на грамматике входного языка, необходимо решать следующие три задачи:

- *упрощение грамматики(приведение к well–formed);*
- *регуляризация грамматики (эквивалентные преобразования);*
- *разрешение конфликтных ситуаций (недетерминизм),* возникающих в процессе построения анализатора и связанные с языковой неоднозначностью и недетерминированностью магазинного автомата.

Для этой цели используются различные технологии разработки трансляторов, наиболее распространенными из которых являются средства, основанные на применении различных подклассов контекстно–свободных грамматик для описания входного языка.

1.1 Compiler Description Language (CDL)

Compiler Description Language — технология С.Н.А. Костера (аффиксные грамматики [9]). Технология имеет три версии, имеющих значительные различия в дизайне, но мало отличающиеся в функциональности. Последняя версия языка совмещает преимущества синтаксических формализмов и языков программирования. Структуры данных были спроектированы таким образом, чтобы упростить реализацию анализаторов языка. Однако, его применение не ограничивается использованием для создания компиляторов, также возможно применение в синтаксически управляемых приложениях, например, для межпроцессорных коммуникаций, использующих протоколы. Пример задаваемого правила в системе CDL3 [17], описывающего уникальность ключей в списке, показан на рисунке 1. Нетерминальные аффиксы обозначаются заглавными буквами. Они задаются с вместе с правилом, использующим нетерминальные аффиксы и выражения, состоящие из терминальных аффиксов и переменных. Примеры даны в работе [2; 17].

1.2 Lex/Yacc, Flex/Bison

1.2.1 Историческая справка

Lex/Yacc, Flex/Bison – семейство компиляторов GNU[1] [11].

GNU Flex (Fast Lexical Analyzer Generator) — свободно распространяемая версия генератора лексических анализаторов Lex. GNU Bison — это свободно распространяемая замена Yacc (Yet Another Compiler Compiler), – программа автоматической генерации синтаксических анализаторов по данному описанию грамматики.

Lex и Yacc были разработаны в Bell Laboratories в 70–х годах Стефаном Джонсоном (Stephen C. Johnson) и Майком Леском (Mike Lesk) в качестве стандартных инструментов в составе операционной системы Unix. Они являются коммерческими продуктами. Позже Фондом свободного

[1] GNU-свободная Unix-подобная операционная система

программного обеспечения (FSF) в рамках проекта GNU были созданы их бесплатные свободно распространяемые аналоги – Flex и Bison. По сути, Flex – это переписанный Джефом Посканцером (Jef Poskanzer), генератор лексических анализаторов Lex с устраненными недостатками. Аналогично Bison – это переписанный и расширенный Робертом Корбетом (Robert Corbett), Ричардом Столлменом (Richard Stallman) и Чарльзом Донелли (Charles Donnelly) генератор синтаксических анализаторов Yacc.

1.2.2 Bison

Bison преобразует описание контекстно–свободной грамматики с ограничениями (*LALR*–грамматики) *L*ook *A*head *L*eft–to–right *R*ightmost грамматики в программу на языке программирования C для разбора этой грамматики.[2] Именно эти ограничения на класс КС–грамматик создают конфликтные ситуации в процессе синтаксического разбора входных программ. Bison совместим с Yacc, то есть, все правильные грамматики для Yacc должны без изменений работать с Bison, обладающий более широкими возможностями и справляющийся как с анализом простых арифметических выражений, так и с разбором текстов реальных C–программ. Последние версии Bison используются как инструмент при создании GCC — семейства компиляторов GNU C для различных архитектурных платформ и современных операционных систем.

Bison — средство, обладающее развитыми возможностями, простое в использовании, с открытыми исходными текстами, бесплатное, поддерживаемое энтузиастами, существует на многих платформах, давно и широко используется, снабжено полной и понятной документацией. Среди недостатков Bison можно выделить работу только с грамматиками *LALR*(1) (ограничение всего множество *LR*(1)–грамматик), генерацию кода на языке C (хотя есть версии, способные генерировать текст анализатора на других объектных языках), в реализации анализатора используются глобальные переменные (их использование можно исключить специальной опцией), использование процедурно–ориентированного интерфейса (не объектно–ориентированный подход, отсюда небезопасность, засорение глобального пространства имен). Многие минусы обусловлены языком C, на котором генерируется анализатор. Версии, генерирующие вывод на других объектных языках свободны от большинства таких недостатков, но являются платными.

1.3 Eli

Eli [13] — коллективная разработка нескольких университетов (США), предназначенная для уменьшения стоимости разработки различных компиляторов. Среда состоит из набора отдельных инструментов и одной управляющей программы, что позволяет гибко использовать модули, ис-

[2] Bison – GNU parser generator, http://www.gnu.org/software/bison/

правлять или заменять их. Eli подходит для расширения существующих языков, поскольку возможно использование грамматик других систем, которые генерируют требуемый компилятор. Распознавание только *LALR*(1) языков и генерация кода только для языка С значительно ограничивают применение данной системы. Вследствие данных ограничений среда используется только в проектах, требующих поддержки совместимого кода.

1.4 Форт–технология

Технология, разработанная и применяемая С.Н. Барановым [14], и британской группой разработчиков. Система программирования Форт предоставляет полную систему исполнения кода: операционную систему, интерпретатор для диалогового исполнения, компилятор, ассемблер, текстовый редактор и обслуживающие программы. Кроме этого, Форт является метасистемой, т. е. системой, описывающей саму себя. В Форт–системе используется минимум правил и накладывается минимум ограничений, т. е. она обходится почти без синтаксиса и жестко контролируемых интерфейсов для взаимодействия модулей. Это обеспечивает максимум возможностей для программиста по модификации системы, сохраняющей ее относительную независимость от аппаратуры. К недостаткам системы относятся сложность поддержания программ из–за сложности чтения текстов программ и отсутствие контроля типов. Пример разбора формата даты на языке Форт, состоящей из дня, месяца и года, показан на рисунке 3.

1.5 ANTLR

ANTLR (ANother Tool for Language Recognition) [6] — среда, используемая для генерации парсера по заданной грамматике, визуализирующая процесс разбора входных данных с помощью деревьев. Пример грамматики, описывающей операции с целыми числами в Java показан на рисунке 3.

ANTLR принимает на вход контекстно–свободную грамматику, не содержащую неявную или скрытую левую рекурсию (скрытая левая рекурсия возникает при порождении нетерминалом пустого символа, например, $A \rightarrow BA$, $B \rightarrow \varepsilon$). Синтаксис грамматик похож на грамматики в Yacc, и использует форму extended БНФ. Результатом работы с грамматикой является лексический анализатор (система преобразования входной последовательности символов в лексемы, токены) и парсер (программа, осуществляющая формирование выражений языка из токенов) на языке Java или C#, что позволяет программистам, работающим с результатом преобразования, получать и обрабатывать информацию о процессе разбора с использованием интерфейса используемого языка программирования. ANTLR поддерживает использование семантических предикатов, функций на языке программирования, предназначенных для определения семантической валидности продукций (правил грамматики), обозначающих

соответствие разобранного выражения синтаксису грамматики, что позволяет управлять процессом построения анализатора. Процесс удаления левой рекурсии имеет сложность *O(n4)* [23].

1.6 Язык Python

Анализаторы языка Python, используют грамматики из класса *LL*(1) и *PEG* (Parsing Expression Grammar) [7]. Пример части грамматики для распознавания действий в языке Python, показан на рисунке 4. В отличие от контекстно–свободных грамматик, грамматики PEG более точно задают процесс разбора предложения языка с помощью упорядочивания применяемых правил при использовании оператора выбора. Например, для правила “A;B;C” *LL*(1)–анализатор будет проверять все возможные валидные комбинации, порождаемые правилами. PEG–парсер будет использовать второе правило, в том случае если генерация с использованием первого правила закончилась неудачей, и третье — если первые два правила не валидны для заданной строки. Такой подход с перебором позволяет частично избежать конфликтных ситуаций при разборе, появляющихся при использовании КС–грамматик. Наивная реализация PEG–парсера имеет экспоненциальное время разбора, поскольку предпросмотр выражения не ограничен, как при использовании *LL*(1) грамматик, однако использование оптимизаций позволяет сократить сложность разбора до линейного.

1.7 LLLPG

LLLPG [12] — генератор анализаторов (парсеров) для языка C#, поддерживает класс *LL*(k) грамматик. Создан для улучшения качества получаемого кода по сравнению с результатами работы ANTLR. Использование платформо–зависимых оптимизаций для языка C# и среды Microsoft Common Language Runtime позволяет улучшать получаемый код для чтения и по производительности: скорости разбора выражений и работы с памятью. Также, в отличие от ANTLR, данная среда не предназначена для работы в качестве отдельного приложения, и разрабатывалась для встраивания в другие языки программирования, и имеет консольный интерфейс для взаимодействия с пользователем. Поскольку система предназначена для работы только с *LL*(k) грамматиками, в процессе работы с языками не возникает конфликтных ситуаций.

```
FUNCTION enter (>ALIST>,>KEY,>ELEMENT):
[ALIST–>KEY1 ELEMENT1 and ALIST1],
([KEY=KEY1],
[KEY ELEMENT and ALIST1–>ALIST];
enter(ALIST1,KEY,ELEMENT),
[KEY1 ELEMENT1 and ALIST1–>ALIST]);
[KEY ELEMENT and ALIST–>ALIST].
```

Рис. 1. Пример описания правила в системе CDL

```
: year? \ year — t / f
  1752 2050 within?     ;

: month? \ month — t/f
  1 12 within?     ;

: day? \ day year month — t|f
  >days 1 swap within?     ;

: valid? \ day month year — t|f
  dup year?
  if over month?
      if swap day?
      else 2drop drop 0
      then
 else 2drop drop 0
 then
;
```

Рис. 2. Пример описания правила в языке Форт

```
grammar Count;

@header {
   package foo;
}

@members {
   int count = 0;
}

list
@after {System.out.println(count+" ints");}
: INT {count++;} (',' INT {count++;} )*
;

INT : [0–9]+ ;
WS : [ \r\t\n]+ –> skip ;
```

Рис. 3. Пример грамматики в системе ANTLR

```
action[str]: "{" ~ target_atoms "}" { target_atoms }

target_atoms[str]:
   | target_atom target_atoms { target_atom + " " + target_atoms }
   | target_atom { target_atom }

target_atom[str]:
   | "{" ~ target_atoms "}" { "{" + target_atoms + "}" }
   | NAME { name.string }
   | NUMBER { number.string }
   | STRING { string.string }
   | "?" { "?" }
   | ":" { ":" }
```

Рис. 4. Часть грамматики, разработанной для PEG–генератора кода

1.8 PEG in Rust

Parsing Expression Grammars in Rust [8] — система для построения шаблонов правил парсеров, основанных на грамматиках PEG.

Правила в системе задаются в виде выражения “rule NAME(PARAMETERS) —>RETURN_TYPE = PEG_EXPR”, где PEG_EXPR это PEG–выражение, описывающее то, как входные параметры создают результат. Пример построения парсера для разбора массивов показан на рисунке 6. Для задания парсера используется макрос языка Rust, в котором дается набор синтаксических правил для порождения языка. Настройка работы с языком производится в текстовом редакторе. В данной системе этапы упрощения и регуляризации грамматики не применяются.

1.9. JetBrains Grammar–Kit[3]

JetBrains Grammar–Kit [15] — расширение для интегрированной среды разработки IntelliJ IDEA, предназначенное для реализации языков. Система принимает описание грамматики языка в форме БНФ и преобразует в анализатор на языке Java. Взаимодействие со средой разработки позволяет визуализировать процесс работы с грамматикой. Grammar–Kit используется как генератор анализаторов языков Dart и Perl5 в средствах разработки различных приложений в лаборатории JetBrains. Пример генерируемого кода оператора упорядоченного вывода показан на рисунке 7. Для правила генерируется код, последовательно проверяющий части правила. Если ни одно правило не было применено, состояние разбора возвращается в начальное состояние.

1.10. QReal, REAL.NET

Среды QReal [18] и REAL.NET [19], разрабатываемые на кафедре системного программирования СПбГУ, являются metaCASE инструментами, то есть предоставляют возможность создания новых визуальных языков и интегрированных сред разработки для них. Например, с использованием платформы QReal была построена среда QReal:Robots [20] для визуального программирования роботов Lego Mindstorms NXT 2.0, а также среда TRIK Studio [21] –для обучения робототехнике. QReal и REAL.NET используют различные модели описания визуального языка. QReal применяет метамодели, задаваемые в визуальных средах, которые описывают множество корректных моделей предметно–ориентированного языка. Среда REAL.NET использует подход глубокого метамоделирования [22], в котором сущность модели рассматривается одновременно как тип, и как экземпляр некоторого типа.

[3] **JetBrains** (ранее — IntelliJ) — международная компания, которая разрабатывает инструменты для разработки на языках Java, Kotlin, C#, F#, C++, Ruby, Python, PHP, JavaScript и многих других

1.11. WebDPF

WebDPF [22] — основанная на DPF (Diagram Predicate Framework) [23] среда для проектирования предметно–ориентированных языков. Платформа использует подход многоуровневого метамоделирования и позволяет задавать ограничения на разрабатываемый язык с использованием предикатов, которые должны выполняться на требуемых подграфах задаваемого визуального языка. Среда поддерживает подход авто дополнения частично реализованных моделей, что позволяет ускорить процесс моделирования и обеспечивает процесс выполнения моделей, описывающих процессы. WebDPF имеет масштабируемый интерфейс и средства навигации, позволяющие проектировать, просматривать и делать запросы к сложным моделям. WebDPF выполняет анализ завершаемости моделируемых процессов на основе принципов, адаптированных из грамматик многоуровневых графов [26], правила преобразования применяются по слоям, в процессе обработки отдельных уровней происходит удаление циклов.

1.12. Melanie

Melanie (Multi–Level Modeling and Ontology Engineering Environment) [24] — платформа для многоуровневого предметно–ориентированного моделирования. Среда поддерживает проверку правильности онтологии на всех уровнях проектируемого языка и исправление модели после каждого изменения на всех уровнях, что позволяет не допускать несоответствий в процессе разработки. В отличие от существующих двухуровневых сред разработки, не позволяющих изменять объект, реализующий метамодель, Melanie позволяет изменять сущности моделей на всех описываемых уровнях без изменения метамодели. Графический интерфейс отображаемого предметно–ориентированного языка использует библиотеки SWT[4] и JMF (Java Media Framework) для отображения различных частей одной модели, также поддерживается текстовый ввод языка. К недостаткам системы можно отнести асимптотическую сложность алгоритма валидации онтологии *O(n8)*, значительно увеличивающуюся при большой связности объектов модели [26]. Также среда исполнения приложения ограничивается применением интегрированной среды разработки Eclipse.

[4] Библиотека SWT (**Standard Widget Toolkit**). SWT библиотека представляет собой кросс-платформенную оболочку для графических библиотек конкретных операционных систем

```
peg::parser!{
  grammar list_parser() for str {
    rule number() -> u32
      = n:$(['0'..='9']+) {? n.parse().or(Err("u32")) }

    pub rule list() -> Vec<u32>
      = "[" l:number() ** "," "]" { l }
  }
}

pub fn main() {
    assert_eq!(list_parser::list("[1,1,2,3,5,8]"), Ok(vec![1, 1, 2, 3, 5, 8]));
}
```

Рис. 5. Построение PEG–парсера на языке Rust

```
rule ::= part1 | part2 | part3

public boolean rule() {
  <header>
  boolean result = false;
  result = part1();
  if (!result) result = part2();
  if (!result) result = part3();
  if (!result) <rollback any state changes>
  <footer>
  return result;
}
```

Рис. 6. Пример получаемого кода в среде Grammar–Kit

В системах «Умный дом», основанных на платформе Node–RED Smart Home Control [16], для генерации кода управления датчиками используется визуальный граф управления устройствами и программирования сценариев (рисунок 7). Процесс обработки данных в такой технологии управляется синтаксической структурой этих данных, которые описываются посредством графов, представляющих правила грамматик. Чаще всего для этой цели используются контекстно–свободные и автоматные грамматики.

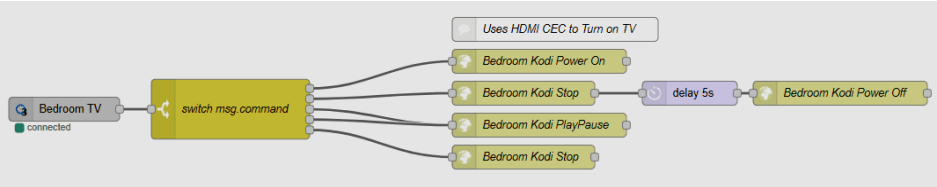

Рис.7. Пример графа управления умным домом в среде Node–RED Smart Home Control

Основные технологии, применяемые для разработки инструментария в современных языках программирования, показаны в таблице 1.

Таблица 1
Технологии работы с грамматиками

Название технологии	**Классы языков**	**Особенности**
ANTLR	*LL*(*)	Генерация деревьев разбора грамматики; применение в Twitter для обработки запросов, парсинг C++ в NetBeans IDE
CPython	*LL*(1), PEG	Ускорение в 1.58 раз с использованием PEG анализатора по сравнению с *LL*(1)
Bison	*LR*(k), Generalized *LR*, *LALR*	Применение для создания синтаксических анализаторов семейства GNU
Rust–peg	PEG	Строит рекурсивный парсер по определению грамматики
LLLPG	*LL*(k)	Приоритет на читаемость кода человеком, разбор и генерация анализатора во время компиляции
JetBrains Grammar–Kit	PEG	Расширение для интегрированной среды разработки IntelliJ IDEA
QReal, REAL.NET	Визуальные языки	MetaCase среда, применение для сред QReal:Robots и TRIK Studio
WebDTF	Визуальные языки	Авто дополнение частично реализованных моделей
Melanie	Визуальные языки	Многоуровневое предметно–ориентирован–ное моделирование

Таблица 2
Сравнение характеристик технологий

№	**Характеристики** (опции и настройки**)**	**JetBrains Grammar–Kit**	**LexYacc**	**Flex/Bison**	**ANTLR**
1	Объем (Кбайт)	813	30924	1024	2052
2	Время (чел/дни)	3660	16790	13140	11680

Таблица 3
Сравнение характеристик технологий

№	**Характеристики** (опции и настройки**)**	**SynGT**	**CPython PEG**	**Rust PEG**	**LLLPG**
1	Объем (Кбайт)	693	26975	3188	4300
2	Время (чел/дни)	8395	365	1825	2117

Таблица 4
Сравнение характеристик технологий

№	**Характеристики** (опции и настройки)	**Trik Studio**	**QReal: Robots**	**WebDPF**	**Melanie**
1	Объем (Кбайт)	48845	34202	24678	12854
2	Время (чел/дни)	3674	1006	10900	583

Сравнение характеристик технологий показано в таблицах 2, 3, 4. Все указанные среды распространяются бесплатно, но значительно отличаются по объему.

При построении приложений, например, транслятора, решая задачи упрощения грамматик и разрешения конфликтных ситуаций, связанные с языковой неоднозначностью и недетерминированностью магазинного автомата (преобразователя), удобно воспользоваться инструментальной системой, разрабатываемой в СПб ФИЦ РАН, основанной на регулярной модели спецификации правил грамматик с регулярными выражениями и атрибутами в виде семантик и предикатов, которые также могут быть проанализированы инструментальной системой. При реализации такой системы с атрибутами в качестве основы взят код инструментального комплекса SynGT (SynGT Graph Transformation).

2 Инструментальная система SynGT

Система SynGT, предназначенная для решения указанных задач при работе с грамматиками, реализована в среде программирования Delphi и имеет оконный пользовательский интерфейс.

Графический редактор осуществляет взаимодействие пользователей в системе с грамматикой и функциями по ее обработки, регулярными выражениями, построенными в соответствии с синтаксисом выражений.

Основная функциональность подтверждается алгоритмом регуляризации. Реализованные в системе модули подробно описаны в работах [2–4].

Алгоритм регуляризации предполагает выполнение следующих базовых эквивалентных преобразований над регулярными выражениями и правилами (продукциями) КС–грамматики в регулярной форме:

- подстановка вместо нетерминала его порождения;
- удаление лево–(право)рекурсивного нетерминалов в правой части правила с помощью прямого преобразования [2];
- объединение общих префиксов в графе для нетерминала;
- удаление повторяющихся альтернатив для нетерминала;
- удаление рекурсий для самовложенных нетерминалов;
- сведение регулярного подвыражения в новый нетерминал в графе для, и создание, нового правила в грамматике;
- удаление лишних правил (приведение грамматики).

Набор базовых функций, реализованных в SynGT, служит основной цели: автоматизированное преобразование КС–грамматики с целью ее максимальной регуляризации, то есть превращение синтаксической граф–схемы грамматики в минимальный набор конечных автоматов для построения анализатора языка. Если КС–грамматика порождает регулярный язык, то синтаксическая граф–схема вырождается в один граф, представляющий эквивалентное регулярное выражение. Алгоритм и примеры даны в работах [2–4].

2.1 Основные модули:

- *текстовый редактор (Класс Creator)* проверяет корректность регулярных выражений (конечно–автоматная модель); формирует приведенную (well–formed) грамматику (удаление лишних правил (с пустым порождением, цепные правила и т.д.), редактирование словарей символов);
- *графический редактор* предоставляет пользовательский интерфейс для *визуализации* грамматики (Классы DrawPoint, Arrow, DrawObject, Analyzer); предоставляет интерфейс для обработки грамматики (Классы Main, Clild, методы обработки пользовательских событий);
- *эквивалентные синтаксические преобразования* формируют *грамматику* для синтеза детерминированного процессора (Класс Parser, функции; класс RegularExpression; класс CharProducer); алгоритм исключения лево и праворекурсивных нетерминальных символов (Класс TransGrammar, функции leftEl, rightEl); удаление непродуктивных нетерминальных символов (Класс TransGrammar, функции leftEl, rightEl); удаление цепочек символов, порождающих циклы (Классы TransGrammar, Minimization); парсер, управляющий процессом построения регулярного выражения на основе грамматики разбора (Классы TransGrammar, TransParser, TransRE_Tree).
- *генератор тестов (*по *определенному критерию).*

2.2 SynGT: задачи, решаемые в системе

Инструмент автоматизирует решение следующих задач:

1. Предварительная обработка текстового представления исходной грамматики. В результате должны иметь: правильные записи выражений скобочной структуры, контроль всех операций в регулярных выражениях, наличие конечного символа; все входящие нетерминалы являются продуктивными.

2. Преобразование текстового представления регулярного выражения в графическое, построение синтаксической граф–схемы.

3. Собственно редактор граф–схемы (стандартные редакторские функции — масштабирование подсветка вершин и прочее).

4. Обратное преобразование граф–схемы в эквивалентное текстовое регулярное выражение на линейке внизу окна редактора.

5. Набор эквивалентных преобразований грамматики и синтаксической граф–схемы (пополняется студенческими работами).

6. Регуляризация КС–грамматики (общая схема).

7. Синтез анализатора (parser).

Заключение

Инструментальные средства активно применяются в различных промышленных комплексах и научных исследованиях, не исчезает потребность в создании новых программ инструментов для работы с моделями, грамматиками и предметно–ориентированными языками. Разрабатываемые инструментальные системы, применяемые в промышленности, распространяются, в основном, по свободным лицензиям, для бесплатного их использования среди разработчиков, вовлечения энтузиастов и популяризации языка и системы.

С развитием робототехнических систем и инструментальных систем остаются неизменными основные требования для систем генерации кода и используемых программных средств, а именно:

- визуализация грамматик, моделей и языков;
- оптимизация получаемого кода по скорости, объему используемой памяти или читаемости генерируемого кода.

Однако, изменение используемых языков программирования и их особенностей стимулирует разработчиков создавать новые инструментальные средства для синтаксически ориентированной обработки данных.

Литература

1. Handbook of Formal Languages, Vol. 2. In: Rozenberg, G., Salomaa, A. (eds.). Springer–Verlag, Berlin, New York, 1997. 527 p.
2. Федорченко Л. Н. Регуляризация контекстно–свободных грамматик. LAP LAMBERT Academic Publishing GmbH & Co. KG Dudweiler Landstr. 99, 66123 Saarbrucken, Germany, 2011. 180 с. ISBN: 978–3–8443–5360–0
3. Novikov F., Fedorchenko L., Vorobiev V., Fatkieva R., Levonevskiy D. Attribute-based Approach of Defining the Secure Behavior of Automata Objects // Proceedings of the 10th International Conference on Security of Information and Networks (SIN'17). ACM, NY, USA, 2017. P. 67–72. https://doi.org/10.1145/3136825.3136887
4. Fedorchenko L., Baranov S. Equivalent Transformations and Regularization in Context-Free Grammars // Bulgarian Academy of Sciences. Cybernetics and Information Technologies (CIT). Sofia, 2015. №4(14). P. 11–28.
5. Chien–Yu Lu, Fei–Hsu Chen, Wen–Chiung Hsu, Yu–Qiang Yang, Te–Jen Su. Constructing Home Monitoring System with Node-RED // Sensors and Materials, 2020. Vol. 32. P. 1701–1710.
6. Terence Parr, Kathleen Fisher. LL(*): The Foundation of the ANTLR Parser Generator // Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI). 2011. P. 425–436.
7. Guido van Rossum, Pablo Galindo, Lysandros Nikolaou. New PEG parser for CPython. https://www.python.org/dev/peps/pep–0617
8. Kevin Mehall. Parsing Expression Grammars in Rust. https://github.com/kevinmehall/rust–peg

9. Koster Affix Grammars // Algol 68 Implementation: Proceedings of the IFIP Working Conference on ALGOL 68 Implementation, Munich, July 20–24, 1970 (ed. by J. E. L. Peck) / Library of Congress Catalog Card Number: 79–146198. P. 95–111. ISBN: 0720420458
10. Sam Harwell, Kathleen Fisher. Adaptive LL(*) Parsing: The Power of Dynamic Analysis // OOPSLA 2014. P. 579–598. https://doi.org/10.1145/2660193.2660202
11. John Levine. Flex & Bison. O'Reilly Media, Inc.,2009. 292 p.
12. David Piepgrass. LLLPG. https://ecsharp.net/lllpg
13. Robert W. Gray, Steven P. Levi, Vincent P. Heuring, Anthony M. Sloane, William M. Waite. Eli: a Complete, Flexible Compiler Construction System // Commun. ACM. 1992. № 2(35). P. 121–130.
14. Баранов С. Н., Ноздрунов Н. Р. Язык Форт и его реализации. М.: Машиностроение, 1988. 157 с.
15. Greg Shrago. Grammar–Kit. https://github.com/JetBrains/Grammar–Kit
16. Kumar Selvaperumal, Waleed Al–Gumaei, Raed Abdulla, Vinesh Thiruchelvam. Integrated Wireless Monitoring System Using LoRa and Node-Red for University Building // Journal of Computational and Theoretical Nanoscience. 2019. №8(16). P. 3384–3394.
17. Cornelis H. A. Koster et al. CDL3 manual. 2004. https://www.cs.ru.nl/cdl3/cdl3.pdf
18. Архитектура среды визуального моделирования QReal / А. Н. Терехов, Т. А. Брыксин, В. А. Литвинов и др. // Системное программирование. 2009. № 4. С. 171–196.
19. Литвинов Ю. В., Кузьмина Е. В., Небогатиков И. Ю., Алымова Д. А. Среда предметно–ориентированного визуального моделирования REAL.NET // Материалы 7–й всероссийской научной конференции по проблемам информатики СПИСОК–2017. Санкт-Петербург: ВВМ, 2017. С. 80–89. http://spisok.math.spbu.ru/2017/txt/SPISOK-2017.pdf
20. Терехов А. Н., Брыксин Т. А., Литвинов Ю. В. Среда визуального программирования роботов QReal:Robots // III Всероссийская конференция «Современное технологическое обучение: от компьютера к роботу» (сборник тезисов). Санкт-Петербург, 2013. С. 1–4.
21. Mordvinov D., Litvinov Yu., Bryksin T. TRIK Studio: Technical Introduction // Proceedings of the 20th Conference of Open Innovations Association (FRUCT 2017). P. 296–308. ISSN: 2305–7254
22. Atkinson C., Kühnen T. In Defence of Deep Modelling // Information and Software Technology. 2015. Vol. 64. P. 36–51.
23. Fazle Rabbi, Yngve Lamo, Ingrid Yu, Lars Kristensen. Diagrammatic Development of Domain-Specific Modelling Languages with WebDPF // International Journal of Vehicle Systems Modelling and Design. 2016. Vol. 7. P. 93–114.
24. Colin Atkinson, Ralph Gerbig. Melanie: Multi-level Modeling and Ontology Engineering Environment // Proceedings of the 2nd International Master Class on Model-Driven Engineering: Modeling Wizards, 27–29 September 2012. P. 1–2. https://doi.org/10.1145/2448076.2448083
25. Bastian Kennel. A Unified Framework for Multi-level Modeling. 2012. https://d-nb.info/1034315374/34
26. Rabbi F., Lamo Y., Yu I. C., Kristensen L. M. WebDPF: A Web-based Metamodelling and Model Transformation Environment // 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2016). P. 87–98.

Статья поступила в редакцию 06.12.2021; одобрена после рецензирования 09.12.2021; принята к публикации 14.12.2021.

TOOLS IN LANGUAGE IT TECHNOLOGIES

Ludmila N. Fedorchenko
Ph.D., Senior Researcher
St. Petersburg Federal Research Center of the Russian Academy of Sciences
(SPC RAS)
39 14th Line of V.O., St. Petersburg 199178, Russia
lnf@iias.spb.su

Abstract. The article presents a list of tools, developed or in the process of development, used in programming systems in laboratories and departments of St. Petersburg State University. Both modern and the most frequently used translator–building tools are considered. 30 years have passed since the situation was covered. Nevertheless, the reader will surely find this navigation scheme useful, which gives a key to studying the activities of various groups and teams in the field of programming both in Russia and abroad.

Keywords: systems for translator developing, domain-specific languages, context-free grammars in regular form.

For citation
Fedorchenko L. N. Tools in Language IT Technologies // Bulletin of Buryat State University. Mathematics, Informatics. 2021. N. 4. Pp. 3–18.

The article was submitted 06.12.2021; approved after reviewing 09.12.2021; accepted for publication 14.12.2021.