ИПМ им.М.В.Келдыша РАН

Абрау-2017 • Труды конференции

Л.В. Городняя

**Резервы синтаксически
ориентированного конструирования
систем программирования**

Рекомендуемая форма библиографической ссылки

Городняя Л.В. Резервы синтаксически ориентированного конструирования систем программирования // Научный сервис в сети Интернет: труды XIX Всероссийской научной конференции (18-23 сентября 2017 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2017. — С. 120-129. — URL: <http://keldysh.ru/abrau/2017/32.pdf> doi:[10.20948/abrau-2017-32](https://doi.org/10.20948/abrau-2017-32)

Размещена также [презентация к докладу](#)

Резервы синтаксически ориентированного конструирования систем программирования

Л.В. Городняя

Институт систем информатики СО РАН, НГУ

Аннотация. Доклад посвящен анализу возможностей использования современного потенциала ИТ при решении задач обработки больших и сложных данных на примере текстов программ и определений языков программирования. Рассматривается проблема совершенствования современных систем программирования и создания новых языков программирования, нацеленных на эффективное решение задач разработки надёжных и удобных информационных систем.

Ключевые слова: системы программирования, декомпозиция программ, реализационная прагматика, определение языков программирования

Введение

Основные подходы к реализации систем программирования (СП) опираются на инструментальную поддержку конструирования лексических и синтаксических анализаторов текста программы, дополненных средствами создания перевода программы на промежуточный язык (фронт-энд) и генератора исполнимого кода программы (бек-энд) [2,3]. Статистика использования таких средств при реализации новых СП показывает, что массово применяются конструкторы анализаторов, достаточно популярны средства создания фронт-энда (Clang-API, Lex, YACC и т.п.) и существенно реже используются средства создания бек-энда (LLVM) [24]. Таким образом, доминирует горизонтальное слоение СП, разработка которых сводится к не слишком трудоёмкой реализации лексики, синтаксиса, семантики отдельно от наследуемой прагматики, дополненной комплексом библиотечных модулей [1,4,9,10,12].

Определённые трудности связаны со сложностью учёта динамики развития массово доступных информационных технологий и их приложений в разных областях, включая параллельные вычисления, распределённую обработку данных, популярный интернет, формальные методы обеспечения надёжности и безопасности программ. Практичность применения инструментов конструирования СП зависит от непрерывной эволюции определения языков программирования (ЯП) и требований к эффективности и надёжности его СП, неизбежно происходящей при сроках разработки, превышающих полгода [5]. Кроме того, долгоживущие и новые ЯП характеризуются тенденцией к

мультипарадигмальности, что означает требование поддержки вариантов реализации, используемых в зависимости от условий применения программ. Реализация таких особенностей обычно сказывается на всех уровнях горизонтального слоения СП, значительный объем которых требует дополнительного структурирования. Такое структурирование даёт методика вертикального слоения программ, в свое время предложенная А.Л.Фуксманом [14], концептуально близкая современному аспектно-ориентированному программированию, что применительно к СП может быть названо функционально-парадигмальной декомпозицией. Такую декомпозицию можно рассматривать как метод выбора критериев для согласованного выделения типовых компонентов в описании ЯП и определении СП, включая спецификацию эксплуатационных аспектов и реализационной прагматики, обеспечивающих варьирование используемых парадигм в зависимости от схемы жизненного цикла решаемых задач и требований к эффективности и производительности программируемых решений, что приводит к проблеме выделения автономно развиваемых подсистем.

Синтаксис, семантика и прагматика

Проблема определения ЯП и СП была тщательно проработана в Венской методике определения языков программирования [10,17]. Основная идея — использование абстрактного синтаксиса (АС) и абстрактной машины (АМ) при определении семантики языка программирования и отделения её от реализационной прагматики на конкретной машине (КМ). Трудоемкость определения АС над АМ строго меньше трудоемкости определения КС над КМ. Конкретный синтаксис (КС) языка отображается в АС, АМ может быть реализована с помощью КМ, причем и отображение, и реализация могут иметь небольшой объем и невысокую сложность.

Диаграмма	Пояснение
$КС \leftrightarrow АС$ $\downarrow$ $АМ \rightarrow КМ$	<p>Существует отображение конкретного синтаксиса (КС) в абстрактный (АС) и обратно.</p> <p>АС отображается в абстрактную машину (АМ).</p> <p>АМ реализуется с помощью конкретной машины (КМ) на базе языка Си.</p>

Схема 1. Абстрактная декомпозиция определения ЯП по Венской методике.

Современный инструментарий реализации ЯП в рамках проекта LLVM-Clang [24], включая новые проблемно-ориентированные языки (DSL), поддерживает подобную схему.

Диаграмма	Пояснение
$DSL \rightarrow AST$ $\downarrow$ $IR \rightarrow MSP$	<p>Существует отображение конкретного синтаксиса (DSL) в дерево абстрактного синтаксиса (AST).</p> <p>AST отображается на язык виртуальной машины (IR).</p> <p>IR реализуется с помощью конкретной платформы (MSP).</p>

Схема 2. Абстрактная декомпозиция определения DSL для реализации с помощью LLVM-Clang.

Интересно, что хотя демонстрация Веской методики была выполнена на материале языка PL/1, неявную реализацию она получила в истории создания языка Lisp [13,18,21]. Многие замечают явное сходство графового представления примеров AST, приведённых в руководстве по применению Clang [24] со списочным представлением Lisp-выражений, что позволяет создавать прототипы новых инструментов на базе Lisp-систем.

C	AST → Lisp
// Example.c #include <stdio.h>	; Комментарии не имеют представления ; Макросы не имеют представления
int global;	(VarDecl global 'int)
int main (int argc, char *argv[]) {	(FunctionDecl main (int, char *) (ParmVarDecl argc 'int) (ParmVarDecl argv 'char** 'char**) (CompoundStat
int param = 1;	(DeclStat (VarDecl param 'int) (IntegerLiteral 1 'int))
myPrint (param);	(CallExpr 'void (ImplicitCastExpr 'void '(*) '()) (DeclRefExpr 'param 'int))
return 0;	(ReturnStmt (IntegerLiteral 0 'int))
}	)

Схема 3. Lisp-аналог Clang-представления примера программы на Си.

Для перехода от определения ЯП к реализации СП ключевое значение имеет семантика [9,10], но для эффективного программирования на любом ЯП требуется понимание не только условий эксплуатации программ, но и реализационной прагматики (РП), которая может быть не представлена в определении или стандарте ЯП, но подразумеваться традиционно или воспроизводится по прототипу. Выбор РП существенно влияет на пространство процессов выполнения программ, неустранимо зависящего от аппаратуры. Реализация LLVM опирается на прагматику языка Си, обеспечивающую доступ к большому числу архитектур [24].

Реализационная прагматика, затрагивая все уровни определения ЯП, в основном предоставляет решения в области конкретной организации вычислений, уточняющей решения и принципы, провозглашенные в определении АМ. В первую очередь это относится к вопросам защиты областей памяти и их конечности, т.е. реагирования на дефицит памяти и учёт разницы в скоростях доступа. Следует сразу особо отметить, что основные традиционные ЯП апеллируют к операционной памяти, время жизни состояний в которой ограничено пределами времени исполнения конкретной программы. Обработка внешней и многоуровневой памяти, а также взаимодействия с независимо создаваемыми программами, возможно подверженными изменениям при исполнении, представляется как побочные эффекты. Это создает проблемы

перехода к большеобъёмным и распределённым данным, а также к синхронизации процессов.

Для основных ЯП определение АМ сводится к системе команд (СК) над небольшим числом регистров, обычно от двух до шести. СК может быть получена как отображение части базовых средств ЯП, а особенности их функционирования отражают реализационную прагматику обработки данных.

Новые горизонты компьютерных языков

Экстенсивное порождение предметно-ориентированных языков в XXI веке знаменует переход к методике конструирования компьютерных языков в качестве средства, позволяющего пользователю, умеющему работать с текстами, таблицами и визуальными оболочками на уровне Word-Excel и Visual-Studio, формировать грамматики, покрывающие пространство понятных решений текущих задач [15,19,20,22]. Происходящее в этом процессе уточнение такого пространства приводит к проблеме минимизации объёма изменений, возникающих в ранее сложившихся парадигмах программирования, нацеленных на борьбу за эффективность однопроцессорных программ над оперативной памятью, рассчитанных на надёжность хранения данных на носителях, допускающих защиту и контроль [23].

Теперь пафос системного программирования переходит к обработке большеобъёмных слабо структурированных данных, не вполне защищённых от искажений, допускающих распределённую и многопроцессорную обработку. Организация параллельных вычислений, конструирование проблемно-ориентированных языков (DSL), обработка big data, обеспечение надёжности Web-сервисов, распределённых баз данных и многое другое [51,16] требуют исследования и формализации специальных методов и парадигм компьютерных языков, в которых практически выделение типовых компонент, встраиваемых в различные информационные комплексы. Предстоит исследование новых эксплуатационных аспектов обработки данных и определение абстрактных машин, полезных при конструировании реализационной прагматики компьютерных языков.

Системное программирование долгое время развивалось под прессом сервисных и заказных работ, нацеленных на создание инструментальных систем массового применения. Свойственный таким работам производственный подход опирается на предпочтение воспроизводимых процессов и стабильных программ, разрабатываемых для многократного использования. Ведущие критерии качества связаны с измеримыми характеристиками эффективности, такими как скорость выполнения готовых программ и объем используемой памяти. Для таких программ оправдана компиляционная схема обработки, статический анализ свойств, автоматизированная оптимизация и контроль. В этой области доминирует императивно-процедурный стиль программирования. Он допускает некоторую стандартизацию и модульное программирование, но обрастает довольно сложными построениями, спецификациями, методами тестирования, средствами сборки программ и т.п. Ещё в конце 1970-х

В.Е. Котов отмечал, что системному программированию нужна методика абстрагирования, сравнимая с дифференциальным и интегральным исчислением в математике.

Основные методы компиляции и традиции разработки систем программирования сложились в условиях низкой производительности оборудования, принудившей к предпочтению алгоритмов, ориентированных на минимизацию объёма хранимых данных, не влияющих на результат исполнения программы даже если они полезны программисту при отладке [7,8]. Жесткость требований к эффективности и надежности отчасти удовлетворяется разработкой профессионального инструментария, использующего сложные ассоциативно семантические эвристики наряду с методами синтаксически-управляемого конструирования и генерации программ. Бесспорный потенциал такого инструментария на практике ограничен трудоемкостью освоения - возникает квалификационный ценз, провоцирующий чрезмерное увлечение автоматичностью, что воспрепятствовало формированию средств и методов сознательного диалога «человек-машина» по поводу процесса разработки и отладки программ и программных систем. Границы между системами разработки программ, системами программирования, средствами препроцессора, языками программирования, стандартами на языки программирования, традиционными методами их реализации, средствами сборки, отладки и тестирования программ визуально слабо подкреплены, нередко воспринимаются как однородная рабочая смесь воздействий на компокуемый по шаблонам текст, начинающий работать по волшебству.

Прогресс в области ИТ привёл к резкому увеличению объёмов доступной памяти, скоростей обработки данных, расширению возможностей применения параллельных вычислений и распределённых информационных сервисов, опережающим возможности человека освоить новые горизонты. Изменился и стартовый уровень интуитивного понимания особенностей обработки данных в сетях, проблем с появлением новых версий программного продукта, обновлением текущих версий, восстановлением данных при прерывании процесса, дистанционной поддержке пользователей, пользовательских интерфейсов и т. п. Всё это даёт основания не только для наращивания мощной, нередко бездумной, автоматизации обработки данных, но и для полного пересмотра походов к организации данных, переходу к интеллектуальному стилю обработки текстов программ в системах разработки программ и системах программирования, реабилитации дискредитированных преждевременной реализацией программистских идей, следуя принципам:

- 1) Всегда доступно всё, что было представлено в программе — комментарии, макросы и макровыводы, идентификаторы, описания и структуры данных.
- 2) Система сохраняет все результаты анализа и любых эмпирических решений.
- 3) Существует общий центр хранения библиотек, тестов, результатов прогона, документации и т. п. - всего, что может пригодиться в других программах.

- 4) Система предназначена для серии сеансов отладки, включая сопоставление результатов разных сеансов.
- 5) Поддерживается развитие постановки задачи (надзадачи, подзадачи).
- 6) При формализации постановок задач, освобождающей от конкретных легенд, возможен диалог в исходных терминах.
- 7) Преобразования данных обратимы — всегда можно восстановить исходный вид.

Такие принципы допускают поддержку работы программиста в удобных ему терминах, без принудительного освоения понятий разработчика СП. Полезны множественная мнемоника, синонимы и сокращения, дающие лаконизм. Желательно использовать любые свойства имён и констант, хотя бы в виде сохранённых комментариев. Билингвистический интерфейс позволит видеть опорную иноязычную терминологию.

Опыт реализации языков функционального программирования и современная тенденция к мультипарадигмальности новых языков программирования дали достаточные основания для перехода к СП, одновременно содержащим интерпретатор, компилятор, мемоизатор и декомпозитор программ. Можно вспомнить, что проекты первых компьютеров допускали, что ряд математических функций будут реализованы в форме таблиц. В экспериментах .Net и GNU по разработке многоязыковых систем программирования заметна тенденция интеграции программ из библиотечных модулей, созданных на разных языках, что можно рассматривать как движение к интегральному исчислению программ, но без опоры на соответствующее дифференциальное, зачатки которого можно видеть в функциональном стиле программирования, аспектно-ориентированном программировании, исследованиях по срезам программ и технике формирования проекций в смешанных вычислениях. Доступны средства вывода логики программы и её верификации на соответствие целям и требованиям. Комплекты поставки программных инструментов нередко включают в себя удостоверенные тесты и документацию, в пределах которых гарантируется работоспособность программы. В этом плане популярные методы работы с определениями языков программирования, рассматриваемых как специализированные программы, могут существенно расширить пространство средств информационной обработки.

Механизмы приведения грамматики языка к виду, удобному для автоматизации конструирования обработчиков программ, включая проверку текстов на возможность их включения в программы при макроподстановках и работе с фрагментными переменными, и грамматики с переводом позволяют принципиально расширить спектр базовых языков и средств обработки программ с накоплением их правильности при отладке. Возможен обратный вывод тестового покрытия программы для сравнения с предполагаемым тестовым набором. Декомпозиция программы по тестовым наборам может дать специализированные проекции, снижающие накладные расходы. Конверторы,

освобождающие от лишних переходов на уровень машины, могут способствовать выделению эквивалентных подязыков и, следовательно, типизации средств их реализации. Раскрытие недоопределённостей в программе при отладке можно выполнять по спецификациям, тестам или запросам. Приведение любых выражений к функциональной форме уже активно используется в новых инструментах создания СП. Сдерживающим фактором является риск ошибки в компромиссе между автоматизацией и ручным управлением.

Кроме того, целесообразно учесть возможность разумной обработки программ и настройки программных систем квалифицированным программистом, подобно тому как создатели операционных систем допускают существование **сисадмина**, способного понимать устройство операционки и её окружения, а потому имеющего права выполнять настройки на архитектурные особенности и другие воздействия на функционирование системы. На этом пути достаточно естественно выглядит допущение авторских преобразований программ с проверкой корректности их применения, сравнение разных схем обработки программ и их оптимизация, использование параллелизма обработки данных по частям, сбор статистики и построения гистограмм функционирования на разных тестовых наборах.

Системный анализ в программировании

Любые массово распространённые технологии апеллируют к понятию «система» и программирование в этом не исключение. Это понятие в конце XIX века привлекло внимание философов, сформулировавших большое число его свойств, за каждым из которых стоит великое имя, но редко встречается чёткая формулировка для определения границ понятия или наличия свойства в реальных комплексах. Существует общая теория систем без представления механизмов создания и функционирования систем.

Понятие «системный анализ» активно используется в менеджменте при организации производственных процессов на уровне интуитивного понимания. На этом фоне понятие “система” приобрело множество толкований, имеющих отношение ко многим производным понятиям в практике создания и применения ИТ. Методы управления программистскими проектами используют системный анализ при подготовке постановки задачи к передаче разработчикам программ. В этом плане программирование обладает своей спецификой:

- Система команд – базовое понятие при определении компьютерных архитектур.
- Операционные системы – неотъемлемая составляющая любой практической компьютерной конфигурации.
- Системный подход – методика учёта особенностей используемой аппаратуры при выборе эффективных решений.
- Система программирования – инструментальный комплекс, поддерживающий работу на языках программирования.
- Системное программирование – деятельность по созданию

инструментальных информационных систем и комплексов.

- Информационные системы – средства обработки данных, рассматриваемых как представление информации о реально существующих объектах.

- Системная информатика – область знаний, исследующая закономерности создания и применения информационных систем и систем программирования.

Независимо от сферы приложения особенно важны жизнеспособные системы, обладающие устойчивостью, безопасностью и надёжностью. Часто говорят о системообразующих факторах, системных исследованиях, методах и процедурах. Ассоциация АПКИТ опубликовал стандарт на профессию «Системный архитектор».

Системные особенности программирования по существу связаны с присущей ему методикой обобщения частных прецедентов обработки конкретных данных к организации многократно воспроизводимых процессов обработки типовых форматов данных. Такое обобщение содержательно напоминает описанные По́я подходы к решению математических задач [11]. Основа так понимаемого системного анализа в программировании сводится к декомпозиции постановок решаемых задач на более общие и более конкретные задачи с определённым прогнозом по развитию обстановки применения решения задачи. При таком рассмотрении и задача и программа её решения являются эволюционирующими объектами, что позволяет моделировать поведение программ как живых систем, используя результаты этологии. Интересно отметить, что К. Лоренц даёт вполне конструктивное определение понятия «система», сводит его к возможности в человеческой речи называть сложное одним словом [25, 26].

Заключение

Сложность современных информационных систем требует специальной декомпозиции программ, допускающей ясное связывание с речевой практикой, что может дать вертикальное слоение по принципу функционально-парадигмальной декомпозиции, позволяющей разделять фрагменты по функциональному назначению и вариантам их реализации. Для разработчика программы мало заметна граница между системой программирования и системой разработки программ.

Разнообразие реалий современных ИТ трудно вписывается в модели обработки данных в защищённой оперативной памяти, что влечёт массовое появление новых проблемно-ориентированных языков программирования, требующее развития методов конструирования систем программирования, рассматриваемых как живые системы.

Прогресс в области эксплуатационных характеристик оборудования даёт основания для пересмотра и развития подходов к организации обработки программ системами программирования. Системность в системном программировании начинается с системного анализа доступных возможностей

и тенденций их развития, дополненного систематическим накоплением типовых методов и отлаженного конструктива. Средства и методы синтаксически ориентированного конструирования СП прошли серьёзные испытания предыдущей эволюцией языков и систем программирования в качестве инструментов снижения трудоёмкости программирования и обеспечения надёжности разрабатываемых программ. Теперь стоит задача исследования их возможностей и резервов в новых эксплуатационных условиях.

Работа выполнена при поддержке Российского фонда фундаментальных исследований, проект 15-07-06345-а.

Литература

1. Ахо А.В., Хопкрофт Дж.Э., Ульман Дж.Д. Структуры данных и алгоритмы. - М.: Вильямс, 2000. – 384 с.
2. Разработка компиляторов: Дмитрий Булычев, Наталья Вояковская, Антон Москаль, Андрей Терехов – URL: <http://www.intuit.ru/studies/courses/26/26/info>
3. Теория и реализация языков программирования. Галочкин м., Гончар Д., Серебряков В., Фурутян М. – URL: <http://www.intuit.ru/studies/courses/1157/173/info>
4. Городняя Л.В. Парадигма программирования: курс лекций // Новосиб. Гос. Ун-т. – Новосибирск : РИЦ НГУ, 2015. – 206 с.
5. Зуев Е. История разработки компилятора Си++ по заказу иностранной фирмы в ранне постсоветское время. – URL: http://www.gramotey.com/?open_file=1269097005
6. Касьянов В.Н., Евстигнеев В.А. Графы в программировании: обработка, визуализация и применение. – С-Пб.: ЕХП-Петербург, 2003, – 1104 с.
7. Крайнева И.А., Марчук А.Г. Игорь Васильевич Поттосин. Из истории новосибирской школы программирования (к 80-летию со дня рождения). Вестник НГУ, Серия: математика, механика, информатика. 2013, №1. С. 2-12.
8. Крайнева И.А., Черемных Н.А. Альфа-язык и транслятор // Открытые системы. – 2014. – №6. – URL: <http://www.novsu.ru/file/867726> – Открытые системы. СУБД 2016 № 01.
9. Лавров С.С. Методы задания семантики языков программирования. // Программирование, N 6, 1978. – С. 3-10.
10. Оллонгрэн А. Определение языков программирования интерпретирующими автоматами. // М.: Мир, 1977. – 288 с.
11. Пойя Д. Как решать задачу. – М.: Либроком, 2010. — 208 с.
12. Уоткинс Д., Хаммонд М., Эйбрамз Б. Программирование на платформе .Net // М. Вильямс, 2003. – С. 367.
13. Хендерсон П. Функциональное программирование. – М.: Мир, 1983. – 349 с.
14. Фуксман А.П. Технические аспекты создания программных систем. – М.: Статистика, 1979. – 180 с.

15. Baar T. Verification Support for a State-Transition-DSL Defined with Xtext. Perspectives of System Informatics – 10th International Andrei Ershov Informatics Conference, PSI 2015, in Memory of Helmut Veith, Kazan and Innopolis, Russia, August 24-27, 2015, Revised Selected Papers. Lecture Notes in Computer Science 9609, Springer 2016, p.50-60.
16. Knoop J. Compiler Construction. – 20th International Conference, CC 2011. Held as Part of the Joint European Conferences on Theory and Practice of Software, Lecture Notes in Computer Sciences, 6601. ETAPS 2011 Saarbrcken, Germany, March 26 – April 3, 2011. Springer. 330 p.
17. Lucas P., Lauer P., Stigleitner H. Method and Notation for the Formal Definition of Programming Languages. IBM Laboratory – Venna, TR 25.087, 1968.
18. McCarthy J. LISP 1.5 Programming Mannual. – The MIT Press., Cambridge, 1963. – 106p.
19. Mernik M. Formal and Practical Aspects of Domain-Specific Languages. – IGI Global, 2012. – [ISBN 978-1-4666-2092-6](#).
20. Voelter M., DSL Engineering: Designing, Implementing and Using Domain-Specific Languages, 2013 – URL: <http://voelter.de/dslbook/markusvoelter-dslengineering-1.0.pdf>, <http://dslbook.org/>
21. Сайт с материалами по особо эффективной реализации Lisp-a – CMUCL. – URL: <http://www.cons.org/cmucl/>
22. Taha W. Domain-Specific Languages. Houston. 2009. – URL: <http://www.effective-modeling.org/p/walid-taha.html>
23. JetBrains, Metaprogramming System MPS – URL: <https://www.jetbrains.com/mps/>
24. http://clang.llvm.org/get_involved.html материалы по Clang – LLVM.
25. <https://www.livelib.ru/book/1000334658-tak-nazyvaemoe-zlo-konrad-lorents> сайт с текстом книги Конрада Лоренца «Так называемое зло» в переводе А.И. Федорова (А.И. Фета).
26. <http://www.rulit.me/books/tak-nazyvaemoe-zlo-read-311025-1.html> сайт с возможностью чтения книги Конрада Лоренца «Так называемое зло» в переводе А.И. Федорова (А.И. Фета).