Actor Prolog to Java translation*

Morozov A. A.
Kotel'nikov IRE RAS, Moscow, Russia

A method of compilation of the Actor Prolog concurrent object-oriented logic language is developed and implemented. The method includes global optimization based on analysis of domains of predicate arguments and domains of attributes of classes, determinism analysis, analysis of constructors of class instances, and flow analysis. Translation to intermediate Java code provides robustness and platform independence of executable code. At the same time, high performance of executable code is achieved (up to 60% in comparison with the exe-code). The Actor Prolog programming system can be extended by independent developers by appending new built-in classes written in Java.

Трансляция Акторного Пролога в Джаву*

Морозов А. А.[1]
morozov@cplire.ru
[1]Москва, ИРЭ им. В. А. Котельникова РАН

Разработан и реализован метод компиляции параллельного объектно-ориентированного логического языка Акторный Пролог с использованием трансляции в промежуточный код на языке Джава. Метод компиляции предусматривает глобальную оптимизацию программ с использованием анализа типов данных аргументов предикатов и атрибутов классов, анализа детерминированности предикатов, анализа вариантов использования конструкторов экземпляров классов, а также потокового анализа. Использование Джавы в качестве промежуточного языка позволяет обеспечить надёжность исполняемого кода, а также независимость от аппаратной платформы и операционной системы. При этом удалось обеспечить высокую скорость исполнения программ, достигающую 60% по сравнению с exe-кодом. Достоинством выбранной схемы компиляции является также возможность расширения системы программирования независимыми разработчиками путём добавления новых предопределённых классов, реализованных на языке Джава.

Качественная компиляция логических языков представляет собой сложную задачу, решение которой требует разработки и реализации нетривиальных методов глобального анализа и оптимизации программ. Сложность этой задачи связана с тем, что компилятор, фактически, должен реализовать переход из логической в процедурную парадигму программирования, и, в частности, каким-то образом реализовать такие высокоуровневые механизмы логического языка как унификация аргументов и откат программы. Ещё более интересную задачу представляет собой компиляция объектно-ориентированных и параллельных логических языков, обеспечивающих более высокий уровень абстракции исходной программы.

Результаты экспериментов с компиляцией параллельного логического объектно-ориентированного языка Акторный Пролог [1–7] показывают, что в настоящее время является актуальным и целесообразным применение сложных схем компиляции, включающих трансляцию в промежуточный код на высокоуровневом платформо-независимом языке (в качестве такого языка использовался язык Джава JDK7). Производительность современных компьютеров оказывается вполне достаточной, чтобы можно было жертвовать скоростью исполнения кода ради достижения надёжности работы программ, независимости от архитектуры машины и операционной системы, а также (и это, по мнению автора, является наиболее важным) возможности быстро развивать систему программирования и поддерживать её в актуальном состоянии при появлении новых архитектур процессоров и новых версий операционных систем.

*Автор благодарит своих родителей и друзей за поддержку при выполнении проекта.

Современное состояние области

Существующие проекты разработки трансляторов логических языков с использованием промежуточного высокоуровневого языка можно классифицировать по следующим признакам:

1. *Исходный логический язык.* Некоторые исследователи разрабатывают методы трансляции для своих собственных логических языков (Mercury, KLIJava, KLIC, λProlog, Janus), другие проекты (BinProlog, P#, PrologCafe, jProlog, wamcc) нацелены на компиляцию стандартного (Клоксин-Меллиш) Пролога. Поддержка в логических языках деклараций типов и направлений передачи аргументов позволяет кардинально увеличить скорость исполняемого кода.
2. *Промежуточный язык.* Как правило, используют языки Java, C, C#. Использование языка C позволяет достичь наилучших показателей по скорости исполняемого кода (Mercury, KLIC, wamcc), использование языка Джава обеспечивает платформо-независимость (PrologCafe, KLIJava, SAE-Prolog, jProlog).

3. *Понятность промежуточного кода.* В некоторых проектах (P#, SAE-Prolog) прямо сформулирована задача получения кода, удобного для чтения человеком. Показательной является история проекта Mercury, постепенно эволюционирующего от задач достижения максимальной скорости кода любыми средствами в сторону задач обеспечения понятности кода и возможности высокоуровневой оптимизации на этапе компиляции промежуточного языка.
4. Использование *абстрактной машины Уоррена* (WAM) или её расширений. Применение WAM является классическим подходом в области компиляции логических языков и оказало существенное влияние на многие проекты (P#, Prolog Cafe, wamcc). Использование WAM упрощает трансляцию, но при этом делает промежуточный код сложным для чтения.
5. *Поддержка неосновных термов* (структур данных, включающих неопределённые переменные). В общем случае, конечно, любой логический язык должен поддерживать как основные, так и неосновные термы, так как это следует из математического определения унификации. Тем не менее, некоторые разработчики отказываются от поддержки неосновных термов (в терминах языка Visual Prolog это означает отказ от поддержки так называемых «ссылочных» типов, в языке Mercury возникновение неосновных термов предотвращается синтаксисом языка, при этом разработчики почему-то не считают, что это обедняет его выразительные возможности). Унификация основных термов может быть реализована очень эффективно (унификация заменяется операциями сравнения, не нужен трэйл). Поддержка неосновных термов существенно усложняет трансляцию языка.

Акторный Пролог существенно отличается от Клоксин-Меллиш Пролога как синтаксически, так и семантически. При этом некоторые средства Акторного Пролога существенно упрощают задачу компиляции (наличие деклараций типов и направлений передачи аргументов, а также детерминизма предикатов), а некоторые существенно усложняют (объектно-ориентированные средства, логические акторы, параллелизм).

В качестве промежуточного языка был выбран язык Джава, чтобы обеспечить платформонезависимость исполняемого кода. При этом была поставлена задача генерации качественного, понятного человеку кода на Джаве. Транслятор Акторного Пролога не использует WAM и переводит конструкции логического языка непосредственно в конструкции Джавы. Система типов Акторного Пролога поддерживает как основные, так и ссылочные типы данных (домены).

Схема трансляции

Компиляция программы на Акторном Прологе включает следующие основные этапы:

1. *Лексический и синтаксический анализ программы.* Реализованы методы «умной трансляции», предотвращающие повторный анализ исходных файлов, не изменившихся после предыдущего запуска программы.
2. *Анализ связей между классами программы.* На этом этапе собирается информация о всех вариантах использования классов программы (в частности, о типах аргументов, передаваемых в конструкторы экземпляров классов).
3. Проверка *типов аргументов* предикатов, а также *типов атрибутов* классов.
4. Проверка *детерминированности* предикатов.
5. *Глобальный потоковый анализ программы.* На этом этапе осуществляется сквозной анализ направлений передачи аргументов предикатов программы.
6. *Генерация исходных файлов на Джаве.*
7. *Трансляция Джава-программы.*

Анализ детерминированности предикатов программы позволяет использовать разные методы трансляции для разных видов предикатов. Различаются три вида предикатов:

1. *Недетерминированные предикаты.* Трансляция недетерминированных предикатов осуществляется на основе механизма передачи континуаторов. Недетерминированные предикаты реализуются с помощью классов Джавы (предложения одного предиката соответствуют одному или нескольким классам Джавы).
2. *Детерминированные предикаты.* Такие предикаты представляются с помощью процедур (все предложения одного предиката соответствуют одному методу Джавы). При этом откат программы реализуется с помощью механизма исключений (используется «облегчённая» разновидность исключений, без трассировки стека).
3. Так называемые *«императивные предикаты»*. Императивными предикатами в Акторном Прологе называются предикаты, исполнение которых всегда заканчивается успехом и не вызывает создание точек отката. Императивные предикаты также представляются с помощью методов Джавы и, фактически, являются обычными подпрограммами процедурного языка. На транслятор Акторного Пролога возлагается обязанность проверки императивности предикатов; для этого осуществляется анализ всех предложений предиката по отдельности, а также их влияния друг на друга.

Для рекурсивных предикатов выполняется оптимизация хвостовой рекурсии (если это возможно). В случае взаимной рекурсии предикатов опти-

мизация не выполняется. В результате этой оптимизации рекурсивные предикаты представляются на уровне Джавы в виде циклов *while*. Заметим, что оптимизация хвостовой рекурсии в логическом языке является принципиально важным элементом компиляции, так как иначе язык будет непригоден для программирования длинных циклических операций (через некоторое время будет возникать переполнение стека).

Сложность выбранной схемы трансляции определяется, в основном, необходимостью сопряжения различных методов представления предикатов в тех случаях, когда недетерминированные предикаты вызываются в теле детерминированных или императивных предикатов (и наоборот). Кроме того, разные способы представления предикатов предполагают использование разных способов реализации отсечений (оператора *cut*).

Отдельной задачей является реализация унификации. Основные (ground) типы аргументов предикатов реализуются более эффективно (с помощью операций сравнения), чем ссылочные (reference). Для работы со структурами данных ссылочных типов используется трэйл. При откате программы трэйл очищается, и отменяются операции связывания и сцепления переменных ссылочных типов.

Качество исполняемого кода

Стандартные тесты скорости исполняемого кода (см. таблицу 1) показывают, что выбранная схема компиляции даёт результаты, вполне пригодные для промышленного использования Акторного Пролога. В случае если используются основные типы данных и детерминированные предикаты, скорость исполняемого кода достигает 60% по сравнению с exe-кодом (тест *Nrev* показывает скорость в десятки мегалипсов на стандартном персональном компьютере). При использовании недетерминированных предикатов (тесты *Crypt*, *Queens*, *Query*) скорость исполняемого кода может упасть в десять раз и более, что связано с объективными трудностями трансляции. В то же время, существенная разница на тестах с целочисленной арифметикой (*QSort*, *Tak*) и символьными вычислениями (*Deriv*, *Poly*) демонстрирует значительные резервы для дальнейшей оптимизации.

Таблица 1. Скорость откомпилированного кода (Pentium Q9450, 2.67 GHz, 3.25 GB).

Test	Java	exe
NREV	34,807,018 lips	54,385,965 lips
CRYPT	8.25 ms	4.12 ms
CRYPT[!]	5.282 ms	4.16 ms
DERIV	0.085265 ms	0.00335 ms
POLY_10	19.25 ms	3.42 ms
PRIMES	0.102350 ms	0.172 ms
QSORT	0.113750 ms	0.0139 ms
QUEENS	35.562 ms	2.42 ms
QUERY	4.5984 ms	0.308 ms
TAK	10.64 ms	1.55 ms

Выводы

Основными результатами работы, по мнению автора, является то, что:

1. Компиляция Акторного Пролога с использованием Джавы в качестве промежуточного языка позволяет получить достаточно качественный исполняемый код, пригодный для практического применения.
2. Разработанная схема компиляции даёт возможность независимым разработчикам развивать систему программирования «Акторный Пролог», создавая свои собственные Джава-классы.
3. Джава в качестве промежуточного языка открывает более широкие возможности для использования Акторного Пролога в таких областях как трёхмерная графика (Java3D), веб-агенты на основе протоколов различного уровня, взаимодействие с базами данных, эксперименты с пользовательским интерфейсом.

Литература

[1] *Морозов А. А.* Акторный Пролог // Программирование. — 1994. — № 5. — С. 66–78.

[2] *Morozov A. A.* Actor Prolog: an Object-Oriented Language with the Classical Declarative Semantics // IDL'99. — Paris, France, 1999. `http://www.cplire.ru/Lab144/paris.pdf`

[3] *Morozov A. A., Obukhov Yu V.* Logic Object-Oriented Model of Asynchronous Concurrent Computations // Pattern Recognition and Image Analysis. — 2003. — Vol. 13, No. 4. — P. 640–649.

[4] *Morozov A. A.* Development and Application of Logical Actors Mathematical Apparatus for Logic Programming of Web Agents // ICLP'03. — Springer-Verlag, 2003. — LNCS 2916. — P. 494–495.

[5] *Морозов А. А.* Об одном подходе к логическому программированию интеллектуальных агентов для поиска и распознавания информации в Интернет // Журнал радиоэлектроники. — 2003. — Ноябрь. `http://jre.cplire.ru/jre/nov03/1/text.html`

[6] *Morozov A. A.* Operational Approach to the Modified Reasoning, Based on the Concept of Repeated Proving and Logical Actors // CICLOPS'07. — Porto, Portugal, 2007. — P. 1–15. `http://www.cplire.ru/Lab144/ciclops07.pdf`

[7] *Morozov A. A.* Visual Logic Programming Method Based on Structural Analysis and Design Technique // ICLP'07. — Springer-Verlag, 2007. — LNCS 4670. — P. 436–437.