рому она принадлежит. Наконец, роль индексальных знаков играет программное окружение данного объекта, т. е. процедуры, функции и переменные, не включённые в объекты в качестве методов и полей. В свете всего сказанного объект следует рассматривать, как элемент разбиения семиотического пространства. Таким образом, объектно-ориентированное программирование может представлять собой предмет исследования в рамках количественной семиотики.

Интересно отметить также и следующее. Всякое уточнение описания объекта, или, что то же самое, всякое поступление дополнительной информации о денотате приводит к измельчению соответствующего разбиения. Подобный факт позволяет построить математическую модель многоэтапного процесса семиозиса, в которой каждая новая порция информации будет оформляться как самостоятельное разбиение семиотического пространства, измельчающее ранее построенное разбиение.

Выводы

1. Формализация представлений семиотики, построенной на понятиях треугольника Фреге и процесса семиозиса, может быть осуществлена за счёт представления основных элементов процесса семиозиса в виде множеств и алгебры, определённой на этих множествах.
2. Переход к количественной семиотике осуществляется за счёт определения меры на указанных множествах и следующий шаг на этом пути состоит во введении понятия измельчения соответствующих разбиений.
3. Измельчение разбиений может происходить с применением методик объектно–ориентированного программирования и анализа инкапсуляции. Такое разбиение может рассматриваться как модель программного продукта.

УДК 681.3.068

О. А. Мунтян

МНОГОПЛАТФОРМЕННЫЙ ТРАНСЛЯТОР

В области создания программного обеспечения за многие годы человеческой деятельности накопилось множество идей выраженных в виде алгоритмов и реализованных на различных языках программирования для различных платформ. По прошествию многих лет некоторые языки программирования перестали существовать или были перенесены на другие платформы. Однако накопленный опыт по решению определённых задач не должен быть утерян только из-за того, что тот или иной язык программирования стал недоступен или в настоящее время потерял актуальность.

Найти выход из данной ситуации поможет создание транслирующей подсистемы, способной обеспечить такие возможности работы, как анализ программы с восстановлением её исходного алгоритма, перевод восстановленного алгоритма на язык промежуточного представления (*с последующим сохранением его в*

библиотеке алгоритмов) и трансляция алгоритма, выбранного из библиотеки алгоритмов, на заданную платформу.

В качестве обоснования необходимости создания такой транслирующей подсистемы может быть приведена моделирующая система, использующая множество моделей, которые могут быть написаны на различных входных языках моделирования (*MathLab, Siber, Spice, ACSL и др.*). Однако использование их в оригинальном виде довольно затруднительно из-за различных факторов, среди которых весьма важным, на взгляд автора, является совместимость платформ реализации той или иной программы. Для обеспечения межплатформенной совместимости предлагается разработать и создать транслирующую подсистему, обеспечивающую такую совместимость за счет транслирования программных модулей на язык промежуточного представления.

Основой такой подсистемы должен стать транслятор, обладающий возможностью распознавания и разбора поступающих на его вход исходных текстов программ, написанных на различных языках для различных платформ. Такой транслятор должен иметь весьма развитую систему описания лексических и синтаксических конструкций языков, которые будут "*пониматься*" транслятором. В данном случае сам процесс трансляции несколько отличен от традиционного. Традиционный процесс трансляции предусматривает перевод исходной программы в объектный код с последующим его выполнением, а в рассматриваемом подходе осуществляется только перевод в промежуточную форму (*замена объектного кода*).

Осознавая тот факт, что невозможно охватить полностью все языки моделирования и создать транслятор, понимающий их и обрабатывающий программы написанные на этих языках, поэтому круг языков был ограничен. Было решено использовать язык моделирования ACSL и язык структурного моделирования (ЯСМ), разрабатываемого на кафедре ВТ ТРТУ для использования в работе многопроцессорного моделирующего стенда в качестве базового языка описания моделей.

При разработке транслятора, обеспечивающего межплатформенную трансляцию с языков ACSL и ЯСМ, возникли сложности в плане несовместимости внутренних структур программ, написанных на данных языках. Программа на языке ACSL представляет собой неявно выраженный параллельный алгоритм, т.е. распараллеливание программы производится фирменным транслятором, в то время как ACSL программа представлена линейным алгоритмом, а программа на языке структурного моделирования имеет четко выраженную параллельную структуру.

В связи с этим было решено разработать язык высокого уровня для описания алгоритма программы, который будет использоваться транслятором для представления программы в промежуточной форме. Данный язык должен обладать достаточным набором команд и операторов, чтобы дать возможность представления программ, написанных на разных языках и имеющих разные грамматики.

Одним из направлений развития работ стала необходимость представления поступающих на трансляцию программ с линейным алгоритмом в виде парал-

лельного алгоритма для выполнения на многопроцессорных расширителях. В процессе работы были исследованы методы и алгоритмы распараллеливания программ с целью создания транслятора, генерирующего параллельные команды с исходной программы, представленной в виде линейного алгоритма.

Одним из возможных подходов к выявлению параллелизма является использование векторизующего транслятора, напрямую генерирующего параллельные команды по исходной последовательной программе. Однако такой транслятор будет жестко привязан к архитектуре аппаратных средств, для которых создается параллельная программа, что не позволит реализовать идею многоплатформенной трансляции. Следовательно, более предпочтительным является использование разработчиком специальных параллельных выражений для описания алгоритмов программ. Данный подход также позволяет учитывать потенциальный параллелизм, заложенный в структуру ACSL программ.

На первом шаге преобразования осуществляется описание алгоритма программы при помощи средств формализации. В качестве форм представления алгоритма могут быть использованы такие выражения, как программы с однократным присваиванием, рекурсивные алгоритмы, кадры, графы зависимостей. Выбор формы определяется удобством, четкостью, краткостью и универсальностью перечисленных выражений.

По мнению автора, граф зависимостей более подходит для решения задачи описания зависимостей вычислений, представленных в программе, по данным и по управлению. При решении задач многоязыковой и многоплатформенной трансляции наиболее удобным, универсальным и естественным выражением алгоритмов программ как на уровне их описания, так и в параллельной форме является их задание при помощи графов зависимостей.

Существование множества решений конкретных проблемных задач в виде программ на языках высокого уровня делает целесообразным предварительный анализ и переход от программы к графу. При этом следует привести программу к некоторому стандартному виду. Анализ производится последовательно для каждого оператора, причём особое внимание уделяется выделению циклов внутри анализируемой программы. Необходимость использования программ, написанных на разных языках, повлечёт за собой внесение изменений лишь для модулей синтаксического и лексического анализаторов транслятора и их настройку на соответствующий язык.

Графом алгоритма программы будем называть ориентированный граф G=(V,F) с конечным числом вершин V и дуг F. Каждому оператору исходной программы ставится в соответствие вершина графа V_i, имеющая вес t_i, который определяется временем выполнения соответствующего оператора. Несомненно, точная оценка времени выполнения оператора затруднена, однако при формировании графа важно задание верхних оценок. Вес t_i определяется по составу операций: $t_i = \sum_{j=1}^{n} k_i * t_{oj}$, где k_j – число операций j-го типа в выражении; t_{oj} – время вычисления j-й операции; n - число возможных операций. В число возможных

операций входит и вычисление функций с соответствующим временем. Помимо веса t_i каждая вершина характеризуется набором используемых в операторе переменных. При переходе от программы к графу она трансформируется в два множества – входных переменных $I(V_i)$ и выходных переменных $O(V_i)$ для оператора V_i. Оператор может иметь l входных и m выходных переменных, причем $I(V_i) \cap O(V_i) \notin \varnothing$, т.е. одни и те же переменные могут быть использованы как входные и выходные. Множество входных переменных $I(V_i)$ включает в себя переменные, используемые для вычислений в данном операторе. Множество $O(V_i)$ содержит переменные, которые изменяют свое значение в процессе вычисления оператора.

Множество дуг F определяется существующими между операторами зависимостями, относящимися к одному из следующих типов: зависимость по данным прямая, определяемая как $O(V_i) \cap I(V_j) \notin \varnothing$, где предшествующий V_j оператор; зависимость по выходу при выполнении условия $O(V_i) \cap O(V_j) \notin \varnothing$, где V_i – предшествует V_j; зависимость по данным инверсная, определяемая соотношением $I(V_i) \cap O(V_j) \notin \varnothing$, где V_i предшествует V_j; логическая зависимость, соответствующая связям по управлению последовательностью выполнения операторов определяется операторами переходов и ветвления.

Число дуг, выходящих из вершины V_i, равно числу вершин, использующих результат из V_i, и зависит от внешних переменных. Общее число дуг в данном случае может быть достаточно большим и затруднит описание задачи, следовательно, целесообразно указывать для каждой вершины V_i только входные дуги, общее количество которых определяется числом используемых в операторе входных переменных. Определенная сложность заключается в поиске входной дуги для вершины V_i, при которой необходимо среди множества предыдущих вершин V найти именно ту вершину V_K, последней вычисляющую требуемый аргумент для V_i. Для решения этой задачи, а также последующего анализа и распараллеливания циклических конструкций имеет смысл ввести для каждого оператора программы ряд дополнительных параметров $p_1,...p_i$, указывающих на последовательность выполнения операторов, определяемых ходом вычислений, а также закрепляющих между операторами отношения предшествования.

После обработки всей программы каждый оператор будет иметь три параметра (*для первого уровня один значащий*), по которым можно однозначно определить принадлежность конкретного оператора циклу, поскольку все операторы одного цикла имеют равные p_1, а также порядок выполнения этого оператора в цикле по p_2 и уровень вложенности цикла данного оператора по параметру p_3.

Но наиболее важной и полезной является предоставляемая таким описанием возможность вычисления порядка выполнения конкретных операторов. Помимо указанных параметров каждому оператору, отображаемому в дальнейшем на вершину графа, присваивается номер N_i, причем нумерация осуществляется не в порядке появления операторов во фрагменте программы, а согласно логическим связям, указывающим на ход вычислений. Данный способ описания вершин графа

программы также позволяет сократить общее количество вершин графа за счет исключения операторов цикла, что приводит к удалению дуг, определяющих циклический возврат назад и получение ациклического графа.

Поскольку при формировании графа циклического участка алгоритма следует учитывать зависимости по данным в цикле, то необходима дополнительная процедура анализа зависимости. Так, при выполнении итераций цикла можно выделить следующие виды зависимости: внеитерационную, при которой операторы зависят друг от друга при выполнении одной и той же итерации; итерационную, когда зависимость образуется при вычислении последующих итераций цикла. Для удобства дальнейшего распараллеливания целесообразно ввести числовой параметр, отображающий число витков между итерациями для зависимых операторов – вершин, который назовем расстоянием итерационной зависимости R. При расчёте R потребуется ряд величин, формируемых при переходе от операторов программы к вершинам графа.

Рассмотрим оператор V_i, охватываемый k циклами, каждый из которых имеет переменную цикла $l_m, m = \overline{1,k}$. Тогда оператору V_i можно поставить в соответствие вектор $C_i = (C_{i1}, C_{i2}, ..., C_{ik})$, где $C_{im}, m = \overline{1,k}$ - значения переменных циклов, перечисленные в порядке вложенности этих циклов. Следовательно, зависимость между операторами может быть выражена через сравнение векторов этих операторов, выполненное для входящих в оператор элементов данных. Выбранный для этих целей вектор расстояния имеет размерность k по числу циклов и определяется как: $r_{im} = C_{im} - C_{jm}, m = \overline{1,k}$, где C_{im} - m-я переменная вектора C_i оператора V_i, C_{jm} - m-я переменная вектора C_j оператора V_j, причём V_i предшествует V_j. Данное расстояние определяется для соответствующих элементов данных, и при наличии у двух операторов нескольких одинаковых элементов данных из множества расстояний выбирается минимальное. При определении R_{ij} возможны случаи: $R_{ij} = 0$, т.е. между V_i и V_j существует внеитерационная зависимость; $R_{ij} < 0, R_{ij} > 0$, между V_i и V_j существует итерационная зависимость, причём $|r_{ij}|$ определяет количество витков цикла до связанной отношением зависимости итерации, а знак - направление. Помимо этого, поскольку элементы вектора $R_{ij} = (r_{i1}, r_{i2}, ..., r_{ik})$ упорядочены соответственно вложенности циклов, то можно определить, на котором из уровней вложенности, т.е. для какого цикла, появляется зависимость операторов.

Все рассчитанные на данном этапе параметры приобретают особое значение при решении задачи распараллеливания, а также при формировании дуг графа алгоритма.

Как отмечалось ранее, множество дуг F графа G определяется четырьмя типами зависимостей, как для линейных, так и для циклических участков, но зависимости по данным внутри циклов относятся также к одному из типов итерационных зависимостей. Для учёта данной особенности предлагается каждой входной дуге F_{ij} оператора, принадлежащего циклу, ставить в соответствие вектор расстояния зависимостей R, задающий количественные характеристики итерацион-

ной зависимости и номер вершины V_i, последней вычисляющей необходимое для V_j значение переменной.

Введение вектора расстояний R для дуг графа позволит также легко выяснить тип циклического участка и возможность его распараллеливания. Так, например, при определении типа цикла важно выявить зависимость между итерациями. Если итерации информационно независимые, т.е. цикл относится к типу DOPAR, то вектор R данного цикла должны иметь $r_s = 0$, где s является уровнем вложенности данного цикла. В этом случае цикл поддаётся векторизации произвольными группами итераций.

По завершению всех указанных преобразований программы формируется граф алгоритма G=(V,F), каждой вершине V_i которого ставится в соответствие группа параметров: $V_i = [N_i, t_i, I(V_i), O(V_i), P_i, C_i]$ где N_i - номер вершины в графе; t_i - время её выполнения; $I(V_i), O(V_i)$ - множества входных и выходных переменных; P_i - тройка параметров (p_{i1}, p_{i2}, p_{i3}); C_i - вектор значений переменных цикла. Множеству дуг F соответствует множество векторов расстояний $F_{ij} = [R_{ij}]$.

Основная идея создания многоплатформенного транслятора – универсальность. Универсальность такого подхода заключается в том, что одна и та же задача, написанная на языке из используемого подмножества языков, имеет одно и то же представление на внутреннем языке транслятора. Последнее возможно, если представление исходной программы во внутренней форме транслятора близко к математическому алгоритму задачи, что в действительности имеет место. Другими словами, предлагаемый транслятор обеспечивает перевод с исходного языка в форму математического алгоритма. Это позволяет, используя полученный алгоритм, автоматически перевести его в программу, представленную на базовом языке любой платформы.

УДК 681. 658

А.Г. Цукерт

ПРОБЛЕМЫ И ПЕРСПЕКТИВЫ ИНФОРМАЦИОННОГО ПОИСКА

Автоматизация информационного поиска стала объективной необходимостью, обусловленной накоплением огромных фондов информации различного вида и потребностью в сокращении затрат времени на поиск нужной информации. Первые автоматизированные информационно-поисковые системы работали преимущественно с информацией фактического характера, например, характеристиками объектов и их связей. Со временем появилась возможность обрабатывать текстовые документы на естественном языке и другие форматы представления данных.