УДК 004.4

И.Ю. Шкулипа, С.Д. Погорелый

Автоматизированный синтез программ на основе *САА-М*-схем

Описаны особенности создания инструментария автоматизированного преобразования параллельных алгоритмов. Рассмотрены принципы создания автоматизированного генератора программного кода, на основе представления алгоритмов в виде формализованных *САА-М*-схем. Предложены подходы к практической реализации генератора программного кода.

The features of the automated tools for the algorithms transformation are described. The principles of creating an automated code generator, based on the presentation of the algorithms in the form of formalized SAA-M-schemes are considered. The approaches to the practical implementation of the code generator are suggested.

Описано особливості створення інструментарію автоматизованого перетворення паралельних алгоритмів. Розглянуто принципи створення автоматизованого генератора програмного коду, на основі подання алгоритмів у вигляді формалізованих *САА-М*-схем. Запропоновано підходи до практичної реалізації генератора програмного коду.

Введение. Одной из основных компонент системы автоматизированного параметрического преобразования алгоритмов [1, 2] является генератор кода программ на целевых языках программирования. Поскольку в описанной системе нотация алгоритмов основана на их представлении в виде формализованных *САА-М*-схем [3, 4], то создание такого генератора кода фактически сводится к построению компилятора, преобразующего *САА-М*-схему в программный код на целевом языке программирования.

В классической модели [5], процесс компиляции состоит из следующих этапов:

- лексический анализ: последовательность символов исходного файла преобразуется в последовательность лексем;
- синтаксический анализ: последовательность лексем преобразуется в дерево разбора;
- семантический анализ: дерево разбора обрабатывается с целью установления его семантики; здесь же происходит привязка идентификаторов к их декларациям, типам, проверка совместимости, определение типов переменных и др.;
- оптимизация: выполняется удаление излишних конструкций и упрощение кода с сохранением его смысла;
- генерация кода: из промежуточного представления порождается код на целевом языке.

В конкретных реализациях компиляторов эти этапы могут быть разделены или совмещены в том или ином виде.

Все упомянутые этапы генерации программного кода происходят последовательно в указанном порядке. В данной реализации опускается только этап оптимизации, так как он реализован в компиляторах целевых языков программирования.

Особенности реализации лексического анализа

На фазе лексического анализа исходная схема, представляющая собой поток символов, разбивается на лексемы [5, 6]. В процессе выделения лексем лексический анализатор выдает значения для каждой лексемы при очередном к нему обращении.

Определим грамматику языка внутренней нотации *САА-М*-схем в системе и введем некоторые условные обозначения.

Файл схемы должен начинаться с названия схемы и знака « = », все, что находится после этого знака, считается собственно схемой. Идентификатор названия алгоритма должен соответствовать требованиям записи идентификаторов языка *C*.

Имена операторов содержат только буквы латинского алфавита верхнего регистра (*A*…*Z*), цифры (1…9) и знак нижнего подчеркивания (_). Приведем примеры имен операторов:

- корректные операторы – *STEP*1, *A_B*, *R*, *INC_I*, *DEC_J*;
- некорректные операторы – 1*S*, 123, 4*Ht*.

Имена условий содержат только буквы латинского алфавита нижнего регистра (*a*…*z*), цифры (1…9) и знак нижнего подчеркивания (_). Имя условия должно начинаться с буквы. Примеры имен условий таковы:

• корректные условия – *a*1, *alpha*, *beta*, *a_less_than_b*;

• некорректные условия – 2*e*, 123, 5*Ab*, *Alpha*.

Для записи операций *CAA*, используются следующие обозначения:

* – последовательное выполнение операторов (*A* * *B*);

| – синхронная дизъюнкция (*A* | *B*);

|| – асинхронная дизъюнкция, или параллельное выполнение операторов (*A* || *B*);

a (*A* * *B* * *C*) – условный оператор (*if* (*a*) {*A*; *B*; *C*;});

a (*A* * *B*, *C* * *D*) – оператор альтернативы (*if* (*a*) {*A*; *B*;} *else* {*C*; *D*;});

a {*A* * *B* * *C*} – циклический оператор (*while* (*a*) {*A*; *B*; *C*;});

–*a*–* *A* – *a* – фильтрация.

Для записи булевых операций, выполняемых над условиями, используются следующие обозначения:

! – логическое отрицание (*NOT a*);

& – логическое «и» (*a AND b*);

@ – логическое «или» (*a OR b*);

^ – исключающее «или» (*a XOR b*).

Для группирования условий используются символы «[» и «]». Например, запись «![*A* ^ *b*]» – корректное условие.

На этапе лексического анализа обнаруживаются некоторые простейшие ошибки: недопустимые символы, неправильная запись идентификаторов и др. Проверка корректности записи схемы на этапе лексического анализа проводится по следующим критериям:

• в схеме отсутствуют любые символы, не входящие в алфавит принятых условий и обозначений;

• количество открытых скобок определенного типа (круглых, квадратных или фигурных) должно быть равно количеству закрывающих скобок того же типа;

• идентификаторы условий и операторов должны соответствовать принятым условным обозначениям.

Если условия корректности записи не выполняются, то выдается сообщение об ошибке.

Особенности реализации синтаксического анализа

Основополагающий этап процесса генерации кода – построение дерева разбора *CAA-M*-схемы алгоритма. Основной компонент генератора кода – синтаксический анализатор – часть генератора, анализирующий входную схему алгоритма и представляющий ее в виде дерева разбора. Дерево разбора схемы фактически представляет собой блок-схему результирующей программы, реализующей заданный алгоритм, без интерпретации операторов и условий, обозначенных идентификаторами *CAA-M* при проектировании исходной схемы.

Дерево разбора схемы состоит из вершин, описываемых следующей структурой:

• имя – поле текстового типа, содержащее название оператора или условия *CAA-M*;

• тип – поле типа «тип вершины», описывающее тип конструкции *CAA-M*;

• количество соседей – количество вершин, напрямую связанных с данной;

• соседи – массив ссылок на вершины–соседи данной;

• уровень – задает уровень вложенности вершины.

Поле «Тип», задающее тип вершины дерева, может принимать значения:

• оператор – вершина, соответствующая определенному оператору;

• условие – вершина дерева, соответствующая определенному условию;

• цикл (начало) – вершина, определяющая начало цикла; сигнализирует генератору кода о том, что следующая вершина должна быть условием;

• цикл (конец) – вершина, задающая конец цикла;

• условный оператор (начало) – вершина, соответствующая началу условного оператора; сигнализирует генератору кода о том, что следующая вершина должна быть условием;

• условный оператор (конец) – вершина, задающая конец условного оператора;

• асинхронная дизъюнкция – асинхронное параллельное выполнение операторов;

• синхронная дизъюнкция – синхронное параллельное выполнение операторов, свидетельствует о наличии в этом месте схемы критической секции;

• фильтр – вершина, соответствующая реализации операции фильтрации *САА-М*;

• булева операция – задает операцию булевой алгебры;

• параллельная часть (начало) – сигнализирует генератору кода о том, что в следующем фрагменте кода присутствуют параллельные операции;

• параллельная часть (конец) – задает конец параллельной части программы.

Следует отметить, что два последних типа вершин введены исключительно для удобства построения дерева разбора параллельной части программы и могут отсутствовать в исходной схеме в явном виде.

Поле «Уровень» используется для задания отступов в сгенерированной программе и определяется количеством предшествующих открытых блоков типа *цикл* и *условный оператор*.

Описанная структура дерева разбора позволяет значительно упростить последующий этап генерации кода программы, а так же сделать программу более простой для чтения, используя отступы в тексте программы, на основе поля «Уровень».

Особенности реализации семантического анализа

Генератор кода решает центральную задачу компилятора: превращает лексемы в последовательности команд целевого языка программирования, выполняя это в процессе обхода синтаксического дерева. Из узлов дерева в определенном порядке извлекаются лексемы. Когда набирается количество, достаточное для определения нужной операции, происходит генерация соответствующих конструкций языка, и обход дерева возобновляется. В этой реализации генератора кода в качестве операций предусмотрены все основные конструкции структурного программирования, кроме циклов *for* и *do..while*, эти циклы при проектировании *САА-М*-схемы должны быть заменены соответствующими конструкциями с использованием цикла *while*.

Как и на предыдущих этапах трансляции, обход дерева осуществляется в соответствии с правилами языка, используемого для внутренней нотации *САА-М*-схем в системе. Обход начинается с вершины дерева и продолжается в направлении вниз и вправо по его периметру. Когда встречается родительский узел, перебираются по очереди все его дочерние узлы. Если дочерний узел также является родительским, то перебираются и его дочерние узлы. Так происходит спуск по дереву, пока не встретится узел, не имеющий ответвлений. Из такого узла берется лексема и рассматривается следующий узел. В результате перебираются все узлы на самом низком уровне в этой ветке дерева. Тогда происходит возврат к наивысшему родительскому узлу, который еще не был обработан, и процесс повторяется. В результате, компилятор обходит дерево всей программы, генерируя код и записывая его в текстовый редактор кода для последующего выполнения или обработки.

Генератор программного кода обладает значительной параметризацией для поддержки генерации программ на различных целевых языках программирования и с применением различных парадигм параллельного программирования. Реализованы парадигмы параллельных процессов и потоков языка *C*, а также парадигмы, основанные на использовании технологий *MPI* и многопоточности в *Java*.

Сопоставления операторов

Одним из важнейших этапов генерации кода программы считается сопоставление операторов *САА-М*-схемы их реализациям на целевом языке программирования. Для корректной работы сгенерированной программы необходимо иметь реализации всех операторов, используемых в исходной *САА-М*-схеме, на целевом языке программирования.

Сопоставление операторов происходит в автоматизированном режиме, на этапе синтеза *САА-М*-схемы из *UML*-диаграммы. Модуль генерации *САА-М*-схем из *UML*-диаграмм – дополнительная компонента системы, функция которой – автоматизированный синтез *САА-М*-схемы алгоритма из спроектированной заранее *UML*-диаграммы. Для ситуаций, когда такой

вариант сопоставления невозможен за отсутствием конкретизаций операторов в исходном *UML*-коде или при возникновении ошибок при синтезе *CAA-M*-схемы, предусмотрена возможность редактирования конкретизаций операторов вручную, при помощи встроенного редактора сопоставлений операторов.

Копия экрана редактора сопоставлений операторов представлена на рис. 1. Из списка операторов алгоритма выбирается необходимый оператор. Затем автоматически подгружаются переменные, с которыми он работает. Затем следует расставить необходимые переменные в нужном порядке их передачи выбранному оператору. В абстрактном представлении оператора в базе данных, переменные обозначаются фиксированной приставкой *var*, за которой следует порядковый номер переменной в списке формальных параметров оператора.

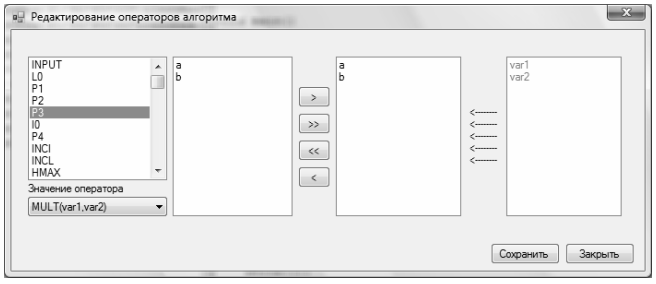

Рис. 1

Для большей гибкости и простоты использования применяется принцип генерации макроопределений. Таким образом, операторы вставляются в текст программы в виде макроопределений, которые затем используются в основном коде. Использование макроопределений позволяет фактически охватить весь спектр *C*-подобных языков программирования и значительно упростить добавление поддержки нового целевого языка.

Операторы системы хранятся в базе данных в виде набора макроопределений и их реализаций на целевых языках программирования, с дополнительными описаниями типа оператора, количества переменных, текстовых описаний операторов и других параметров. Набор операторов при необходимости может быть дополнен или исправлен. На этапе синтеза *CAA-M*-схемы на основе *UML*-диаграммы, если в *UML*-представлении алгоритма встречается оператор, отсутствующий в базе данных, то такой оператор добавляется в базу данных с автоматически присвоенным именем макроопределения и набором остальных параметров, которые могут быть изменены в процессе работы.

На рис. 2. представлена копия экрана редактора набора операторов. Этот редактор позволяет в ручном режиме редактировать набор операторов, добавлять новые или изменять старые операторы.

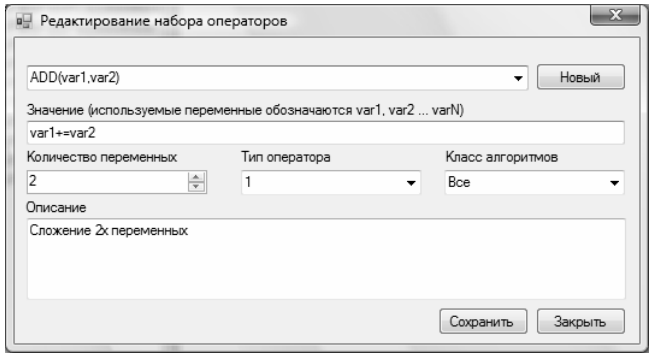

Рис. 2

Реализации операторов сохраняются в таблице базы данных. Запись в таблицу выполняется на этапе проектирования схемы алгоритма. Предусмотрено два варианта сопоставления аналогов операторов: на основе *UML*-схемы исходного алгоритма и вручную на этапе загрузки *CAA*-схемы алгоритма в систему с помощью встроенного редактора сопоставлений операторов.

В базе данных сопоставление операторов реализовано в виде двух связанных таблиц.

Таблица 1. Структура операторов алгоритма

Поле	Тип	Значение
ID	Счетчик	Индекс
Имя	Текст	Имя оператора *CAA-M*
Оператор	Числовой	*ID* оператора
Переменные	Текст	Список *ID* переменных данного оператора
Схема	Числовой	*ID* схемы алгоритма

Такая структура таблиц операторов позволяет повысить гибкость использования операторов. Также благодаря такой структуре таблиц появляется возможность накопления набора операторов для дальнейшего использо-

вания без необходимости заново редактировать операторы.

Таблица 2. Структура таблицы набора макроопределений операторов на целевых языках программирования

Поле	Тип	Значение
ID	Счетчик	Индекс
Имя	Текст	Имя оператора
Тип	Текст	Тип оператора
Переменные	Числовой	Количество переменных оператора
Класс алгоритмов	Текст	Класс алгоритмов, в которых используется оператор
Язык	Числовой	*ID* целевого языка программирования
Значение	Текст	Реализация оператора на языке программирования
Описание	Текст	Комментарий

Переменные

При проектировании *CAA-M*-схемы алгоритма также необходимо описать все переменные, используемые в его реализации. Этот процесс так же, как и сопоставления операторов, может происходить в автоматизированном режиме, на этапе синтеза *CAA-M*-схемы из *UML*-диаграммы. В системе предусмотрена возможность редактирования переменных алгоритма вручную с помощью встроенного редактора переменных.

На рис. 3. представлен снимок экрана редактора переменных, позволяющий редактировать набор переменных алгоритма, задавать их тип и начальное значение.

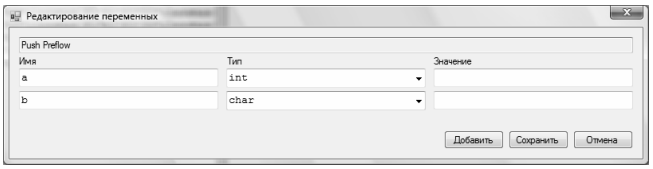

Рис. 3

Переменные алгоритма хранятся в базе данных системы в виде таблицы (табл. 3). Таблица переменных связана с таблицей операторов, посредством поля «Переменные» таблицы операторов алгоритма.

Таблица 3. Структура переменных алгоритма

Поле	Тип	Значение
ID	Счетчик	Индекс
Имя	Текст	Имя переменной
Тип	Текст	Тип переменной
Значение	Текст	Начальное значение переменной (если задано)
Схема	Числовой	*ID* схемы алгоритма

Рассмотрим пример использования макроопределений с соответствующими переменными. Для расчета формулы $c = (a - 1)(b + 1)$ введем *CAA*-схему алгоритма расчета данной формулы.

$$Formula=A1*B1*C$$

Схема, приведенная выше, представляет собой композицию трех простых операторов $A1*B1*C$, где

$A1$ – уменьшение переменной a на единицу,

$B1$ – увеличение переменной b на единицу,

C – присваивание переменной c значения произведения a и b.

Дерево разбора для этого примера будет иметь вид, представленный на рис. 4. Дерево фактически будет представлять собой структуру последовательного выполнения трех операторов.

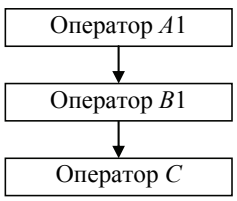

Рис. 4

Для реализации данного алгоритма понадобится определить четыре оператора: инкремент, декремент, умножение и присваивание. Все эти операторы сохраняются в базе данных системы, и могут быть использованы также и в других алгоритмах.

После генерации программы, ее декларативная часть будет иметь следующие определения:

```
#define INC(var1) var1++
#define DEC(var1) var1—
#define MULTIPLY(var1,var2) var1*var2
#define ASSIGN(var1,var2) var1=var2
```

Соответственно основная функция *C*-программы будет выглядеть так:

```
void main()
{
 DEC(a);
 INC(b);
 ASSIGN(c, MULTIPLY(a,b));
}
```

Внешние библиотеки

Зачастую реализация алгоритмов на *C*-подобных языках программирования не может обойтись исключительно встроенными операторами и типами языка [7]. Поэтому в системе предусмотрена возможность подключения внешних библиотек и вставки ссылки на них в сгенерированную программу. Предусмотрено три варианта вставки ссылок на внешние библиотеки: вставка стандартного набора библиотек, которые предусмотрены системой (например, *stdlib*, *math* и др.), выбор необходимых библиотек из списка, предусмотренного в базе данных, перед генерацией программы и вставка необходимых библиотек вручную при редактировании кода результирующей программы. Преимущество первого варианта заключается в исключении вмешательства со стороны пользователя системы в процесс генерации программ, но при таком варианте значительно перегружается код программы и увеличивается время ее компиляции и сборки. Этого недостатка лишены второй и третий варианты вставки внешних библиотек в код программы. В этих случаях будут вставлены ссылки только на необходимые библиотеки.

Заключение. Описанные подходы к построению компилятора схем в программный код успешно реализованы в виде одной из основополагающих компонент автоматизированной системы преобразования алгоритмов и их программных реализаций.

1. *Погорелый С.Д.*, *Шкулипа И.Ю.* Концепция создания автоматизированной параметрической системы проектирования параллельных алгоритмов и их программных реализаций // Кибернетика и системный анализ. – 2009. – № 6. – С. 118–124.
2. *Шкуліпа І.Ю.*, *Погорілий С.Д.* Методика автоматизованої трансформації схем алгоритмів // Проблеми програмування. – Матеріали міжнар. конф. УкрПРОГ, 2010.
3. *Многоуровневое* структурное проектирование программ / Е.Л. Ющенко, Г.Е. Цейтлин, В.П. Грицай и др. – М.: Наука, 1989. – 208 с.
4. *Алгоритмічні* алгебри / К.Л. Ющенко, С.В. Суржко, Г.О. Цейтлін та ін. – К.: Наук. думка, 1997. – 480 с.
5. *Ахо А.*, *Сети Р.*, *Ульман Дж.* Компиляторы. Принципы, технологии, инструменты. – М.: Вильямс, 2003. – 768 с.
6. *Хопкрофт Дж.*, *Мотвани Р.*, *Ульман Дж.* Введение в теорию автоматов, языков и вычислений. – М.: Вильямс, 2002. – 528 с.
7. *Погорілий С.Д.* Програмне конструювання: Підручник. – К.: ВПЦ Київський ун-т, 2005. – 438 с.

Поступила 17.03.2010

Тел. для справок: (097) 389-9392 (Киев)
E-mail: *Igor.Shkulipa@gmail.com*, *SDP@rpd.univ.kiev.ua*

●