МЕТОД ПРЕОБРАЗОВАНИЯ ПРОГРАММ В СИСТЕМУ ВЗАИМОДЕЙСТВУЮЩИХ АВТОМАТОВ

Г.А. Корнеев

Научный руководитель – д.т.н., профессор А.А. Шалыто

В работе предлагается формальный метод преобразования программы на алгоритмическом языке высокого уровня в систему взаимодействующих автоматов, обеспечивающую возможность трассировки исходной программы как вперед, так и назад.

Введение

С 1991 г. в России развивается SWITCH-технология, которая базируется на автоматном программировании [1]. При развитии этой технологии встает вопрос о формальных методах построения конечных автоматов или их систем по программам на императивных процедурных языках программирования. В работе [2] был предложен метод преобразования программ, не содержащих рекурсии. В дальнейшем это метод был развит, что позволило преобразовывать рекурсивные программы [3].

Отметим, что в области аппаратного обеспечения эта задача рассматривалась уже в 70-х годах, и ее решение для одной процедуры приведено в работе [4].

Во всех указанных работах строился один автомат, который позволял осуществлять трассировку только в прямом направлении, в то же время возможность трассировки назад в некоторых приложениях является весьма важной [5]. Кроме того, рассматривались только программы, состоящие из одной процедуры.

В работе [6], при участии автора, был предложен метод преобразования программы из произвольного числа рекурсивных процедур в систему взаимодействующих автоматов. Получаемая система автоматов обеспечивает трассировку исходной программы как в прямом, так и в обратном направлении.

В настоящей статье предлагается основанный на работе [6] формальный метод построения такой системы автоматов по программе. Для системы автоматов, построенной в результате применения предлагаемого метода, доказывается, что трассировка в обоих направлениях осуществляется корректно.

Язык преобразуемой программы

Будем преобразовывать программы, написанные на языке, порождаемом следующей грамматикой:

1	*Программа*	::=	*Процедура Программа*
2		|	*Процедура*
3	*Процедура*	::=	*Операторы*
4	*Операторы*	::=	*Операторы Оператор*
5		|	
6	*Оператор*	::=	*ОператорПрисваивания*
7		|	*ОператорВетвления*
8		|	*ОператорЦикла*
9		|	*ВызовПроцедуры*
10	*ОператорПрисваивания*	::=	*Переменная* = *Выражение*
11	*ОператорВетвления*	::=	*Выражение* *Операторы*$_1$ *Операторы*$_2$
12	*ОператорЦикла*	::=	*Выражение Операторы*
13	*ВызовПроцедуры*	::=	*Процедура*

С одной стороны, данный язык является достаточно простым для формального описания процесса преобразования, а с другой – содержит операторы присваивания,

ветвления, цикла и блочный оператор, и как следствие, является достаточно богатым для описания произвольной программы [7].

Будем предполагать, что при вычислении выражений нет побочных эффектов (значения переменных изменяются только операторами присваивания).

Автоматы и их свойства

Опишем формальный метод преобразования программы в систему взаимодействующих автоматов. Построенная система будет содержать по два автомата для каждой процедуры: *прямой* (для трассировки вперед) и *обратный* (для трассировки назад). Автоматы, построенные по одной процедуре, имеют общее множество состояний, но разные функции переходов [6].

Введем понятие *фрагмент автомата* – набор состояний и переходов. При этом у некоторых переходов может быть не определено начальное или конечное состояние. Такие переходы называются *входами* и *выходами* фрагмента соответственно. Заметим, что один переход может быть входом и выходом одновременно. В дальнейшем будем рассматривать фрагменты автоматов с одним входом и одним выходом.

Рассмотрим дерево синтаксического вывода преобразуемой программы. Для каждой его вершины построим фрагменты прямого и обратного автоматов, соответствующих поддереву с корнем в данной вершине. Построение будем осуществлять в порядке выхода из вершин при обходе дерева в глубину. При этом фрагменты для всех потомков вершины будут построены до построения фрагмента для этой вершины.

Для построенной системы автоматов докажем следующие свойства.

Адекватность – система автоматов выполняет те же действия, что и исходная программа.

Обратимость – при выполнении действий обратного автомата будут восстановлены исходные значения всех переменных при условии, что в точке входа они были такими же, как и в точке выхода при прямом проходе.

Полнота – для каждого состояния, не являющегося конечным, при любых значениях переменных условие на одном из переходов должно быть истинно.

Непротиворечивость – условия на переходах из одного состояния не могут быть истинными одновременно.

Отсутствие недостижимых состояний – любое состояние может быть достигнуто по переходам из начального состояния. При этом состояние называется достижимым, если оно начальное или в него существует переход из достижимого состояния. Таким образом, проверяется только принципиальная возможность достижения состояний, а условия на переходах фактически игнорируются.

Доказательство *адекватности* и *обратимости* позволяет утверждать, что трассировка в обоих направлениях осуществляется корректно. *Полнота* и *непротиворечивость* обеспечивают детерминированность построенных автоматов. *Отсутствие недостижимых состояний* обычно является естественным требованием.

Указанные свойства будем доказывать посредством структурной индукции на дереве вывода преобразуемой программы. Таким образом, предполагая, что эти свойства выполняются для фрагментов, соответствующих потомкам вершины, будем доказывать их для фрагмента, соответствующего вершине.

Для фрагментов автоматов будем рассматривать достижимость состояний от входа фрагмента (напомним, что рассматриваются фрагменты с одним входом и одним выходом). При этом достижимости выхода будет соответствовать достижимости состояния, из которого он выходит.

Для описания состояний автоматов будем использовать нотацию, описываемую следующей грамматикой:

Состояние ::= *ПереходыВ* < *Действия* > *ПереходыИз*
Действия ::= *Действие; Действия*
| *Действие*
ПереходыВ ::= *ПереходВ*
| *ПереходВ*; *ПереходыВ*
ПереходВ ::= *ИмяФрагмента* {, *Действие*}
ПереходыИз ::= *ПереходИз*
| *ПереходИз*; *ПереходыИз*
ПереходИз ::= *Условие*, *ИмяФрагмента*
Действие ::= *Переменная* = *Выражение*
| push(*Выражение*)
| *Переменная* = pop()

Заметим, что переход во фрагмент автомата не порождает новый переход, а лишь «замыкает» вход соответствующего фрагмента на описываемое состояние. Это же верно и для перехода из фрагмента.

Выстраиваемая система автоматов использует общий стек, над которым производятся следующие операции:

- push(expr) – поместить значение выражения expr на вершину стека;
- pop() – прочитать значение на вершине стека и удалить его.

Если для перехода в состояние указан фрагмент *Вход*, то это переход является входом фрагмента. Аналогично, если для перехода из состояния указан фрагмент *Выход*, то он является выходом фрагмента.

Действие на переходе указывается при определении исходящего перехода. Доказательство свойства обратимости будем осуществлять на границах переходов, когда действия, указанные на переходах уже выполнены.

Начнем описание процесса с отдельных операторов (продукции 10–13), затем перейдем к последовательностям операторов (продукции 4–9), а от них – к процедурам (продукции 1–3).

Преобразование отдельных операторов

Оператор присваивания

Фрагменты прямого и обратного автоматов для оператора присваивания состоят из одного состояния каждый.

Преобразуемая продукция:

10 *ОператорПрисваивания* ::= *Переменная* = *Выражение*

Состояние прямого автомата в используемой нотации имеет вид:

Вход < push(*Переменная*); *Переменная* = *Выражение* > true, *Выход*

При этом состояние обратного автомата имеет вид:

Вход < *Переменная* = pop()> true, *Выход*

Здесь true обозначает истину. Таким образом, оба перехода являются безусловными.

Докажем выполнение рассматриваемых свойств для оператора присваивания. Здесь и далее начало и конец доказательства помечаются значками ► и ◄ соответственно.

Адекватность. ► При выполнении действий в состоянии значение *переменной* сохраняется в стеке, и ей присваивается значение выражения. Таким образом, на выходе из фрагмента, как и после выполнения оператора присваивания, значение *переменной* равно значению выражения. При этом вершина стека содержит значение *переменной* до выполнения присваивания.◄

Обратимость. ► На выходе из фрагмента обратного автомата значение *переменной* равно значению в вершине стека при входе во фрагмент. По индуктивному предположению на входе фрагмента вершина стека содержит значение, сохраненное при прямом проходе. Таким образом, на выходе переменная имеет значение, которое она имела при входе во фрагмент прямого автомата при прямом проходе, а стек имеет то же содержимое, что и при входе во фрагмент прямого автомата. ◄

Полнота и *непротиворечивость.* ► Из каждого состояния осуществляется безусловный переход. ◄

Отсутствие недостижимых состояний. ► Вход ведет непосредственно в добавленное состояние. ◄

Оператор ветвления

При преобразовании оператора ветвления добавляются два состояния: входное и выходное [6].

Преобразуемая продукция:

11 *ОператорВетвления* ::= *Выражение Операторы*$_1$ *Операторы*$_2$

Для прямого автомата добавляются состояния следующего вида:

1: *Вход* `< >` *Условие*`,` *Да*`;` ¬*Условие*`,` *Нет*

2: *Да*`, push(true);` *Нет*`, push(false) < > true`, *Выход*

Для обратного автомата:

1': *Да*`;` *Нет* `< > true`, *Выход*

2': *Вход* `< > pop(),` *Да'*; ¬`pop(),` *Нет'*

Номера, указанные для состояний, используются исключительно в доказательствах свойств и не переносятся в построенный фрагмент.

Построенные фрагменты ссылаются на фрагменты для операторов, выполняемых при истинности условия – *Да* (построен для ребенка *Операторы*$_1$) и его ложности – *Нет* (построен для ребенка *Операторы*$_2$). Фрагменты *Да'* и *Нет'* – соответствующие фрагменты обратных автоматов.

Докажем выполнение рассматриваемых свойств.

Адекватность. ► Если *Условие* истинно, то будет произведен переход во фрагмент *Да*, и на вершине стека будет значение `true`. В противном случае будут произведен переход во фрагмент *Нет*, а в вершине стека будет значение `false`. По индуктивному предположению действия, выполняемые фрагментами *Да* и *Нет*, соответствуют *Операторы*$_1$ и *Операторы*$_2$. ◄

Обратимость. ► При обратном проходе вход осуществляется в состояние 2'. Если на вершине стека значение находится значение `true`, то оно снимается со стека и осуществляется переход во фрагмент *Да'*. В противном случае после снятия значения со стека происходит переход во фрагмент *Нет'*.

По индуктивному предположению на входе в вершине стека содержится значение, помещенное туда при прямом проходе. Таким образом, фрагмент обратного автомата выбирается верно. По тому же индуктивному предположению после выхода из фрагментов *Да'* или *Нет'* значения всех переменных будут восстановлены. ◄

Полнота и *непротиворечивость.* ► Переход из состояний 2 (1') является безусловными. Переходы из состояний 1 (2') помечены взаимно обратными условиями. ◄

Отсутствие недостижимых состояний. ► Вход ведет непосредственно в состояние 1 (2'), из которого выходят переходы во фрагменты *Да* и *Нет* (*Да'* и *Нет'*). По индуктивному предположению от входов этих фрагментов достижимы их выходы. Следовательно, состояние 2 (1') также достижимо. ◄

Оператор цикла

При преобразовании цикла с предусловием к фрагменту добавляется новое состояние.

Преобразуемая продукция:

12 *ОператорЦикла* ::= *Выражение Операторы*

Состояние, добавляемое к прямому автомату:

Вход, `push(false)`; *Операторы*, `push(true)` <
> *Условие*, *Операторы*; ¬*Условие*, *Выход*

Состояние, добавляемое к обратному автомату:

Вход; *Операторы'* <> `pop()`, *Операторы'*; ¬`pop()`, *Выход*

Здесь *Операторы* и *Операторы'* — фрагменты прямого и обратного автоматов для тела цикла.

Докажем выполнение рассматриваемых свойств.

Адекватность. ► При входе во фрагмент в вершину стека помещается значение `false`. После этого, пока *Условие* истинно, осуществляется переход во фрагмент *Операторы*, на выходе из которого каждый раз в вершину стека помещается значение `true`. По индуктивному предположению фрагмент *Операторы* адекватен. Таким образом, *Условие* при очередной итерации будет иметь то же значение, что и при исполнении исходной программы. При этом стек будет содержать столько элементов со значением `true`, сколько было итераций цикла, и один элемент со значением `false`. ◄

Обратимость. ► При обратном проходе, пока на вершине стека лежит значение `true`, оно снимается и осуществляется переход во фрагмент *Операторы'*. При этом по индуктивному предположению при каждом попадании в добавленное состояние вершина стека будет содержать значение `true` или `false`, помещенное туда при прямом проходе. Следовательно, во фрагмент *Операторы'* будет осуществлено столько переходов, сколько их было выполнено во фрагмент *Операторы* при прямом проходе.

Во фрагменте обратного автомата на каждом переходе из добавленного состояния значение, записанное при прямом проходе, удаляется из вершины стека. После этого в стеке находятся те же значения, что и были при переходе в добавленное состояние на прямом проходе. ◄

Полнота и *непротиворечивость*. ► Переходы из добавленных состояний помечены взаимно обратными условиями. ◄

Отсутствие недостижимых состояний. ► Добавленные состояния достижимы непосредственно от входа. При этом один из переходов ведет во фрагмент *Операторы* (*Операторы'*). Следовательно, по индуктивному предположению все состояния этого фрагмента достижимы. ◄

Оператор вызова процедуры

Оператор вызова процедуры преобразуется во фрагмент из одного состояния. При этом предполагается, что любая процедура может быть рекурсивной.

Преобразуемая продукция:

13 *ВызовПроцедуры* ::= *Процедура*

Состояние, добавляемое к прямому автомату:

Вход, `child` = *Процедура*`(start)`; *Состояние*
`<child.stepForward()>`
¬`child.isAtEnd()`, *Состояние*; `child.isAtEnd()`, *Выход*

Состояние, добавляемое к обратному автомату:

Вход, `child` = *Процедура*`(end)`; *Состояние*
`<child.stepBackward()>`
¬`child.isAtStart()`, *Состояние*; `child.isAtStart()`, *Выход*

Здесь *Состояние* – добавляемое состояние; *Процедура*`(start)` – создание экземпляра прямого автомата для процедуры в начальном состоянии; *Процедура*`(end)` – создание экземпляра обратного автомата для процедуры в конечном состоянии; `child.stepForward()` – прямой шаг экземпляра автомата; `child.stepForward()`

– обратный шаг экземпляра автомата; `child.isAtStart()` – проверка, что автомат находится в начальном состоянии; `child.isAtEnd()` – проверка, что автомат находится в конечном состоянии.

Докажем выполнение рассматриваемых свойств при дополнительном предположении, что создаваемый экземпляр автомата адекватен и обратим. Это предположение будет доказано в разделе «Завершение доказательства».

Адекватность. ► После входа во фрагмент, пока созданный экземпляр автомата не придет в конечное состояние, производятся шаги вперед созданного экземпляра автомата. По дополнительному предположению это осуществляется корректно. ◄

Обратимость. ► При обратном проходе осуществляются обратные шаги экземпляра автомата до тех пор, пока тот не придет в исходное состояние. По дополнительному предположению после каждого такого шага переменные восстанавливают свои значения. Следовательно, при выходе они примут исходные значения. ◄

Полнота и *непротиворечивость.* ► Переходы из добавленных состояний помечены взаимно обратными условиями. ◄

Отсутствие недостижимых состояний. ► Вход ведет непосредственно в добавленное состояние. ◄

Преобразование последовательностей операторов

Продукция

5 *Операторы* ::=

определяет пустой оператор, результатом преобразования которого будет отдельный переход (фрагмент без состояний), являющийся одновременно входом и выходом. Для такого фрагмента выполняются все рассматриваемые свойства:

Адекватность и *Обратимость.* ► Действия не осуществляются. ◄

Полнота и *Непротиворечивость.* ► Фрагмент не содержит состояний. ◄

Отсутствие недостижимых состояний. ► Вход одновременно является выходом. ◄

В свою очередь, продукция

4 *Операторы* ::= *Операторы Оператор*

определяет последовательность операторов. Для преобразования вершины дерева, порожденной этой продукцией, необходимо объединить фрагменты, соответствующие *Операторам* (*Ф1* и *Ф1'*) и *Оператору* (*Ф2* и *Ф2'*). При этом добавлять новые состояния не требуется, а следует «замкнуть» выход *Ф1* (*Ф2'*) на вход *Ф2* (*Ф1'*).

Докажем, что при этом все рассматриваемые свойства также выполняются.

Адекватность. ► Выход из фрагмента *Ф1* является входом во фрагмент *Ф2*. Таким образом, операторы, преобразованием которых были получены фрагменты *Ф1* и *Ф2*, будут выполняться последовательно, и по индуктивному предположению будет получен корректный результат. ◄

Обратимость. ► По индуктивному предположению, после выхода из фрагмента *Ф2'* значения переменных будут восстановлены в промежуточные значения. Следовательно, после выхода из фрагмента *Ф1'* они будут восстановлены в исходные значения. ◄

Полнота и *непротиворечивость.* ► Состояния не добавляются. ◄

Отсутствие недостижимых состояний. ► По индуктивному предположению выходы фрагментов *Ф1* и *Ф2'* достижимы. Следовательно, достижимы входы фрагментов *Ф2* и *Ф1'*. Поэтому достижимы все состояния построенного фрагмента, а также его выход. ◄

Продукции 6–9 введены для удобства записи, и их преобразование совпадает с результатом преобразования левой части соответствующей продукции.

Преобразование процедур

Рассмотрим преобразования вершин дерева, соответствующих продукциям 1-3:

1 *Программа* ::= *Процедура Программа*
2 | *Процедура*
3 *Процедура* ::= *Операторы*

Продукции 1 и 2 определяют программу как последовательность процедур. Таким образом, результат преобразования программы – множество пар автоматов, получаемых при преобразовании процедур.

Продукция 3 определяет процедуру как последовательность операторов. Как указано в предыдущем разделе, последовательность операторов может быть преобразована во фрагмент автомата. Для построения автомата по процедуре к соответствующим фрагментам требуется добавить по два состояния – начальное и конечное.

Для прямого автомата:

начальное состояние: < > `true`, *Операторы*

конечное состояние: *Операторы* < >

Для обратного автомата:

начальное состояние: < > `true`, *Операторы'*

конечное состояние: *Операторы'* < >

Здесь *Операторы* и *Операторы'* – фрагменты прямого и обратного автоматов, соответствующие телу процедуры.

Докажем, что рассматриваемые свойства выполняются для построенного автомата.

Адекватность. ► По индуктивному предположению фрагмент *Операторы* выполняет действия, описанные в теле процедуры. ◄

Обратимость. ► По индуктивному предположению фрагмент *Операторы'* правильно обращает действия, описанные в теле процедуры. ◄

Полнота и *непротиворечивость*. ► В построенных автоматах из начальных состояний выходят безусловные переходы, а из конечного состояния переходы не выходят. ◄

Отсутствие недостижимых состояний. ► Из начального состояния непосредственно достижим вход фрагмента *Операторы* (*Операторы'*). Следовательно, по индуктивному предположению достижим и выход этого фрагмента, который ведет в конечное состояние. Таким образом, все состояния построенного автомата достижимы. ◄

Завершение доказательства

Так как для каждой из вершин выполняется индуктивное предположение (при условии, что оно выполняется для всех детей вершины, порожденной этой продукцией), то осталось доказать базу индукции. В дереве вывода листами могут быть только вершины, соответствующие операторам присваивания и оператору вызова процедуры. Для первого из них было приведено непосредственное доказательство выполнения рассматриваемых свойств.

Доказательства полноты, непротиворечивости и отсутствия недостижимых состояний для оператора вызова процедуры были приведены в явном виде. Следовательно, эти свойства выполняются для всех автоматов.

Доказательства *адекватности* и *обратимости* основывались на предположении истинности этих фактов для вызываемой процедуры в целом. Таким образом, для них доказательства следует дополнить.

► Рассмотрим дерево вызовов при трассировке программы посредством построенных автоматов. По построению, листы этого дерева не содержат вызовов других процедур. Заметим, что если экземпляр автомата не вызывал другие автоматы, то для

него *адекватность* и о*братимость* уже доказана по индукции (так как для этого не требуется доказывать базу для вызовов). Таким образом, у дерева можно «оборвать листья». При этом по доказанному выше все вызовы, соответствующие «оборванным» листьям, являются *адекватными* и *обратимыми*.

Заметим, что, если дерево вызовов конечно, то, последовательно «обрывая листья», можно оставить только корень, для которого также подходит приведенное доказательство. Поэтому в любой момент все выполненные действия являются *адекватными* и *обратимыми*. Таким образом, построенная система автоматов также является *адекватной* и *обратимой*. ◄

Отметим, что при преобразовании каждой вершины дерева во фрагменты автоматов добавляется не более двух состояний. Таким образом, суммарное количество состояний в полученной системе автоматов линейно по количеству вершины в дереве, а, следовательно, и по числу операторов в исходной программе.

Заключение

В работе рассмотрен формальный метод построения системы взаимодействующих автоматов по программе.

Для предложенного метода приведено доказательство *адекватности*, *обратимости*, *полноты*, *непротиворечивости* и *достижимости* всех состояний построенной системы автоматов. Полученная система содержит по два автомата (прямой и обратный) для каждой процедуры. При этом количество состояний в автоматах линейно по количеству операторов в соответствующей процедуре.

Предложенный метод является математической основой создания пакета *Vizi*, предназначенного для создания визуализаторов алгоритмов, используемых при обучении основам программирования и дискретной математики.

Литература

1. Шалыто А.А. SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.
2. Шалыто А.А., Туккель Н.И. Преобразование итеративных алгоритмов в автоматные. // Программирование. 2002. № 5. С.12–26.
3. Туккель Н.И., Шалыто А.А., Шамгунов Н.Н. Реализация рекурсивных алгоритмов на основе автоматного подхода // Телекоммуникации и информатизация образования. 2002. № 5.
4. Баранов С. И. Синтез микропрограммных автоматов (граф-схемы и автоматы). Л.: Энергия, 1979.
5. Казаков М.А., Столяр С.Е. Визуализаторы алгоритмов как элемент технологии преподавания дискретной математики и программирования // Международная научно-методическая конференция «Телематика-2000». СПб.: 2000. С.189–191.
6. Казаков М.А., Корнеев Г.А., Шалыто А.А. Метод построения логики работы визуализатора алгоритмов на основе конечных автоматов // Телекоммуникации и информатизация образования. 2003. №6. С. 27–58.
7. Грис Д. Наука программирования. М.: Мир, 1984.