УДК 519.683.2
М.С. Чушкин, аспирант,
Институт систем информатики им. А.П. Ершова,
630090, Новосибирск, пр. акад. Лаврентьева, 6, Россия
E-mail: chushkinm@rambler.ru

Система дедуктивной верификации предикатных программ

Разработан новый метод генерации формул тотальной корректности рекурсивных программ без циклов и указателей. Реализована система дедуктивной верификации, генерирующая формулы корректности. Разработана система правил для упрощения процесса генерации формул корректности. Истинность полученных формул проверяется с помощью SMT-решателя CVC3 и в системе PVS.

Ключевые слова: *Предикатное программирование, тотальная корректность программы, дедуктивная верификация, система автоматического доказательства PVS, SMT-решатель CVC3*

Язык предикатного программирования P определяет класс программ-функций, не взаимодействующих с внешним окружением [1]. Бесконечное исполнение таких программ бессмысленно, поэтому корректная программа на языке P всегда должна завершать свое исполнение. В языке P запрещены циклы и указатели.

На базе аппарата формальной операционной семантики разработан метод дедуктивной верификации предикатных программ. Определена формула тотальной корректности предикатной программы относительно своей спецификации, представленной предусловием и постусловием. Разработана система правил [2], позволяющая декомпозировать формулу тотальной корректности набором простых и коротких формул, упрощая тем самым последующее доказательство. Корректность правил доказана в системе PVS [3].

Система дедуктивной верификации включает: генератор формул корректности, транслятор формул во внутреннее представление SMT-решателя CVC3, транслятор формул на язык спецификаций системы PVS.

Генератор формул корректности строит формулы корректности программы, используя систему правил.

В процессе трансляции программы для каждого вхождения переменной (или выражения) реализуется проверка соответствия типа переменной (или выражения) типу той позиции, в которую находится переменная. Контроль соответствия типов не всегда можно провести статически при трансляции. При наличии ошибки несоответствия типов исполнение программы может оказаться неверным даже в том случае, когда доказана тотальная корректность программы. По этой причине для гарантии корректности программы необходимо проводить полный контроль типов.

В трансляторе с языка P реализована статическая проверка совместимости типов. Условия совместимости типов, которые транслятор не может проверить статически, добавляются к формулам корректности программы.

Все формулы проходят проверку в SMT-решателе CVC3. Формулы, которые решатель не смог доказать, оформляются в виде теории для системы PVS.

Система верификации успешно прошла апробацию в рамках университетского курса “Формальные методы в программной инженерии”.

Работа выполнена при поддержке РФФИ, грант № 16-01-00498.

Обзор работ

Методы верификации. Дедуктивная верификация программных систем в основном базируется на логике Хоара [4], определяющей набор правил вывода (аксиом) для троек Хоара. Последовательное применение

правил позволяет получить набор формул – условий частичной корректности императивной программы.

Доказательство завершения программы не реализуется в классической логике Хоара. Позже метод был расширен. Для доказательства завершения исполнения циклов и рекурсивных вызовов применяется аппарат вполне упорядоченных множеств [5]. Каждой рекурсивной функции и циклу приписывается переменная из вполне упорядоченного множества, значение которой должно строго убывать с каждой итерацией или вызовом.

Метод дедуктивной верификации предикатных программ отличается от метода Хоара. Он реализован для невзаимодействующих программ, исполнение которых должно всегда завершаться. Метод реализует доказательство тотальной корректности программы, а не частичной, как в методе Хоара.

В предикатном программировании нет циклов, вместо них используются рекурсивные предикаты. Верифицировать рекурсивный предикат проще, чем цикл. Построение инварианта цикла – нетривиальная задача. В предикатном программировании роль инварианта выполняет предусловие рекурсивного предиката, которое существенно проще инварианта цикла.

В методе Хоара для доказательства тотальности рекурсивных программ используется аппарат вполне упорядоченных множеств. В нашем подходе используется функция меры, строго убывающая на аргументах рекурсивных вызовов. Доказательство с использованием функции меры удобнее, чем доказательство с использованием переменных на упорядоченных множествах.

Разработан простой универсальный механизм обобщения правил для рекурсивных вызовов. Правила для элиминации квантора

существования значительно упрощают процесс построения формул корректности.

Слабейшее предусловие Дейкстры [6] определяет условие тотальной корректности программы. Слабейшее предусловие строится применением к программе преобразователя предикатов wp(S, R) – отображения из множества программ и постусловий во множество слабейших предусловий. Его применения к некоторой программе S приводит к разделению оной на команды, и последующему применению преобразователя предикатов к ним. В результате этого процесса получается формула – слабейшее предусловие.

Классический метод Хоара неприменим для программ с указателями. Метод separation logic [7] расширяет семантику императивного языка метода Хоара, описывая статическую и динамическую память. Язык расширяется командами для выделения и высвобождения динамической памяти. В работе была предложена концепция разделенной памяти, в которой память, выделенная одной структуре данных, не может пересекаться с памятью, выделенной для другой структуры данных. Одна из ключевых идей метода– это расширения стандартной логики Хоара оператором *, где P * Q означает, что динамическая память может быть разделена на две непересекающиеся части, для каждой из которых истинно одни из условий P или Q.

Поведение программы, оперирующей указателями, трудно для анализа. Методы анализа программ, в том числе и метод separation logic, базируются на различных моделях организации динамической памяти. В предикатном программировании нет указателей, вместо них используются алгебраические типы. Это существенно упрощает процесс доказательства корректности программы

Доказательство условий совместимости типов. Язык предикатного программирования P не единственный, где условия семантической корректности формализуются в виде логических формул. Аналогичный метод используется в системе PVS, где семантический контроль нетривиальных конструкций осуществляется через доказательство соответствующих формул корректности (TCCs – type correctness conditions).

Семантический анализатор автоматически строит условие совместимости типов. В дальнейшем система пытается автоматически их доказать. Если автоматически это сделать не удается, то пользователь должен доказать эти условия средствами системы PVS [8].

Предикатное программирование

Предикатная программа является предикатом в форме вычислимого оператора. Предикатная программа не взаимодействует с внешним окружением программы. Точнее, перед началом программы возможен ввод данных и лишь по ее завершению – вывод результатов. Программа обязана всегда завершаться, поскольку бесконечно работающая и невзаимодействующая программа бесполезна. Следовательно, программа определяет функцию, вычисляющую по набору аргументов некоторый набор результатов.

Базисные конструкции языка предикатного программирования P. Программа на языке предикатного программирования P (Predicate programming language) состоит из набора определений предикатов. Определение предиката имеет следующий вид:

```
<имя предиката>
(<список аргументов>: <список результатов>)
pre <предусловие>
{
        <оператор>
}
post <постусловие>
```

Необязательные конструкции предусловие и постусловие являются формулами на языке исчисления предикатов; они используются для улучшения понимания программ и для дедуктивной верификации [9-12]. Для любых значений аргументов, удовлетворяющих предусловию, значения результатов, в случае завершения оператора, должны удовлетворять постусловию.

Ниже представлены основные конструкции языка P: оператор присваивания, блок (оператор суперпозиции), параллельный оператор, условный оператор, вызов программы и описание переменных, используемое для аргументов, результатов и локальных переменных.

```
<переменная> = <выражение>
{ <оператор 1>; <оператор 2> }
<оператор 1> || <оператор 2>
if (<логическое выражение>) <оператор 1> else <оператор 2>
<имя предиката> (<аргументы>: <результаты>)
<тип> <список имен переменных>
```

В языке P запрещены такие языковые конструкции как циклы и указатели, серьезно усложняющие анализ программы. Полное описание языка P дано в работе [1].

Императивное расширение языка P. Императивное расширение языка P содержит дополнительные языковые конструкции: прерывание исполнения цикла (break), циклы for и while, метку и оператор безусловного перехода на метку. Семантика циклов for и while соответствует языку C++.

Эти конструкции возникают в программе в результате проведения трансформаций предикатной программы. Использование этих конструкций в исходной программе недопустимо.

Система предикатного программирования. Система предикатного программирования осуществляет трансляцию программы с языка P на язык C++ (Рис. 1). Синтаксический анализатор производит разбор кода программы на языке P и формирует ее образ во внутреннем представлении.

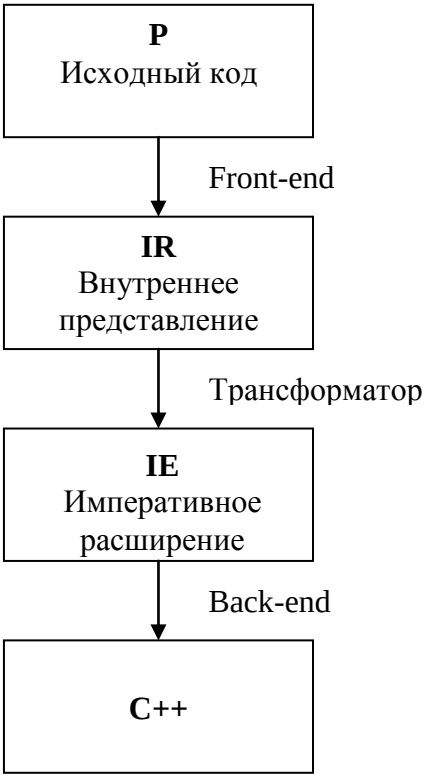

Рис. 1. Система предикатного программирования

Метод дедуктивной верификации предикатных программ

Семантика языка предикатного программирования P. *Предикатная программа* H(x: y) с аргументами x и результатами y есть

предикат в форме вычислимого оператора. *Операционную семантику* программы H(x: y) определим в виде предиката:

$\mathcal{R}(H)(x, y) \cong$ для значения набора x исполнение программы H всегда завершается и существует исполнение программы, при котором результатом вычисления является значение набора y.

Минимальный полный *базис предикатных программ* определен в виде языка $\mathbf{P_0}$. В таблице 1 представлен полный базис вычислимых предикатов и соответствующих им операторов.

H(x: y) ≅ ∃z. B(x: z) & C(z: y)	H(x: y) { B(x: z); C(z: y) }
H(x: y, z) ≅ B(x: y) & C(x: z)	H(x: y, z) { B(x: y) || C(x: z) }
H(x: y) ≅ (e ⇒ B(x: y)) & (¬e ⇒ C(x: y))	H(x: y) { **if** (e) B(x: y) **else** C(x: y) }
H(x: y) ≅ B(x˜ : y)	H(x: y) { B(x˜: y) }
H(A, x: y) ≅ A(x: y)	H(A, x: y) { A(x: y) }
H(x: D) ≅ ∀y,z. D(y: z) ≡ B(x, y: z)	H(x: D) { D(y: z) {B(x, y: z)} }
H(A, x: D) ≅ ∀y,z. D(y: z) ≡ A(x, y: z)	H(A, x: D) { D(y: z) {A(x, y: z)} }

Таблица 1. Вычислимые предикаты и их программная форма

Здесь x, y и z – разные непересекающиеся наборы переменных. Набор x может быть пустым, наборы y и z не пусты; B и C – имена предикатов, A и D – имена переменных предикатного типа. Набор x˜ составлен из набора переменных x с возможным добавлением имен предикатных программ.

Для языка $\mathbf{P_0}$ построена формальная операционная семантика и доказано тождество $\mathcal{R}(H) = H$ [13]. На базе языка $\mathbf{P_0}$ последовательным расширением и сохранением тождества $\mathcal{R}(H) = H$ построен язык предикатного программирования **P** [14]:

$$P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P$$

Корректность программы. Рассмотрим определение предиката следующего вида:

```
A(X x:Y y)
pre P(x)
{ S(x: y) }                                    (1)
post Q(x, y)
measure m(x)
```

Тотальная корректность программы (1) определяется истинностью следующей формулы:

$$Corr(H, P, Q)(x) \cong \forall x.\ P(x) \Rightarrow [\forall y.\ \mathcal{R}(H)(x, y) \Rightarrow Q(x, y)\ ] \ \&\ \exists y.\ \mathcal{R}(H)(x, y) \quad (2)$$

Здесь $\forall y\ (R(S)(x, y) \Rightarrow Q(x, y))$ является условием частичной корректности, т.е. условием того, что спецификация истинна при завершении программы. Подформула $\exists y\ R(S)(x, y)$ определяет условие завершение исполнения оператора S(x: y).

Далее термин «корректность» будем использовать в смысле тотальной корректности.

Система правил вывода формул корректности. Используя формулу тотальной корректности, можно автоматически построить формулу корректности для некоторого оператора S(x: y). Итоговая формула корректности будет длинной и сложной даже для коротких программ; она будет намного длиннее программы S(x: y). Специализация формулы тотальной корректности для разных видов операторов позволяет декомпозировать длинную формулу корректности к нескольким более коротким и простым формулам.

Определяется система правил вывода условий корректности для различных операторов. Доказательства правил приведены в работах [2, 14]. В дополнении к этому формальные доказательства корректности правил проведены в системе автоматического доказательства PVS; см. доказательства корректности правил в формате PVS [3].

Предположим, что наборы переменных x, y, z и v не пересекаются, а множества x и v могут быть пустыми. В таком случае, корректны следующие правила:

$$\textbf{QP:}\ \frac{\mathrm{Corr}(B, P, Q_B)(x);\quad \mathrm{Corr}(C, P, Q_C)(x);}{\mathrm{Corr}(B(x\colon y) \,||\, C(x\colon z), P, Q_B(x, y)\ \&\ Q_C(x, z))(x)}$$

$$\textbf{QS:}\ \frac{\begin{array}{l}P(x) \to \exists z, v\ R(B)(x, (z, v));\\ \forall z\ \mathrm{Corr}(C, (\ P(x)\ \&\ \exists z, v\ R(B)(x, (z, v))\ ), Q)(x, z);\end{array}}{\mathrm{Corr}(B(x\colon z, v);\ C(x, z\colon y), P, Q)(x)}$$

$$\textbf{QSB:}\ \frac{\begin{array}{l}\mathrm{Corr}^*(B, P_B, Q_B)(x);\quad P(x) \to P^*_B(x);\\ \forall z\ \mathrm{Corr}(C, \lambda x, z.\ (\ P(x)\ \&\ Q_B(x, z)\ ), Q)(x, z)\end{array}}{\mathrm{Corr}(B(x\colon z, v);\ C(x, z\colon y), P, Q)(x)}$$

$$\textbf{QC:}\ \frac{\begin{array}{l}\mathrm{Corr}(B, \lambda x.P(x)\ \&\ E(x), Q)(x);\\ \mathrm{Corr}(C, \lambda x.P(x)\ \&\ \neg E(x), Q)(x);\end{array}}{\mathrm{Corr}(\textbf{if}\ (E(x))\ B(x\colon y)\ \textbf{else}\ C(x\colon y))(x)}$$

Используя эти правила, доказательства корректности оператора суперпозиции, параллельного оператора и условного оператора можно свести к доказательству корректности их подоператоров B и C.

В правилах, описанных ниже, если подоператор B является рекурсивным вызовом, то посылка Corr*(B, …) опускается, а $P^*_B(x)$ заменяется на $P_B(x)$ & $m(x) < m(z)$, где z обозначает аргументы предиката B. Если же подоператор B не является рекурсивным вызовом, то Corr* и P* обозначают просто Corr и P. Вхождения Corr* и P* для подоператора C трактуются аналогично

$$\textbf{RP:}\ \frac{\begin{array}{l}\mathrm{Corr}^*(B, P_B, Q_B)(x);\quad \mathrm{Corr}^*(C, P_C, Q_C)(x);\\ \forall y, z\ (Q_B(x, y)\ \&\ Q_C(x, z) \to Q(x, y, z));\\ P(x) \to P^*_B(x)\ \&\ P^*_C(x);\end{array}}{\mathrm{Corr}(B(x\colon y) \,||\, C(x\colon z), P, Q)\ (x)}$$

$$\textbf{RS:}\ \frac{\begin{array}{l}\mathrm{Corr}^*(B, P_B, Q_B)(x);\quad \forall z\ \mathrm{Corr}^*(C, P_C, Q_C)(x, z);\\ \forall z, v, y\ (\ P(x)\ \&\ Q_B(x, z, v)\ \&\ Q_C(x, z, y) \rightarrow\ Q(x, y)\ )\\ \forall z, v\ (\ P(x)\ \&\ Q_B(x, z, v) \rightarrow\ P^*_C(x, z)\ );\\ P(x) \rightarrow\ P^*_B(x);\end{array}}{\mathrm{Corr}(B(x: z, v); C(x, z: y), P, Q)(x)}$$

$$\textbf{RC:}\ \frac{\begin{array}{l}\mathrm{Corr}^*(B, P_B, Q_B)(x);\quad \mathrm{Corr}^*(C, P_C, Q_C)(x);\\ P(x)\ \&\ E \rightarrow P^*_B(x);\quad P(x)\ \&\ \neg E \rightarrow P^*_C(x);\\ \forall y\ (P(x)\ \&\ E\ \&\ Q_B(x, y) \rightarrow Q(x, y));\\ \forall y\ (P(x)\ \&\ \neg E\ \&\ Q_C(x, y) \rightarrow Q(x, y))\end{array}}{\mathrm{Corr}(\textbf{if}\ (E(x))\ B(x: y)\ \textbf{else}\ C(x: y), P, Q)(x)}$$

Описанное ниже правило – это специализация правила RS. Вызов предиката B(x: z) может быть записан как z = B(x). Таким образом, мы можем использовать конструкцию C(B(x): y) как эквивалент оператора суперпозиции B(x, z); C(z: y).

Представленные выше правила принципиально упрощают формулы корректности рекурсивных программ.

$$\textbf{RB:}\ \frac{\begin{array}{l}\forall z\ \mathrm{Corr}^*(C, P_C, Q_C)(z);\\ P(x) \rightarrow P_B(x)\ \&\ P^*_C(B(x));\\ \forall y\ (\ P(x)\ \&\ Q_C(B(x), y) \rightarrow Q(x, y)\ );\\ \forall x, z_1, z_2\ P_B(x)\ \&\ R(B)(x, z_1)\ \&\ R(B)(x, z_2) \rightarrow z_1 = z_2;\end{array}}{\mathrm{Corr}(C(B(x): y), P, Q)(x)}$$

Последовательное применение вышеописанных правил к исходной формуле тотальной корректности декомпозирует ее на множество формул вида $W(x, y) \Rightarrow R(S)(x, y)$, где W(x, y) — произвольная посылка. Ниже даны правила доказательства формулы для различных видов операторов в позиции оператора S(x: y):

$$\textbf{FP:}\ \frac{\begin{array}{l}W(x, y, z) \rightarrow R(B)(x, y);\\ W(x, y, z) \rightarrow R(C)(x, z)\end{array}}{W(x, y, z) \rightarrow R(B \parallel C)(x, (y, z))}$$

$$\textbf{FS:}\ \frac{\begin{array}{l}W(x, y) \rightarrow \exists z\ R(B)(x, z);\\ W(x, y)\ \&\ R(B)(x, z) \rightarrow R(C)(z, y)\end{array}}{W(x, y) \rightarrow R(B; C)(x, y)}$$

$$\textbf{FC:}\ \frac{W(x, y)\ \&\ E \rightarrow R(B)(x, y);\quad W(x, y)\ \&\ \neg E \rightarrow R(C)(x, y)}{W(x, y) \rightarrow R(\textbf{if}\ (E)\ B\ \textbf{else}\ C)(x, y)}$$

Приведенные выше правила достаточно просты. Их можно применять многократно для декомпозиции вхождений R(S)(x, y) при доказательстве формул вида: W(x, y) → R(S)(x, y). Таких формул большинство среди посылок в приведенных выше правилах. Однако правило FS использует посылки двух других видов: W(x, y) → ∃y R(S)(x, y) и W(x, y) & R(S)(x, y) → H(x, y), где W(x, y) и H(x, y) — произвольные формулы. Ниже приведены правила для декомпозиции вхождений R(S)(x, y) в этих новых видах формул.

$$\textbf{EP:}\ \frac{W(x) \rightarrow \exists y\ R(B)(x, y);\quad W(x) \rightarrow \exists z\ R(C)(x, z)}{W(x) \rightarrow \exists y\ R(B \parallel C)(x, (y, z))}$$

$$\textbf{ES:}\ \frac{W(x) \rightarrow \exists z\ R(B)(x, z);\quad W(x)\ \&\ R(B)(x, z) \rightarrow \exists y\ R(C)(z, y)}{W(x) \rightarrow \exists y\ R(B; C)(x, y)}$$

$$\textbf{EC:}\ \frac{W(x)\ \&\ E \rightarrow \exists y\ R(B)(x, y);\quad W(x)\ \&\ \neg E \rightarrow \exists y\ R(C)(x, y)}{W(x) \rightarrow \exists y\ R(\textbf{if}\ (E)\ B\ \textbf{else}\ C)(x, y)}$$

Пусть A(x: y) — нерекурсивный вызов предиката, а P(x) — предусловие этого предиката. Вхождение логики A(x: y) под квантором существования декомпозируется следующим правилом:

$$\textbf{EB:}\ \frac{Corr(A, P, Q)(x);\quad \forall y\ (W(x) \rightarrow P(x))}{W(x) \rightarrow \exists y\ R(A)(x, y)}$$

Приведенная выше система правил позволяет элиминировать вхождение квантора существования. Вхождения логики оператора в левой части формулы, декомпозируются по следующим правилам:

$$\textbf{FLP:}\ \frac{W(x, y, z)\ \&\ R(B)(x, y)\ \&\ R(C)(x, z) \to H(x, y, z)}{W(x, y, z)\ \&\ R(B \parallel C)(x, (y, z)) \to H(x, y, z)}$$

$$\textbf{FLS:}\ \frac{W(x, y)\ \&\ R(B)(x, z)\ \&\ R(C)(z, y) \to H(x, y)}{W(x, y)\ \&\ R(B; C)(x, y) \to H(x, y)}$$

$$\textbf{FLC:}\ \frac{\begin{array}{l} W(x, y)\ \&\ E\ \&\ R(B)(x, y) \to H(x, y); \\ W(x, y)\ \&\ \neg E\ \&\ R(B(C))(x, y) \to H(x, y); \end{array}}{W(x, y)\ \&\ R(\textbf{if}\ (E)\ B\ \textbf{else}\ C)(x, y) \to H(x, y)}$$

$$\textbf{FLB:}\ \frac{P_A(x);\quad W(x, y)\ \&\ Q_A(x, y) \to H(x, y);}{W(x, y)\ \&\ R(A)(x, y) \to H(x, y)}$$

Модель системы правил вывода в PVS. Для доказательства корректности описанных выше правил была разработана модель в системе PVS.

Для операторов языка P_2 в модели определены теории. Внутри теорий определены входные и выходные параметры операторов, определены логики операторов. В модели представлены теории для условного оператора, параллельного оператора, для четырех видов оператора суперпозиции.

В модели определены теория для тотальности оператора и для однозначности спецификации. Определены теории для формул Corr. Каждое правило представляется в виде теоремы в теории на PVS.

Система верификации

Система верификации разрабатывалась как back-end в системе предикатного программирования. Основной компонентой системы верификации является генератор формул корректности программы, представленной во внутреннем представлении (рис. 2).

В трансляторе с языка P реализован статический контроль типов. Транслятор формирует условия совместимости и пытается автоматически их разрешить. В том случае, если условие совместимости типов не может

быть разрешено статически при трансляции, это условие генерируется в виде логической формулы для последующего доказательства в CVC3 или PVS. Данный механизм ранее был реализован в инструменте контроля динамической семантики [15]. Условия совместимости типов добавляются к сгенерированным формулам корректности программы.

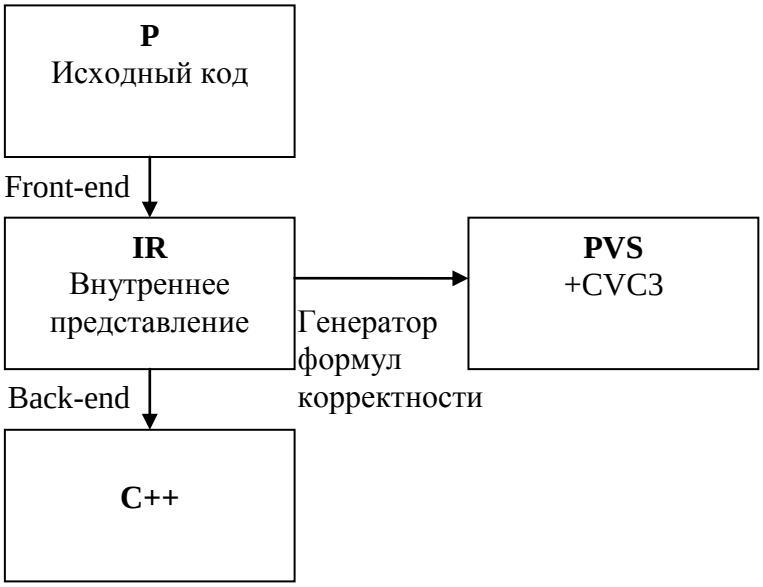

Рис. 2. Система верификации

Значительная часть генерируемых формул являются простыми для доказательства. Истинность таких формул проверяется автоматически с помощью решателя CVC3. Для доказательства более сложных формул корректности используется система интерактивного доказательства PVS.

Генерация формул корректности

Генерация формул корректности проходит по схеме, представленной на рисунке 3.

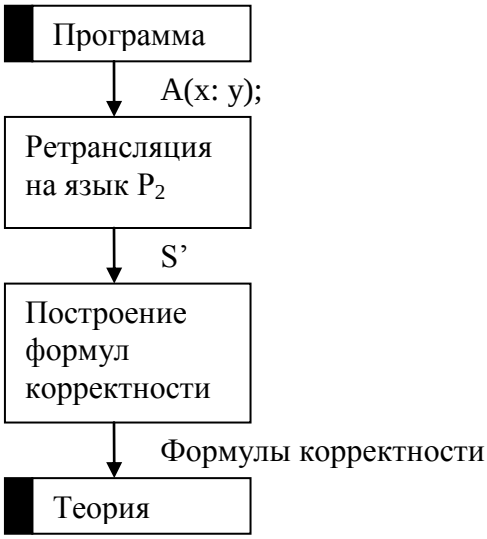

Рис. 3. Общая схема генерации условий корректности

Задачей алгоритма является автоматическое построение формул корректности предикатной программы. На языке P предикатная программа представлена набором определений предикатов. Корректность программы — это корректность всех определенных предикатов.

Из программы последовательно извлекаются определения предикатов вида (1). Для тела предиката, представленного оператором S, реализуется преобразование к канонической форме – это трансляция с языка P на язык P_2, существенно упрощающая структуру исходной программы.

Для преобразованного оператора происходит построение условий корректности.

На основе полученных условий корректности строится теория, которая содержит в себе, помимо самих условий, еще и определение необходимых формул и типов, а также ссылки на импортируемые теории.

Пример

Дадим иллюстрацию работы метода дедуктивной верификации на примере программы умножения двух натуральных чисел a и b, с использованием лишь операций сложения и вычитания.

Для того чтобы получить программу в форме хвостовой рекурсии и автоматически преобразовать ее в программу с циклом for на этапе компиляции, мы должны рассмотреть более общую задачу mult(a, b, d: c) со спецификацией [$a \geq 0$ & $b \geq 0$ & $d \geq 0$, $c = a * b + d$].

Ниже приведена предикатная программа умножения через сложение.

```
mult(nat a, b, d: nat c)
pre a ≥ 0 & b ≥ 0 & d ≥ 0
{
   if (a = 0)
      c = d
   else
      mult(a – 1, b, d + b: c)
}
post c = a * b + d
measure a;
```

Определим предусловие, постусловие и меру в виде отдельных формул:

```
formula P_mult(nat a, b, d) = a ≥ 0 & b ≥ 0 & d ≥ 0;
formula Q_mult(nat a, b, d, c) = c = a * b + d;
formula m(nat a: nat) = a;
```

Корректность предиката mult, по определению (2), выражается формулой:

```
Corr(mult, P_mult, Q_mult)(a, b, d)                    (3)
```

Таким образом, предикат mult будет корректен, если нам удастся доказать истинность формулы (3). В рамках эксперимента, можно

раскрыть эту формулу по определению и попытаться доказать. Это будет намного сложнее по сравнению с нашим методом, описанным ниже.

Разложим формулу (3), заменив в ней вхождение предиката mult на его тело, представленное условным операторам. Применение правила RC порождает две новые цели:

$$Corr(c = d, \lambda a, b, d.\ P\_mult(a, b, d) \ \&\ a = 0\ ,\ Q\_mult) \quad (4)$$

$$Corr(mult(a - 1, b, d + b: c),\ \lambda a, b, d.\ P\_mult(a, b, d) \ \&\ \neg\ a = 0, Q\_mult) \quad (5)$$

Формула (4) раскрывается по определению (2) в виде следующей леммы:

$$\textbf{lemma}\ P\_mult(a, b, d) \ \&\ a = 0 \ \&\ c = d \Rightarrow Q\_mult(a, b, d, c); \quad (6)$$

К формуле (5) применяется правило RB, где $B(a, b, d) = |a - 1, b, d + b|$ и $PB = a - 1 \geq 0$. Первая посылка правила RB опускается ввиду рекурсивности предиката mult. Вторая посылка тождественно истинна. Третья и четвертая посылки записываются в виде следующих лемм:

$$\textbf{lemma}\ P\_mult(a, b, d) \ \&\ \neg\ a = 0 \Rightarrow a - 1 \geq 0 \ \&\ m(a - 1) < m(a); \quad (7)$$

$$\textbf{lemma}\ P\_mult(a, b, d) \ \&\ \neg\ a = 0 \ \&\ Q\_mult(a - 1, b, d + b, c) \Rightarrow\ Q\_mult(a, b, d, c); \quad (8)$$

Доказательство корректности предиката mult сводится к доказательству истинности трех описанных выше лемм. Эти леммы проверяются решателем CVC3.

Для оценки эффективности метода проведем сравнение описанного процесса верификации предиката mult с классическим процессом верификации по Хоару для императивной программы, полученной трансляцией предиката mult на императивное расширение языка P, в ходе которой хвостовая рекурсия заменяется циклом **while**. В качестве

инструмента воспользуемся платформой Why3. Императивная программа, соответствующая предикату mult, на языке WhyML представляется следующим образом:

```
module Mul

use import int.Int
use import ref.Ref

predicate pre (x y z: int) =
        x >= 0 /\ y >= 0 /\ z >=0

predicate post (x y z r: int) =
        r = x*y + z

predicate inv (x y z r: int) =
        r = (x - y)*z

let mul (a0 b d: int)
        requires { pre a0 b d }
        ensures { post a0 b d result }
=
        let c = ref 0 in
        let a = ref a0 in
        while !a <> 0 do
                invariant { inv a0 !a b !c }
                a := !a - 1;
                c := !c + b;
        done;
        (!c + d)
end
```

Отметим, что введение дополнительной переменной a0 и оператора a := a0 необходимо для построения инварианта (a0 – a)*b = c.

Why3 автоматически строит теорию для доказательства корректности функции mul. Для удобства сравнения формулы корректности программы mul приведены здесь к виду, используемому в системе PVS:

pre(a0, b, d) => inv(a0, a0, b, 0) (9)

```
pre(a0, b, d) & inv(a0, a, b, c) & a != 0
        => inv(a0, a - 1, b, c + b)                    (10)
pre(a0, b, d) & inv(a0, a, b, c) & a = 0
        => post(a0, b, d, c + d)                       (11)
```

Наборы формул корректности для программ mult и Mul принципиально различаются; совпадающих формул нет. Формула (9) аналогична второй посылке правила RB, истинность которой определена на стадии построения формул. Исключим из рассмотрения формулу (7), определяющую условие завершение программы mult. Отметим, что формула, генерируемая для доказательства завершения императивной программы Mul сравнима по сложности.

Две группы формул (6, 8) и (10, 11) структурно подобны: вхождениям постусловия Q_mult в первой группе соответствует вхождения inv во второй группе. Вторая группа формул несколько сложнее первой – в ней используется пять переменных против четырех в первой группе. При этом обе группы автоматически доказываются SMT-решателями CVC3 и Alt-Ergo.

Для доказательства корректности программы Mul дополнительно требуется построить инвариант (a0 – a)*b = c. Это нетривиально, поскольку необходимо догадаться инструментировать программу оператором a := a0. Построение инварианта намного сложнее написания формул предусловия P_mult и постусловия Q_mult.

Доказательство истинности формул корректности

Алгоритм, описанный выше, позволяет автоматически генерировать формулы тотальной корректности предикатной программы. Для каждого определения предиката строиться теория, содержащая в себе определения

формул корректности и типов. В дальнейшем необходимо доказать истинность этих формул.

Все формулы проходят проверку на SMT-решателе CVC3 [16]. Формулы транслируется во внутреннее представление решателя. Решатель осуществляет проверку на истинность, и в зависимости от результата выставляет формуле статус valid, invalid или unknown.

Для доказательства истинности формул, которые решатель не смог проверить, используется система PVS. Теории, построенные генератором, транслируются на язык спецификаций системы. После чего пользователю необходимо в интерактивном режиме, построить доказательство на языке команд системы PVS.

Опыт применения системы верификации

В рамках учебного курса “Формальные методы в описании языков и систем программирования” магистрантам Факультета Информационных Технологий Новосибирского Государственного Университета было предложено задание по написанию формальной спецификации содержательно сформулированной задачи. Каждый магистрант выбрал одну из 40 предложенных задач. В качестве дополнительного задания для желающих (10 магистрантов) предложено написать предикатную программу и провести доказательство ее корректности в системе автоматического доказательства PVS.

Построение формул корректности для 10 предикатных программ проводилось применением системы верификации, описанной в настоящей работе. Это первый опыт производственной эксплуатации данной системы верификации.

Суммарно по всем задачам было сгенерировано 363 формулы. Из них 260 – это условия тотальной корректности. Остальные 103 – это условия совместимости типов.

Программы небольшие. Число операторов не более 13. Однако, количество формул корректности и семантики весьма значительно. В связи с этим от работы решателя CVC3 требуется показать приемлемый результат. Условия совместимости типов желательно проверить полностью автоматически.

Решатель смог проверить 72% формул семантической корректности и 45% формул тотальной корректности.

Заключение

В работе описан метод, позволяющий доказывать тотальную корректность предикатных программ. На основании данного метода была разработана система дедуктивной верификации предикатных программ, которая автоматически генерирует формулы корректности для программ с исходным кодом на языке P. Проведены результаты апробация разработанной системы в рамках курса “Формальные методы …” в 2013-2015гг.

Разработанный метод дедуктивной верификации предикатных программ имеет существенные преимущества по сравнению с классическим методом Хоара. Верификация предикатной программы ориентировочно в 2-3 раза превосходит верификацию аналогичной императивной программы при сравнении по трудоемкости и времени верификации. Это определяет целесообразность разработки и верификации сложных и критических фрагментов кода программной комплекса в системе предикатного программирования с получением итогового кода таких фрагментов на языке C++.

Дальнейшие планы. Необходимо разработать метод доказательства корректности предикатов с переменными предикатного типа в качестве параметров; разработать правила для доказательства корректности гиперфункций. Необходимо снабдить генерируемую теорию на PVS подробными комментариями.

Список литературы

1. Карнаухов Н.С., Першин Д.Ю., Шелехов В.И. Язык предикатного программирования Р // Препринт №153. – Новосибирск: ИСИ СО РАН, 2010. – 42с., http://persons.iis.nsk.su/files/persons/pages/plang12.pdf
2. Шелехов В.И. Методы доказательства корректности программ с хорошей логикой // Межд. конф. "Современные проблемы математики, информатики и биоинформатики", посвященная 100-летию со дня рождения А.А. Ляпунова. — 2011. — 17с., http://conf.nsc.ru/files/conferences/Lyap-100/fulltext/74974/75473/Shelekhov_prlogic.pdf
3. http://www.iis.nsk.su/persons/vshel/files/rules.zip
4. Bertrand Meyer. Towards a Calculus of Object Programs // ETH Zurich, ITMO & Eiffel Software Technical Report, July 2011
5. Manna, Pnueli, Axiomatic Approach to Total Correctness of Programs // Acta Informatica, September 1974, Volume 3, Issue 3, pp 243-263
6. Edsger W. Dijkstra. Guarded commands, nondeterminancy and formal derivation of a programms // Communications of the ACM, Volume 18 Issue 8, Aug. 1975, Pages 453-457
7. Reynolds J.C. Separation Logic: A Logic for Shared Mutable Data Structures // LICS '02 Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science, Pages 55-74
8. Owre, N. Shankar, J. M. Rushby, D. W. J. Stringer-Calvert. PVS Language Reference
9. Shelekhov V. I. 2011. Verification and Synthesis of Addition Programs under the Rules of Correctness of Statements // Automatic Control and Computer Sciences. Vol. 45, No. 7, 421–427
10. Шелехов В.И. Верификация и синтез эффективных программ стандартных функций в технологии предикатного программирования // Программная инженерия, 2011, № 2. 14-21

11. В.А. Вшивков, Т.В. Маркелова, В.И. Шелехов. Об алгоритмах сортировки в методе частиц в ячейках // Научный вестник НГТУ, Т.4(33), с. 79-94, 2008
12. Шелехов В.И. Разработка и верификация алгоритмов пирамидальной сортировки в технологии предикатного программирования // Новосибирск, 2012. – 30с. – (Препр. / ИСИ СО РАН. N 164)
13. Шелехов В.И. Семантика языка предикатного программирования // ЗОНТ-15. — Новосибирск, 2015. — 13с., http://persons.iis.nsk.su/files/persons/pages/semZont1.pdf
14. Предиктное программирование. Учебное пособие // Под ред. Шелехова В.И. НГУ. Новосибирск, 2009. 111 С
15. Каблуков И.В., Шелехов В.И. Контроль динмаической семантики предикатной программы // Новосибирск, 2012. — 28с. — (Препр. / ИСИ СО РАН; N 162)
16. Clark Barrett, Cesare Tinelli. CVC3. // Proceedings of the 19th International Conference on Computer Aided Verification (CAV '07), volume 4590 of Lecture Notes in Computer Science, pages 298-302. Springer, July 2007. Berlin, Germany

Mikhail Chushkin, graduate student,
A.P. Ershov Institute of Informatics Systems, Novosibirsk, Russia,
E-mail: chushkinm@rambler.ru

System for deductive verification of predicate programs

The method of generating formulas total correctness of recursive programs without loops and pointers are developed. A tool for deductive verification that generates formulas of correctness is implemented. The system of rules to simplify the process of correctness formuals generating is developed. The satisfiability of formulas is checked using SMT-solver CVC3 and proof assist system PVS.

Keywords: *Predicate programming, total correctness of a program, deductive verification, proof assist system PVS, SMT-solver CVC3.*

References

1) Karnauhov N.S., Pershin D.Ju., Shelehov V.I. Jazyk predikatnogo programmirovanija P // Preprint №153. – Novosibirsk: ISI SO RAN, 2010. – 42s.

2) Shelehov V.I. Metody dokazatel'stva korrektnosti programm s horoshej logikoj // Mezhd. konf. "Sovremennye problemy matematiki, informatiki i bioinformatiki", posvjashhennaja 100-letiju so dnja rozhdenija A.A. Ljapunova. — 2011. — 17c.

3) http://www.iis.nsk.su/persons/vshel/files/rules.zip

4) Bertrand Meyer. Towards a Calculus of Object Programs // ETH Zurich, ITMO & Eiffel Software Technical Report, July 2011

5) Manna, Pnueli, Axiomatic Approach to Total Correctness of Programs // Acta Informatica, September 1974, Volume 3, Issue 3, pp 243-263

6) Edsger W. Dijkstra. Guarded commands, nondeterminancy and formal derivation of a programms // Communications of the ACM, Volume 18 Issue 8, Aug. 1975, Pages 453-457

7) Reynolds J.C. Separation Logic: A Logic for Shared Mutable Data Structures // LICS '02 Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science, Pages 55-74

8) Owre, N. Shankar, J. M. Rushby, D. W. J. Stringer-Calvert. PVS Language Reference

9) Shelekhov V. I. 2011. Verification and Synthesis of Addition Programs under the Rules of Correctness of Statements // Automatic Control and Computer Sciences. Vol. 45, No. 7, 421–427

10) Shelehov V.I. Verifikacija i sintez jeffektivnyh programm standartnyh funkcij v tehnologii predikatnogo programmirovanija // Programmnaja inzhenerija, 2011, № 2. 14-21

11) V.A. Vshivkov, T.V. Markelova, V.I. Shelehov. Ob algoritmah sortirovki v metode chastic v jachejkah // Nauchnyj vestnik NGTU, T.4(33), s. 79-94, 2008

12) Shelehov V.I. Razrabotka i verifikacija algoritmov piramidal'noj sortirovki v tehnologii predikatnogo programmirovanija // Novosibirsk, 2012. – 30s. – (Prepr. / ISI SO RAN. N 164)

13) Shelehov V.I. Semantika jazyka predikatnogo programmirovanija // ZONT-15. — Novosibirsk, 2015. — 13s.

14) Prediktnoe programmirovanie. Uchebnoe posobie // Pod red. Shelehova V.I. NGU. Novosibirsk, 2009. 111 C

15) Kablukov I.V., Shelehov V.I. Kontrol' dinmaicheskoj semantiki predikatnoj programmy // Novosibirsk, 2012. — 28s. — (Prepr. / ISI SO RAN; N 162)

16) Clark Barrett, Cesare Tinelli. CVC3. // Proceedings of the 19th International Conference on Computer Aided Verification (CAV '07), volume

4590 of Lecture Notes in Computer Science, pages 298-302. Springer, July 2007. Berlin, Germany