Аксиоматический подход к формальной верификации рекурсивных программ на функционально-потоковом языке параллельного программирования

М.С. Кропачева

Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Сибирский федеральный университет»

Работа посвящена применению дедуктивного анализа для автоматизированного доказательства корректности функционально-потоковых параллельных программ на языке Пифагор. Строится исчисление Хоара для этого языка. Корректность рекурсивных функций доказывается с помощью метода индукции.

1. Введение

В настоящее время происходит повсеместный переход на параллельное программирование, что обусловлено широким применением многоядерных процессоров, кластерных систем и графических ускорителей. Использование традиционных подходов, базирующихся на императивных моделях, для разработки параллельного программного обеспечения достаточно часто ведёт к появлению ошибок. В связи с тем, что в большинстве случаев отказы систем связаны с нештатным функционированием программного обеспечения, актуальны вопросы его надёжности. Для повышения надёжности программного обеспечения всё чаще используется формальная верификация.

Под формальной верификацией понимается доказательство корректности программы, которое заключается в установлении соответствия между программой и её спецификацией, описывающей цель разработки [1]. При этом, соответствие программы её спецификации устанавливается посредством строгого математического доказательства. Главным преимуществом формальной верификации является возможность доказать отсутствие ошибок в программе, тогда как тестирование позволяет лишь выявлять отдельные ошибки.

Один из методов формальной верификации был предложен Хоаром [2]. В его основе лежит аксиоматический подход на основе исчисления Хоара — расширения какой-либо формальной теории $\mathfrak{I}$ введением в неё формул специального вида, называемых тройками Хоара. Тройка Хоара — это аннотированная программа, то есть программа, к которой приписаны две формулы теории $\mathfrak{I}$, описывающие ограничения на входные переменные и требования к результату выполнения программы. Эти формулы называются предусловие и постусловие соответственно. Тройки Хоара обычно записывается в виде $\{\varphi\}$ `Prog` $\{\psi\}$, где `Prog` — программа, а φ и ψ — предусловие и постусловия для `Prog`. Также для расширения теории $\mathfrak{I}$ вводится набор аксиом для операторов языка и правила вывода, с помощью которых из аксиом можно выводить утверждения о свойствах программ, в том числе о свойствах корректности. При этом, расширение теории $\mathfrak{I}$ строится таким образом, чтобы корректность программы соответствовала истинности тройки Хоара для этой программы. Основная идея данного подхода заключается в том, чтобы на базе аксиом, с помощью последовательных применений правил вывода, преобразовать тройку Хоара в формулу теории $\mathfrak{I}$, и доказать истинность формулы в этой теории.

Основным преимуществом аксиоматический подхода на основе исчисления Хоара, в отличие от менее формализованных методов дедуктивного анализа, является возможность его частичной автоматизации. Кроме того, данный подход является универсальным методом и применим для произвольных языков программирования, однако аксиомы и правила

вывода не универсальны, так как строятся на основе семантики языка программирования. Следовательно, различна и сложность доказательства корректности программ в разных аксиоматических системах.

В настоящее время достигнуты определённые успехи в практическом применении данного подхода для императивных языков программирования [3], однако, в целом, проблема не решена. Стоит отметить, что для параллельных императивных программ сложность формальной верификации сильно возрастает. Главной проблемой являются конфликты из-за ресурсов. Например, неправильное совместное использовании общей памяти или взаимные блокировки процессов в случае работы с распределённой памятью. Альтернативным вариантом императивного подхода является функционально-потоковая параллельная (ФПП) парадигма и предложенный для её поддержки язык программирования Пифагор [4]. Этот язык позволяет избежать конфликтов из-за ресурсов. Каждая программа — это функция, поэтому в языке отсутствуют переменные и циклы, а операции выполняются по готовности данных. В результате, сложность формальной верификации ФПП программ сравнима со сложностью верификации последовательных программ. Другая важная особенность языка — возможность достичь максимального параллелизма программы за счёт того, что параллелизм реализуется на уровне операций. Поэтому после доказательства корректности такая программа может быть перенесена на конкретную архитектуру с конечными ресурсами, при необходимости, с ограничением её параллелизма.

На данный момент, вопросы формальной верификации для языка программирования Пифагор проработаны недостаточно, поэтому актуальным является развитие методов анализа и формальной верификации ФПП программ.

2. Описание аксиоматической теории

Для выполнения формальной верификации на основе исчисления Хоара, для ФПП языка Пифагор разработана аксиоматическая теория, в рамках которой проводятся формальные доказательства корректности программ.

Основными объектами аксиоматической теории являются тройки Хоара.

Для описания предусловий и постусловий используется язык логической спецификации на базе логики предикатов первого порядка, которая достаточна для описания большинства требований к программе. Для описания выражений языка логической спецификации используется алфавит исчисления предикатов с функциональными и предикатными символами, которые соответствуют функциям языка программирования. Переменные языка логической спецификации могут быть различных типов:

$$T \quad = \quad \{signal, int, float, char, bool, func, error, datalist, delaylist, parlist, asynclist, \text{user\_type}\},$$

где user_type - соответствует множеству пользовательских типов. Фигурные скобки в выражениях языка логической спецификации используются для обозначения множества.

В исчислении предикатов различают функции, которые формируют термы, и предикаты, которые формируют формулы. В данном случае можно не выделять предикаты в отдельную группу, а считать их подмножеством функций с областью значений *bool*. Введем следующие функциональные и предикатные символы, и кванторы:

1. Арифметические операции $(+, -, *, /)$.
2. Знаки сравнения $(=, \neq, >, <, \geqslant, \leqslant)$.
3. Логические операции и кванторы $(\vee, \wedge, \neg, \Rightarrow, \Leftrightarrow, \forall, \exists)$.
4. Функция длины списка (len), функция выбора элемента из списка $(select)$, функция принадлежности к типу $(\in)$.

Функции len, $select$, $\in$ эквивалентны соответствующим функциям языка Пифагор. Схемы для перечисленных функций будут совпадать со схемами функций из исчисления предикатов и арифметики.

Определим понятие элементарного терма с помощью индукции:

1. Каждая предметная переменная является элементарным термом.
2. Если $t_1, t_2, \ldots, t_n$ — элементарные термы, f — n-местная функция, то выражение $f(t_1, t_2, \ldots, t_n)$ является элементарным термом, при этом все константы считаются нуль-местными функциями.
3. Других элементарных термов нет.

Элементарная формула языка — это элементарный терм типа *bool*. Тогда произвольную формулу (выражение) можно определить по индукции:

1. Каждая элементарная формула является формулой.
2. Если A — формула, то $\forall x A(x)$ и $\exists x A(x)$ тоже является формулой.
3. Других формул нет.

В традиционном виде тройка Хоара записывается в виде $\{\varphi\}$ `Prog` $\{\psi\}$, где `Prog` — программа, а φ и ψ — предусловия и постусловия. Для того, чтобы фигурные скобки тройки не совпадали с фигурными скобками языка программирования или языка логической спецификации, для тройки Хоара вводится следующее обозначение:

$$\boxed{\varphi(x)}\ \texttt{Prog(x)}\rightarrow r\ \boxed{\psi(r)}\ ,$$

где `Prog(x)` — функция с входным аргументом x, r — результат работы программы, $\varphi(x)$ — предусловие для `Prog`, зависящее от аргумента x, $\psi(r)$ — постусловия для `Prog`, зависящее от результата выполнения функции.

Например, рассмотрим следующую функцию на языке Пифагор:

```
Fun << funcdef arg {
   arg:F >>  return
}
```

Пусть P и Q — предусловия и постусловия для этой функции. Тогда тройка Хоара будет иметь вид:

$$\boxed{P(arg)}\ \texttt{arg:F}\rightarrow r\ \boxed{Q(r)}\ .$$

В связи с тем, что во многих ситуациях программу на языке Пифагор удобно отображать в виде информационного графа, возможна непосредственная привязка предусловия и постусловия к дугам этого графа. Пример подобной привязки представлен на рисунке 2.

Аксиомами рассматриваемой аксиоматической теории языка программирования Пифагор служат тройки Хоара для встроенных функций. Возможны различные алгоритмы вычисления встроенной функции в зависимости от некоторых условий на аргументы [5]. Каждому такому алгоритму соответствует отдельная аксиома. Использование нескольких аксиом для одной встроенной функции, по сравнению с использованием одной сложной аксиомы, упрощает вид итоговых формул, являющихся условиями корректности программы.

Ниже приведены аксиомы для функции создания списка из одинаковых элементов, которая в языке Пифагор задаётся следующим образом: `p:dup`. Предполагается, что `p` — это список, состоящий из двух элементов, где первый элемент — это произвольный аргумент, а второй — целое число, определяющее количество повторений первого элемента. При использовании данной функции могут возникать различные ошибки. Тогда аксиомы для функции дублирования будут следующими:

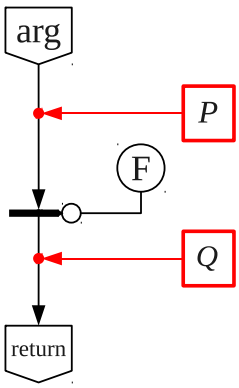

Рис. 1. Тройка Хоара для программы `arg:F`

$$\boxed{\begin{array}{l}(p \in datalist)\wedge\\(len(p)=2)\wedge\\(select(p,2)\in int)\end{array}}\ \texttt{p:dup}\ \to r\ \boxed{\begin{array}{l}(r \in datalist)\wedge\\(len(r)=select(p,2))\wedge\\\forall k\,(k\in\{1,2,\ldots,select(p,2)\}\wedge\\\qquad select(r,k)=select(p,1))\end{array}},$$

$$\boxed{\begin{array}{l}\neg(p \in datalist)\vee\\\neg(len(p)=2)\vee\\\neg(select(p,2)\in int)\end{array}}\ \texttt{p:dup}\ \to r\ \boxed{r=\texttt{BASEFUNCERROR}}.$$

Правила вывода описывают схему преобразования композиций операторов и позволяют выводить теоремы из имеющихся аксиом и уже доказанных теорем. Для того, чтобы процесс преобразования троек Хоара в формулу исчисления предикатов являлся однозначным, используются правила прямого прослеживания, ориентированные на вывод от аргумента функции к её результату. В случае ФПП языка Пифагор таких функций может быть несколько, тогда из них выбирается произвольная. При этом, произвольный порядок применения правила прямого прослеживания для параллельно выполняющихся функций не окажет влияние на результирующую формулу с точностью до порядка операндов конъюнкции.

В общем случае, «правило прямого прослеживания» для некоторой функции с кодом «$\texttt{x:F}_1\texttt{:F}$» имеет следующий вид:

$$\boxed{P_1(x_1)}\ \texttt{x}_1\texttt{:F}\to r\ \boxed{Q(r)},\ A_1,\ A_2\quad\vdash\quad\boxed{P(x)}\ \texttt{x:F}_1\texttt{:F}\to r\ \boxed{Q(r)},\qquad(1)$$

$$A_1:=\boxed{\varphi(x)}\ \texttt{x:F}_1\to x_1\ \boxed{\psi(x_1)},\ A_2:=P(x)\Rightarrow\varphi(x),\ P_1(x_1):=P(x)\wedge\varphi(x)\wedge\psi(x_1),$$

где `x` — входной аргумент, $\texttt{F}_1$ — функция, применяемая непосредственно к входному аргументу, `F` — остальная часть программы, которая может содержать другие функции, применяемые непосредственно к аргументу `x`, $\vdash$ — символ выводимости, который свидетельствует о том, что при истинности выражений в левой части, истинны и выражения в правой части. Согласно этому правилу, на основе аксиомы A_1 для функции $\texttt{F}_1$ и при условии A_2, тройка Хоара $\boxed{P(x)}\ \texttt{x:F}_1\texttt{:F}\to r\ \boxed{Q(r)}$ преобразуется в тройку с более «короткой» программой $\boxed{P_1(x_1)}\ \texttt{x}_1\texttt{:F}\to r\ \boxed{Q(r)}$, при этом предусловие $P(x)$ изменяется на $P_1(x_1)$, а исходный аргумент x заменяется на x_1 — результат применения функции $\texttt{F}_1$ к x.

При последовательном применении правила прямого прослеживания происходит «сокращение» или «свёртка» программы, и по завершению анализа всех операторов получаем

тройку Хоара с «пустой» программой: $\boxed{\text{P}}\ \boxed{\text{Q}}$. Это означает, что предусловие и постусловие приписаны к одной дуге информационного графа. Для завершения преобразования тройки Хоара к формуле исчисления предикатов вводится следующее правило:

$$P \Rightarrow Q \vdash \boxed{\text{P}}\ \boxed{\text{Q}}. \tag{2}$$

Таким образом, с помощью преобразований произвольных троек Хоара на основе правил вывода можно получить формулу исчисления предикатов, истинность которой доказывается в рамках исчисления предикатов. Если эта формула истинна, то истинна и исходная тройка Хоара, откуда следует корректность программы.

В зависимости от входных данных программа может иметь различные пути выполнения. Аксиомы встроенных функций полностью определяют дерево $\mathfrak{T}_0$ всех путей выполнения программы. Количество различных вариантов выполнения программы для каждой функции равно количеству аксиом этой функции. Если на входные данные наложены ограничения (предусловие программы), то часть ветвей в дереве становятся недостижимыми. Эти ветви соответствуют тем аксиомам, предусловие которых не следует из предусловия программы. Откидывая недостижимые пути, получаем новое дерево $\mathfrak{T}_1$, которое является поддеревом $\mathfrak{T}_0$. Каждый путь дерева $\mathfrak{T}_1$ можно рассмотреть отдельно и преобразовать в формулу исчисление предикатов по правилам вывода. В результате таких преобразований получается k формул, где k соответствует числу листьев в дереве $\mathfrak{T}_1$. Если каждая из полученных формул истинна, то программа корректна. В противном случае программа некорректна, а ошибки присутствуют в тех ветвях, которым соответствуют не тождественно истинные формулы. Таким образом, доказательство корректности программы сводится к доказательству истинности нескольких формул.

3. Анализ корректности рекурсий

Основная проблема верификации программ связана с циклическими конструкциями, которые могут, в случае неправильной программы, привести к зацикливанию, при этом информационный граф программы становится бесконечным. В языке Пифагор отсутствуют операторы цикла, и все повторяющиеся конструкции реализуются через рекурсии. Для того, чтобы программа была корректной, рекурсия должна, во-первых, завершаться, а, во-вторых, возвращать правильный результат.

Проанализируем особенности рекурсивной функции. Если в программе присутствует рекурсия, то один и тот же код вызывается несколько раз, а различия будут касаться только аргументов. Тогда необходимым условием завершения рекурсии является то, что аргумент пробегает неповторяющуюся последовательность значений.

Большинство рекурсивных функций достаточно просто представить в следующем виде: в корректной рекурсивной функции обязательно должна присутствовать «точка ветвления», в которой происходит выбор между возможными путями выполнения программы. Одна часть путей приводит к завершению работы и возвращению результата вычислений, другая — к рекурсивному вызову функции. По какому пути пойдут дальнейшие вычисления, определяется значением некоторой функции от входных аргументов. Эту функцию можно рассматривать как некий аналог оператора `if-else` в том смысле, что значения выражения, определяющего переход к следующей итерации, и выражения завершения итерации рекурсии, являются взаимно исключающими, то есть $\neg(\textit{условие перехода} = \textit{true}) \Leftrightarrow (\textit{условие завершения} = \textit{true})$.

Для доказательства корректности рекурсивной программы необходимо выделить «точку ветвления», ветвь, соответствующую рекурсивному пути и ветвь, приводящую к завершению программы, а также условия, которые определяют какой путь будет выбран в «точке ветвления». Назовём эти условия «условием рекурсивного вызова» и «условием завершения функции». Также введём два термина: «текущий аргумент» — входной аргумент функции

и «аргумент рекурсивного вызова» — аргумент для рекурсивного вызова функции.

Правильность рекурсии можно доказать по индукции (доказательство завершения рекурсии проводится аналогично). Введём понятие ограничивающей функции. *Ограничивающая функция* — это ограниченная снизу функция, переводящая аргумент рекурсивной функции во множество натуральных чисел $\mathbb{N}$, при этом аргументы, для которых выполняется условие завершения, отображаются в единицу.

Рассмотрим схему доказательства корректности рекурсивной программы Rec, которое проводится индукцией по значениям ограничивающей функции f.

База индукции: проверяем корректность программы для аргумента $x = p_0$, такого что $f(p_0) = 1$.

Шаг индукции: предположим, что программа корректна для всех аргументов, для которых значение ограничивающей функции меньше N. Числу N соответствует параметр p_N, такой что $f(p_N) = N$. Тогда, для того, чтобы рекурсивная функция была корректна, достаточно, чтобы одновременно выполнялись следующие условия:

1. При выполнении функции $Rec(p_N)$ рекурсивно могут быть вызваны только $Rec(p_i), i = \overline{1,n}$, для которых значения ограничивающей функции $f(p_i)$ меньше N.

2. Параметры $p_i, i = \overline{1,n}$ являются допустимыми аргументами функции Rec. Данное условие не зависит от ограничивающей функции, поэтому его можно использовать, как первоначальную проверку корректности функции.

3. Функция $Rec(p_N)$ вернёт верный результат, если все рекурсивные вызовы заменить на результат выполнения $Rec(p_i), i = \overline{1,n}$, то есть верно, что из корректности функции Rec для аргументов p_i с меньшим значением ограничивающей функции следует корректности функции Rec для текущего значения аргумента p_N.

Описанный алгоритм является достаточным условием корректности рекурсии. В случае, если не удаётся доказать корректность, то это может свидетельствовать о том, что программа некорректна, или неправильно выбрана ограничивающая функция. В последнем случае придется искать другую ограничивающую функцию.

4. Пример доказательства корректности программы с рекурсией

Докажем корректность рекурсивной функции `fact`, вычисляющей факториал x. Код программы на языке Пифагор приведён на рисунке 2.

```
fact << funcdef x {
    fl << ((x,1):[<=,>]):?;
    act << (1,
          { (x, (x,1):-:fact ):*  } );
    return << act:fl:.;
}
```

Рис. 2. Код функции `fact`, вычисляющей факториал числа x

Пользователь задаёт для программы следующую тройку Хоара (дописывает предусловие и постусловие):

$$\boxed{\begin{array}{l}(x \in int) \wedge (x \geqslant 0) \wedge \\ \prod\limits_{i=1}^{x} i \leqslant \texttt{INT\_MAX}\end{array}} \ \texttt{x:fact} \rightarrow r \ \boxed{\begin{array}{l}((r = \prod\limits_{i=1}^{x} i) \wedge (x > 0)) \vee \\ ((r = 1) \wedge (x = 0))\end{array}}, \tag{3}$$

где `INT_MAX` — максимальное целое, которое допускает тип int. Данная тройка Хоара содержит в себе следующую спецификацию программы: если входное значение x — целое число

больше нуля и произведение чисел от 1 до x не превышает максимального целого, которое допускает тип int, то после выполнения функция `fact` возвращает результат перемножения чисел от 1 до x, если $x > 0$ или 1, если $x = 0$.

В виде информационного графа данная тройка Хоара приведена на рисунке 3.В серых кружках указаны идентификаторы дуг, все идентификаторы должны быть различны.

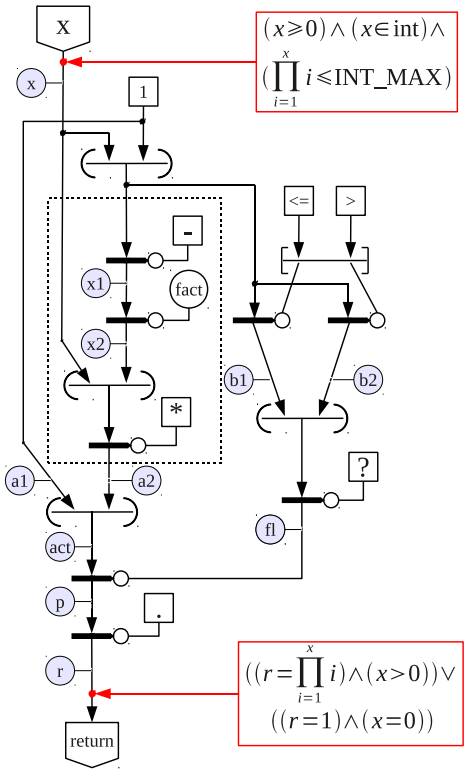

Рис. 3. Информационный граф тройки Хоара для функции `fact`

Рассмотрим процесс вычисления функции `fact`. При $x = 0$ или 1 должно быть возвращено значение 1, при $x > 1$ должен быть совершен рекурсивный вызов `fact` с аргументом $x - 1$. Проверка истинности одного из условий $x \leqslant 1$ или $x > 1$ выполняется в правой ветви, содержащей функцию «?». Параллельно, в левой ветви формируется список p из двух элементов: «1» и «задержанный список» (на рисунке он выделен пунктиром), содержащий подграф с рекурсивным вызовом функции `fact`. Операторы задержанного списка выполняются только после снятия задержки, даже если все аргументы готовы. Выбор одного из элементов в списке p осуществляется в зависимости от того, какое из условий $x \leqslant 1$ или $x > 1$ верно. Во втором случае выбирается задержанный список, с него снимается задержка функцией «.», и происходит рекурсивный вызов. В случае $x \leqslant 1$ из списка p выбирается константа «1», а вычисление функций задержанного списка вообще не происходит.

Для доказательства корректности программы необходимо доказать истинность тройки (3). Преобразуем тройку в формулу, последовательным применением «правила прямого прослеживания» (1) на основе аксиом для встроенных функций. Выполнение функции `fact` начинается с того, что входной аргумент `x` и константа «`1`» формируют список `(x,1)`. К этому списку применяются функции «`<=`» и «`>`». Это встроенные функции, их аксиомы одинаковые и отличаются лишь знаком операции. Для функции «`<=`» аксиомы имеют вид:

$$\boxed{(p_1, p_2 \in \{int, float\}) \vee (p_1, p_2 \in char) \vee (p_1, p_2 \in bool)} \ \texttt{(p1,p2):<=} \ \rightarrow r_1 \ \boxed{(r_1 \in bool) \wedge (r_1 = (p_1 \leqslant p_2))}, \qquad (4)$$

$$\boxed{\begin{array}{l}\neg((p_1,p_2\in\{int,float\})\vee\\(p_1,p_2\in char)\vee(p_1,p_2\in bool))\end{array}}\ \texttt{(p1,p2):<=}\ \to r_1\ \boxed{\begin{array}{l}(r_1\in error)\wedge\\(r_1=\texttt{BASEFUNCERROR})\end{array}}. \qquad (5)$$

Перед применением правила прямого прослеживания необходимо выбрать те аксиомы, у которых предусловие может следовать из предусловия тройки Хоара (3). Все остальные аксиомы отбрасываются. Формируем две формулы, описывающие «условие невыводимости» предусловия аксиом (4) и (5) из предусловия тройки (3), и проверяем их истинность; аксиомы, которым соответствуют истинные формулы — отбрасываем. После согласования обозначений аргументов аксиом (4) и (5) с рассматриваемой тройкой (3) и введения идентификатора b_1 для результата выполнения функции «<=», получаем что $p_1 := x$, а $p_2 := 1$, тогда условия невыводимости опишутся следующими формулами:

$$\neg\left(\left((x\in int)\wedge(x\geqslant 0)\wedge(\prod_{i=1}^{x} i\leqslant \texttt{INT\_MAX})\right)\to \right.$$
$$\left.\to\big((x,1\in\{int,float\})\vee(x,1\in char)\vee(x,1\in bool)\big)\right)$$

и

$$\neg\left(\left((x\in int)\wedge(x\geqslant 0)\wedge(\prod_{i=1}^{x} i\leqslant \texttt{INT\_MAX})\right)\to \right.$$
$$\left.\to\neg\big((x,1\in\{int,float\})\vee(x,1\in char)\vee(x,1\in bool)\big)\right).$$

Первая формула является тождественно ложной, вторая — тождественно истинной, поэтому вторая аксиома отбрасывается, так как выполнение программы не сможет пойти по этому пути.

В результате применения «правила прямого прослеживания» к тройке (3) на основе аксиомы (4), получаем следующую тройку Хоара:

$$\boxed{\begin{array}{l}(x\in int)\wedge(x\geqslant 0)\wedge(\prod_{i=1}^{x} i\leqslant \texttt{INT\_MAX})\wedge\\(b_1\in bool)\wedge(b_1=(x\leqslant 1))\end{array}}\ \texttt{fact1}\ \to r\ \boxed{\begin{array}{l}((r=\prod_{i=1}^{x} i)\wedge(x>0))\vee\\((r=1)\wedge(x=0))\end{array}}, \qquad (6)$$

где `fact1` соответствует код:

```
[( 1,
   { (x, (x,1):-:fact ):*  }
):(b1,(x,1):>):?]:.
```

Далее последовательно применяем «правило прямого прослеживания» на основе аксиом для следующих встроенных функций: «>», «?», функции выбора элемента из списка и «.». В результате преобразований получаются следующие тройки:

$$\boxed{\begin{array}{l}(x\in int)\wedge(x\geqslant 0)\wedge(\prod_{i=1}^{x} i\leqslant \texttt{INT\_MAX})\wedge\\((x\leqslant 1)=true)\wedge((x>1)=false)\wedge(r=1)\end{array}}\ \boxed{\begin{array}{l}((r=\prod_{i=1}^{x} i)\wedge(x>0))\vee\\((r=1)\wedge(x=0))\end{array}}, \qquad (7)$$

$$\boxed{\begin{array}{l}(x\in int)\wedge(x\geqslant 0)\wedge\\(\prod_{i=1}^{x} i\leqslant \texttt{INT\_MAX})\wedge\\((x\leqslant 1)=false)\wedge\\((x>1)=true)\end{array}}\ \texttt{(x, (x,1):-:fact):*}\ \to r\ \boxed{\begin{array}{l}((r=\prod_{i=1}^{x} i)\wedge(x>0))\vee\\((r=1)\wedge(x=0))\end{array}}. \qquad (8)$$

Тройка (7) является тройкой с «пустой» программой, поэтому её можно преобразовать в формулу по правилу (2):

$$\left((x \in int) \wedge (x \geqslant 0) \wedge (\prod_{i=1}^{x} i \leqslant \texttt{INT\_MAX}) \wedge ((x \leqslant 1) = true) \wedge ((x > 1) = false) \wedge (r = 1)\right) \rightarrow$$
$$\rightarrow \left(((r = \prod_{i=1}^{x} i) \wedge (x > 0)) \vee ((r = 1) \wedge (x = 0))\right).$$

Очевидно, что данная формула истинна, так как из $(x \geqslant 0)$ и $((x \leqslant 1) = true)$, следует, что $x = 0$ или $x = 1$, а $r = 1$.

Теперь рассмотрим тройку (8). Функция «-» — ближайшая к входному аргументу, поэтому выполняется первой. При применении «правила прямого прослеживания» на основе аксиом функции «-» к тройке (8) получаем тройку:

$$\boxed{\begin{array}{l}(x \in int) \wedge (x \geqslant 0) \wedge (\prod_{i=1}^{x} i \leqslant \texttt{INT\_MAX}) \wedge \\ ((x \leqslant 1) = false) \wedge ((x > 1) = true) \wedge \\ (x_1 \in int) \wedge (x_1 = x - 1)\end{array}} \text{ (x, x1:fact)}:* \rightarrow r \boxed{\begin{array}{l}((r = \prod_{i=1}^{x} i) \wedge (x > 0)) \vee \\ ((r = 1) \wedge (x = 0))\end{array}}. \qquad (9)$$

Далее должен выполняться рекурсивный вызов функции `fact`. Поэтому использовать «правило прямого прослеживания» напрямую нельзя, так как корректность функция `fact` не доказана.

Отметим, что в рассматриваемой функции «текущий аргумент» — это входной аргумент «`x`», «аргумент рекурсивного вызова» — аргумент для рекурсивного вызова функции «`(x,1):-`» (обозначен x_1); ветвь a_1 информационного графа не содержит рекурсию, а ветвь a_2 — содержит, тогда «условие завершения функции» и «условие рекурсивного вызова» определяют выражения дуг $b_1 := (x \leqslant 1)$ и $b_2 := (x > 1)$ (обозначения приведены на рис. 3).

В начале, необходимо показать, что при рекурсивном вызове функции `fact` ей передаётся корректный аргумент. Это эквивалентно тому, что из предусловия тройки (9) выводимо предусловие функции `fact` (3), в котором «текущий аргумент» x заменяется на выражение, описывающее «аргумент рекурсивного вызова» x_1, тогда «условие невыводимости» примет вид:

$$\neg\left(\left((x \in int) \wedge (x > 1) \wedge (\prod_{i=1}^{x} i \leqslant \texttt{INT\_MAX}) \wedge (x_1 \in int) \wedge (x_1 = x - 1)\right) \rightarrow \right.$$
$$\left. \rightarrow \left((x_1 \in int) \wedge (x_1 \geqslant 0) \wedge (\prod_{i=1}^{x_1} i \leqslant \texttt{INT\_MAX})\right)\right).$$

Исходя из того что $(x > 1)$, а $(x_1 = x - 1)$, то $(x_1 > 0)$. Если $(\prod_{i=1}^{x} i \leqslant \texttt{INT\_MAX})$, то $(\prod_{i=1}^{x_1} i = \prod_{i=1}^{(x-1)} i < \texttt{INT\_MAX})$. Значит формула тождественно ложна, и «аргумент рекурсивного вызова» всегда удовлетворяет предусловию функции `fact`. В противном случае программа сразу бы считалась некорректной.

Зададим ограничивающую функцию для `fact`:

$$f(x) = \begin{cases} 1, & x = 0; \\ x, & x > 0. \end{cases}$$

Докажем корректность функции `fact` индукцией по значению ограничивающей функции.

База индукции. Если аргумент x удовлетворяет предусловию (3) $(x \in int) \wedge (x \geqslant 0) \wedge \prod_{i=1}^{x} i \leqslant \texttt{INT\_MAX}$ и «условию завершения программы» $x \leqslant 1$, то $x = 0$ или $x = 1$. Для этих значений $f(x) = f(0) = f(1) = 1$. Очевидно, что для указанных значений аргументов функция `fact` завершается без рекурсивного вызова, а результат в выполнения функции равен единице ($r = 1$) и удовлетворяет постусловию (3). Формальная проверка этого утверждения реализуется с помощью доказательства корректности тройки Хоара (7), что было сделано ранее.

Шаг индукции. Возьмём аргумент x, удовлетворяющий (3), для которого выполнено «условие рекурсивного вызова» $x > 0$ и значение ограничивающей функции равно N: $f(x) = N$. Предположим, что функция корректна для всех значений аргумента, для которых значение ограничивающей функции меньше N. Покажем, что значение ограничивающей функции «аргумента рекурсивного вызова» всегда меньше N:

$$f(x_1) = f(x-1) = f(N-1) = N - 1 < N.$$

Тогда по индуктивному предположению будет верна тройка Хоара для функция `fact`, применённая к «аргументу рекурсивного вызова»:

$$\boxed{\begin{array}{l}(x_1 = x - 1) \wedge (x_1 \in int) \wedge \\ (x_1 \geqslant 0) \wedge (\prod_{i=1}^{x_1} i \leqslant \texttt{INT\_MAX}\ )\end{array}}\ \texttt{x1:fact}\ \rightarrow x_2\ \boxed{\begin{array}{l}((x_2 = \prod_{i=1}^{x_1} i) \wedge (x_1 > 0)) \vee \\ ((x_2 = 1) \wedge (x_1 = 0))\end{array}}. \tag{10}$$

Полученную тройку можно использовать в качестве теоремы и доказывать корректность программы `fact` с помощью «правила прямого прослеживания», считая функцию нерекурсивной.

Тогда при применении «правила прямого прослеживания» на основе теоремы (10) к тройке (9) получаем:

$$\boxed{\begin{array}{l}(x \in int) \wedge (x \geqslant 0) \wedge (\prod_{i=1}^{x} i \leqslant \texttt{INT\_MAX}) \wedge \\ ((x \leqslant 1) = false) \wedge ((x > 1) = true) \wedge \\ (x_1 \in int) \wedge (x_1 = x - 1)\end{array}}\ (\text{x, x2 }){:}* \rightarrow r\ \boxed{\begin{array}{l}((r = \prod_{i=1}^{x} i) \wedge (x > 0)) \vee \\ ((r = 1) \wedge (x = 0))\end{array}}. \tag{11}$$

В коде тройки (11) осталась только функция умножения «`*`». Применяя «правило прямого прослеживания» на основе аксиом для этой функции получаем тройку с «пустой» программой:

$$\boxed{\begin{array}{l}(x \in int) \wedge (x > 1) \wedge (\prod_{i=1}^{x} i \leqslant \texttt{INT\_MAX}) \wedge \\ (x_2 = \prod_{i=1}^{(x-1)} i) \wedge (x, x_2 \in int) \wedge (x * x_2 \in int) \wedge (r = x * x_2)\end{array}}\ \boxed{\begin{array}{l}((r = \prod_{i=1}^{x} i) \wedge (x > 0)) \vee \\ ((r = 1) \wedge (x = 0))\end{array}}.$$

Далее применяем правило преобразования тройки в формулу (2), и после упрощения получаем:

$$\left((x, x_2 \in int) \wedge (x > 1) \wedge (\prod_{i=1}^{x} i \leqslant \texttt{INT\_MAX}) \wedge (x_2 = \prod_{i=1}^{(x-1)} i) \wedge (r = x * x_2)\right) \rightarrow$$
$$\rightarrow \left(((r = \prod_{i=1}^{x} i) \wedge (x > 0)) \vee ((r = 1) \wedge (x = 0))\right).$$

Полученная формула истинна, а значит истинна исходная тройка Хоара (3), из истинности которой следует корректность программы.

Таким образом, корректность функции `fact` доказана.

5. Заключение

В работе рассмотрены особенности формальной верификация ФПП программ, выстраивается аксиоматическая теория для языка Пифагор, а также показано использование этой теории для доказательства корректности рекурсивных программ. Благодаря тому, что язык программирования Пифагор не ограничивает параллелизм разрабатываемых на нём программ, он может быть использован в качестве инструмента для обобщённой спецификации параллельных программ, обеспечивая их более простую формальную верификацию, тестирование и отладку, а затем — формальное преобразование (автоматическое или ручное) программы в представление, предназначенное для выполнения на конкретной (целевой) архитектуре. В дальнейшем, разработанный подход может быть положен в основу среды инструментальной поддержки доказательства корректности, так как подвергается автоматизации на многих этапах.

Литература

1. Непомнящий В.А., Рякин О.М. Прикладные методы верификации программ. М.: Радио и связь, 1988. 255 с.

2. Hoare C. A. R. An axiomatic basis for computer programming // Communications of the ACM. 1969. Vol. 10, No. 12. P. 576–585.

3. Ануреев И. С., Марьясов И. В., Непомнящий В. А. Верификация С-программ на основе смешанной аксиоматической семантики // Моделирование и анализ информационных систем. 2010. Т. 17, № 3. С. 5–28.

4. Легалов А.И. Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии. 2005. № 1 (10). С. 71–89.

5. Кропачева М.С. Формализация семантики функционально-потокового языка параллельного программирования Пифагор // Материалы XII Всероссийской научно-практической конференции «Проблемы информатизации региона» (ПИР-2011). Красноярск: Сибирский федеральный университет. 2011. С. 144–148.