Модел. и анализ информ. систем. Т. **19**, № **5** (2012) 81–99

УДК 517.51+514.17

Формальная верификация программ, написанных на функционально-потоковом языке параллельного программирования

Кропачева М. С., Легалов А. И. [1]

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

e-mail: ksv@akadem.ru, legalov@mail.ru

получена 23 августа 2012

Ключевые слова: функционально-потоковое параллельное программирование, язык программирования Пифагор, формальная верификация программ

Работа посвящена доказательству корректности параллельных программ на основе аксиоматического подхода. Описана формальная система для функционально-потокового языка параллельного программирования Пифагор, в рамках которой можно проводить доказательства.

1. Введение

На данном этапе развития информационных технологий происходит повсеместный переход на параллельное программирование, что обусловлено широким применением многоядерных процессоров, кластерных систем и графических ускорителей для решения самых разнообразных задач. В основном программы пишутся на императивных языках программирования. Вероятность допустить ошибку в параллельной императивной программе выше, по сравнению с последовательной программой, так как для такой программы появляются свои специфические ошибки.

В связи с тем, что в большинстве случаев отказы систем связаны с нештатным функционированием программного обеспечения, актуальны вопросы его безопасности. 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]. Основным преимуществом формальной верификации является возможность доказать отсутствие ошибок в программе, тогда как тестирование позволяет лишь выявлять

[1]Работа поддержана грантом в рамках ФЦП «Научные и научно-педагогические кадры инновационной России» № 14.A18.21.0396.

ошибки. Помимо этого, формальная верификация предполагает аналитическое исследование свойств программы по её тексту, и цель верификации достигается строгим математическим доказательством соответствия программы её спецификации. В работе [2] предложен 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 записываются в виде $\{\varphi\}$ `Prog` $\{\psi\}$, где `Prog` — программа, а φ и ψ — предусловие и постусловие для `Prog`. В расширенную теорию 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. Тогда корректность программы будет соответствовать истинности тройки Хоара для этой программы. Основная идея подобного подхода заключается в том, чтобы на базе аксиом, с помощью последовательных применений правил вывода, преобразовать тройку Хоара верифицируемой программы в формулу теории $\mathfrak{I}$, и доказать истинность формулы в этой теории.

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 использование памяти при работе с общей памятью и взаимные блокировки процессов в случае работы с распределённой памятью.

Альтернативным вариантом императивного подхода является функционально-потоковая параллельная (ФПП) парадигма. Модель ФПП вычислений и реализующий эту модель язык программирования Пифагор описаны в [4]. В основе модели лежит управление вычислениями по готовности данных. При этом вычисления протекают внутри бесконечных ресурсов. Основная специфика модели заключается в том, что отношения между функциями языка позволяют представить программу в виде ациклического информационного графа. Использование языка ФПП программирования обеспечивает формирование параллелизма программ на уровне операций. Отсутствие других уровней параллелизма позволяет упростить процесс верификации, так как не требует анализа возникающих в традиционных архитектурах дополнительных ресурсных конфликтов. В настоящее время существуют работы, связанные с организацией отладки ФПП программ [5], однако вопросы формальной верификации не проработаны. Поэтому развитие методов анализа и формальной верификации ФПП программ является актуальным.

2. Аксиоматическая теория языка функционально-потокового параллельного программирования

Для выполнения формальной верификации на основе аксиоматического подхода, в первую очередь, необходимо построить аксиоматическую теорию для языка ФПП программирования Пифагор, в рамках которой и будут проводиться формальные доказательства корректности программ. Аксиоматическая теория строится по аналогии с теорией для императивных языков программирования из [1], для этого необходимо задать:

- язык теории;
- аксиомы;
- правила вывода.

2.1. Язык аксиоматической теории

Основными объектами аксиоматической теории для языка программирования Пифагор являются тройки Хоара.

Язык программирования Пифагор. Основным программо-формирующим оператором языка является оператор интерпретации. Он описывает функциональные преобразования аргумента, имеет два входа, на которые поступают функция и аргумент функции. Оператор интерпретации имеет постфиксную и префиксную формы, которые обозначаются знаком «`:`» и «`^`» соответственно. Далее в тексте используется только постфиксная форма оператора. Например, `X:F` — применение функции `F` к аргументу `X`. Ниже описаны функции языка, использующиеся в данной работе (подробное описание приведено в [6]).

Функция нахождения длины списка обозначается символом «`|`». Аргумент данной функции — список данных любой размерности и любого типа элементов. Результат — целое число, задающее количество элементов в списке первого уровня вложенности. Если аргумент не является списком, то результатом будет ошибка операции интерпретации `BASEFUNCERROR`.

Функция выбора элемента из списка задаётся как `p:n`, где `n` — целочисленная константа, применённая к списку `p`. Если `p` не является списком, функция возвращает ошибку операции интерпретации `BASEFUNCERROR`. Функция возвращает ошибку `BOUNDERROR`, если значение константы `n` по модулю больше длины списка, в случае если $\texttt{n} < 0$, из списка `p` исключается n-й элемент. При $\texttt{n} = 0$ функция возвращает пустой элемент (сигнальное значение), который обозначается знаком точка «.». Пустой элемент не имеет величины и отражает только факт появления.

Функция определения типа `type` применяется к объекту и возвращает значение его типа.

Язык логической спецификации. 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:

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

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, формирующие формулы. В данном случае можно не выделять предикаты в отдельную группу, а считать их подмножеством функций с областью значений из множества булевских переменных *bool*. Для функциональных и предикатных символов и кванторов вводятся следующие обозначения:

- арифметические операции $(+, -, *, /)$;
- знаки сравнения $(=, \neq, >, <, \geqslant, \leqslant)$;
- логические операции и кванторы $(\vee, \wedge, \neg, \Rightarrow, \Leftrightarrow, \forall, \exists)$;
- функция длины списка (len), функция выбора элемента из списка $(select)$, функция принадлежности к типу $(\in)$.

Функции len, $select$, $\in$ на своей области определения эквивалентны соответствующим функциям языка Пифагор: функции вычисления длины списка, функции выбора элемента из списка и функции определения типа.

Схемы для перечисленных функций будут совпадать со схемами функций из исчисления предикатов и арифметики. Схемы функций:

$$\{+, -, *, /\} : \{int, float\} \times \{int, float\} \to \{int, float\},$$
$$\{=, \neq, >, <, \geqslant, \leqslant\} : T \times T \to bool,$$
$$\{\vee, \wedge, \Rightarrow, \Leftrightarrow\} : bool \times bool \to bool,$$
$$\neg : bool \to bool,$$
$$len : datalist \to int,$$
$$select : datalist \times int \to T,$$
$$\in : T \times type \to bool,$$

где T — произвольный тип, для которого определена соответствующая операция.

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 (то есть элементы множеств) будем считать нуль-местными функциями;

3) других элементарных термов нет.

Элементарная формула языка — это элементарный терм типа *bool*. Тогда произвольную формулу (выражение) также 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.

Задав алфавит и правила построения выражений, можно формировать предусловия и постусловия для троек Хоара.

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, $\psi(r)$ — постусловие для `Prog`, зависящее от результата выполнения функции.

Например, рассмотрим следующую функцию на языке Пифагор:

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

Пусть P и Q — предусловие и постусловие 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 1.

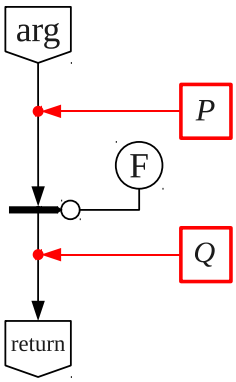

Рис. 1. Тройка Хоара для функции `Fun`

2.2. Аксиомы

В качестве аксиом в разработанной теории используются тройки Хоара для всех встроенных функций языка.

При построении аксиом используются семантические правила языка Пифагор [4, 7, 8], которые можно представить в виде ориентированных деревьев с вершинами двух типов. Ниже показано построение формулы 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. Семантическое правило, определяющее выполнение данного оператора, представлено в виде дерева на рисунке 2. Каждый путь

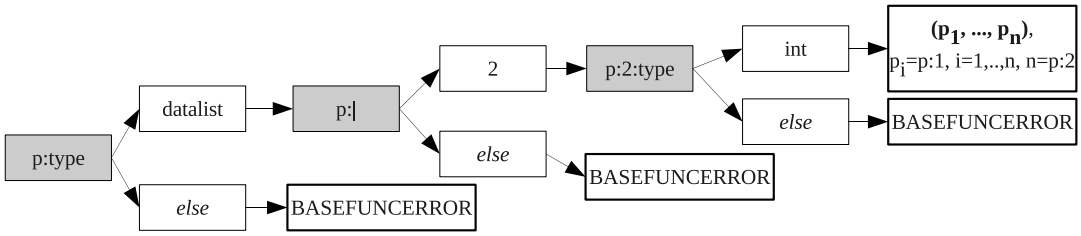

Рис. 2. Семантическое правило для функции дублирования (`p:dup`) в виде дерева

от корня к листу дерева соответствует одной аксиоме. В листьях дерева находятся результаты выполнения функции, получаемые при равенстве выражений, записанных в вершинах тёмного цвета значениям, записанным в последующих «светлых» вершинах. При этом условие равенства всех выражений из «темных вершин» соответствующим значениям из последующих «светлых» вершин есть предусловие

аксиомы, постусловие аксиомы определяется содержимым листа данного пути. Отметим, что все пути, приводящие к одинаковому результату (в нашем примере, `BASEFUNCERROR`), можно объединить в одну аксиому с помощью дизъюнкции.

Тогда аксиомы для функции дублирования будут следующими:

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

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

Особо стоит отметить ситуацию, когда сама функция f является результатом предыдущих вычислений, как, например, функция выбора элемента из списка и функция селектор. В этом случае предусловие и постусловие формируются аналогичным образом. Однако, помимо аргумента, ограничения накладываются и на функцию. Например, рассмотрим функцию селектор `p:b`, где `p` — произвольный аргумент, а `b` — булевская константа. В случае если $\texttt{b} = true$ функция возвращает `p`, иначе $\texttt{b} = false$, и возвращается сигнальное (пустое) значение. Аксиомы функции селектора имеют вид:

$$\boxed{f = true}\ \texttt{p:f}\ \to r\ \boxed{r = p}\ ,$$
$$\boxed{f = false}\ \texttt{p:f}\ \to r\ \boxed{r = .}\ .$$

2.3. Правила вывода

Для определения истинности произвольных троек Хоара вводятся правила вывода. Они описывают схему преобразования композиций операторов и позволяют получать (выводить) теоремы из имеющихся аксиом и уже доказанных теорем. Совокупность аксиом и правил вывода формирует аксиоматическую систему Хоара для языка Пифагор.

Правила вывода определяют порядок преобразования тройки Хоара в формулу исчисления предикатов, далее истинность этой формулы проверяется с помощью аппарата математической логики [9, 10]. Потребуем, чтобы процесс преобразования тройки в формулу исчисления предикатов определялся правилами вывода однозначно, то есть на каждом шаге можно было применить только одно правило.

Ввести правила вывода с этим условием можно двумя способами [1]:

1) использовать правила *прямого прослеживания*, ориентированные на вывод от аргумента функции к её результату;

2) использовать правила *обратного прослеживания*, ориентированные на вывод от результата вычисления функции к её аргументу.

В первом случае правило прямого прослеживания позволяет преобразовывать произвольную тройку Хоара, начиная от функции, применяемой непосредственно к входному значению, то есть функции, которая вычисляется первой при выполнении программы. В случае языка Пифагор таких функций может быть несколько, тогда из них выбирается произвольная. В общем случае правило прямого прослеживания для некоторой функции с кодом «$\mathtt{x{:}F_1{:}F}$» имеет следующий вид:

$$\boxed{P_1(x_1)}\ \mathtt{x_1{:}F} \to r\ \boxed{Q(r)}\ ,\ A_1,\ A_2 \quad \vdash \quad \boxed{P(x)}\ \mathtt{x{:}F_1{:}F} \to r\ \boxed{Q(r)}\ , \tag{1}$$

$$A_1 \equiv \boxed{\varphi(x)}\ \mathtt{x{:}F_1} \to x_1\ \boxed{\psi(x_1)},\ A_2 \equiv P(x) \Rightarrow \varphi(x),$$
$$P_1(x_1) \equiv P(x) \Rightarrow \varphi(x) \Rightarrow \psi(x_1),$$

где $\mathtt{x}$ — входной аргумент, $\mathtt{F_1}$ — функция, применяемая непосредственно к входному аргументу, $\mathtt{F}$ — остальная часть программы, которая может содержать другие функции, применяемые непосредственно к аргументу $\mathtt{x}$, $\vdash$ — символ выводимости, который свидетельствует о том, что при истинности выражений в левой части истинны и выражения в правой части.

Согласно этому правилу, тройка Хоара $\boxed{P(x)}\ \mathtt{x{:}F_1{:}F} \to r\ \boxed{Q(r)}$ преобразуется в тройку с более «короткой» программой $\boxed{P_1(x_1)}\ \mathtt{x_1{:}F} \to r\ \boxed{Q(r)}$, при этом предусловие $P(x)$ изменяется на $P_1(x_1)$, а исходный аргумент x заменяется на x_1 — результат применения функции $\mathtt{F_1}$ к x. В общем, правило вывода заключается в том, что при условии A_2 из истинности тройки $\boxed{P_1(x_1)}\ \mathtt{x_1{:}F} \to r\ \boxed{Q(r)}$ выводится истинность тройки $\boxed{P(x)}\ \mathtt{x{:}F_1{:}F} \to r\ \boxed{Q(r)}$ на основе аксиомы A_1 для функции $\mathtt{F_1}$.

Таким образом, при последовательном применении правила прямого прослеживания происходит «сокращение» или «свёртка» программы, и после анализа каждого оператора можно получить тройку Хоара с «пустой» программой: $\boxed{\mathrm{P}}\ \ \boxed{\mathrm{Q}}$. Это означает, что предусловие и постусловие приписаны к одной дуге информационного графа. Для завершения преобразования тройки Хоара к формуле исчисления предикатов вводится следующее правило:

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

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.

Рассмотрим, как изменяется предусловие программы при применении правила прямого прослеживания на примере следующей тройки Хоара (рис. 3):

$$\boxed{P(x)}\ \mathtt{x{:}f{:}g} \to r\ \boxed{Q(r)}$$

Пусть для функции $\mathtt{f}$ заданы аксиомы:

$$\boxed{P_i(x_a)}\ \mathtt{x}_a\mathtt{:f} \to r_a\ \boxed{Q_i(r_a)}\ ,\ i = \overline{1,n},$$

где x, r — входные и выходные аргументы программы, которые можно рассматривать как идентификаторы (метки) входных и выходных дуг информационного графа программы (необходимо, чтобы все идентификаторы были различными), а x_a и r_a — входной и выходной аргумент аксиом функции `f`, n — количество аксиом для функции `f`.

Применение правила прямого прослеживания состоит в следующем:

1. Введем уникальный идентификатор x_1 для выходной дуги выражения `x:f`, как показано на рисунке 3.А.

2. Пусть для некоторых i выполнено условие:

$$P(x) \Rightarrow P_i(x). \tag{3}$$

   Это означает, что входные данные x удовлетворяют предусловиям этих аксиом, и, следовательно, они могут быть использованы в преобразованиях по правилу прямого прослеживания. Отбрасываем все аксиомы, не удовлетворяющие (3), в результате остаётся k аксиом (которые перенумеруем от 1 до k). Эти аксиомы соответствуют k возможным путям выполнения программы, каждый из которых рассматривается отдельно. Поскольку функции в языке Пифагор полностью определены, всегда найдётся хотя бы одна аксиома, которую можно использовать.

3. Произведём преобразование тройки Хоара, «сворачивая» программу и заменяя предусловие на выражение $P(x) \Rightarrow P_i(x) \Rightarrow Q_i(x_1)$, $i = \overline{1,k}$, фрагмент кода `x:f` заменяется на идентификатор выходной дуги x_1, введенный в п. 1. (рис. 3.Б). В результате получается k новых троек Хоара:

$$\boxed{P(x) \Rightarrow P_i(x) \Rightarrow Q_i(x_1)}\ \mathtt{x}_1\mathtt{:g} \to r\ \boxed{Q(r)}\ ,\ i = \overline{1,k}. \tag{4}$$

   Исходная тройка Хоара будет истинной, если истинны все k полученных троек.

Покажем, что система Хоара с описанным правилом прямого прослеживания непротиворечива, то есть из истинных троек можно получить только истинные тройки. Без ограничения общности, рассмотрим истинность преобразованной тройки при применении аксиомы i. Если исходная тройка истинна, то после выполнения программы с исходными данными, удовлетворяющими P, будет истинно Q. Тройка Хоара для аксиомы истинна по определению (формируется на основе семантического правила). Если применять правило прямого прослеживания, то по условию истинным будет выражение $P \Rightarrow P_i$. Значит, P_i истинно, но тогда истинно и Q_i после вычисления функции `f`. В Q_i содержится выражение, которое описывает результат вычислений функции `f`, поэтому входное значение для новой тройки x_1 будет обладать необходимыми свойствами, а значит, после выполнения остального кода будет истинно и постусловие Q. Это подтверждает, что новая тройка истинна.

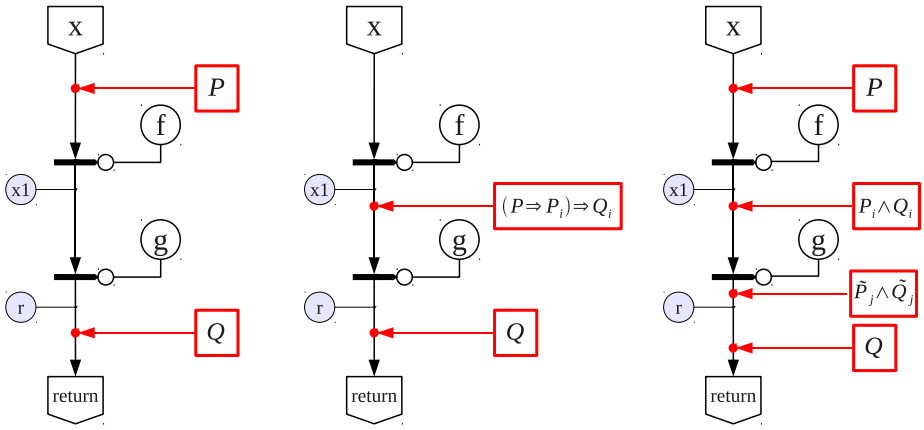

Рис. 3. Преобразования тройки Хоара функции с кодом «`x:f:g`»: А — введение идентификатора выходной дуги поддерева информационного графа функции, Б — тройка Хоара после применения правила прямого прослеживания для функции `f` на основе аксиомы i, В — тройка Хоара после разметки всех дуг

Ввиду того, что при прямом прослеживании предусловие P полагается истинным по определению, выражение $P \Rightarrow Q$ истинно, если истинно более сильное условие $P \wedge Q$, поэтому выполнение предусловия $P \Rightarrow P_i \Rightarrow Q_i$ следует из выполнения $P \wedge P_i \wedge Q_i$.

На информационном графе можно отображать процесс преобразований троек, с помощью разметки дуг формулами. Тогда каждое преобразование тройки Хоара будет заключаться в приписывании к выходной дуге рассматриваемой функции новой формулы, а все формулы ранее рассмотренных дуг будут сохраняться в неизменном виде. В этом случае окончательную формулу можно получить после того, как все дуги графа окажутся размеченными, а к выходной дуге программы будут приписаны две формулы: постусловие и формула преобразований самой внешней функции. Так, например, для тройки на рисунке 3.А после применения правила прямого прослеживания для функции `f` к входной дуге останется приписанным предусловие P, а к дуге x_1 будет приписана формула $P_i \wedge Q_i$. Следующее применение правила прямого прослеживания для функции `g` приведёт к появлению формулы $\tilde{P}_j \wedge \tilde{Q}_j$ у дуги r, где $\tilde{P}_j$ и $\tilde{Q}_j$ — предусловие и постусловие j-ой аксиомы функции g (рис. 3.В). Полученный граф является полностью размеченным, он соответствует следующей тройке Хоара с «пустой» программой: $\boxed{P(x) \wedge P_i \wedge Q_i \wedge \tilde{P}_j \wedge \tilde{Q}_j}\ \boxed{Q(r)}$.

3. Анализ корректности функционально-потоковых параллельных программ

Набор аксиом для базовых функций языка позволяет в дальнейшем использовать их для анализа корректности ФФП программ. Это делается путём последователь-

ного применения аксиом к базовым операторам и формирования на этой основе «свёртки» по описанным правилам. В зависимости от входных данных программа может иметь различные пути выполнения. Аксиомы встроенных функций полностью определяют дерево всех путей выполнения программы $\mathfrak{T}_0$. 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 исчисления предикатов по правилам вывода. В результате таких преобразований получается k формул, где k соответствует числу листьев в дереве $\mathfrak{T}_1$. Если из истинности каждой полученной формулы будет следовать истинность постусловия программы, то такая программа будет корректной. 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.

3.1. 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.

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 $\neg(\textit{условие перехода} = \textit{true}) \Leftrightarrow (\textit{условие завершения} = \textit{true})$.

Правильность рекурсии можно доказать по индукции (доказательство завершения рекурсии проводится аналогично). Введём понятие ограничивающей функции. *Ограничивающая функция* — это ограниченная снизу функция, переводящая аргу-

мент 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 корректность 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

Рассмотрим специфику доказательства корректности программы для вычисления частного и остатка от деления целых чисел. Данный пример рассматривается в работе [1] при доказательстве корректности императивной программы. Ниже приведён код программы вычисления частного и остатка от деления целых чисел x и y на языке Пифагор:

```
DIV << funcdef arg {
  x<<arg:1;  y<<arg:2;

  (x,y,0,x):div_rec >> return

}

div_rec << funcdef arg {
```

```
  x<<arg:1; y<<arg:2; q1<<arg:3; r1<<arg:4;

  ({(x,y,(q1,1):+,(r1,y):-):div_rec},(q1,r1)):
  [((y,r1):[<=, >]):?]:. >> return
}
```

Основная функция `DIV` в качестве входного значения принимает список из двух целых чисел x и y и вызывает рекурсивную функцию `div_rec`, которая вычисляет частное и остаток от деления. Исходя из условия задачи, тройка Хоара для функции `DIV` будет иметь вид (рис. 4.A):

$$\boxed{\begin{array}{l}(x,\ y \in int)\wedge\\ (x \geqslant 0) \wedge (y > 0)\end{array}}\ \texttt{(x,y):DIV}\ \rightarrow (q, r)\ \boxed{\begin{array}{l}(x = y \cdot q + r)\wedge\\ (r < y)\end{array}}.$$

Для упрощения изложения присваивание идентификаторов x и y с помощью функции выбора элемента из списка упускается, и аргумент arg сразу представляется в виде списка (x, y).

Вначале докажем корректность функции `DIV`, считая функцию `div_rec` корректной. Если функция `div_rec` корректна, то корректным значениям её входных аргументов соответствует тройка Хоара

$$\boxed{\begin{array}{l}(x,\ y,\ q_1,\ r_1 \in int)\wedge\\ (x \geqslant 0) \wedge (y > 0)\wedge\\ (q_1 \geqslant 0) \wedge (r_1 \geqslant 0)\wedge\\ (x = y \cdot q_1 + r_1)\end{array}}\ \texttt{(x,y,q1,r1):div\_rec}\ \rightarrow (q, r)\ \boxed{\begin{array}{l}(q, r \in int)\wedge\\ (q \geqslant 0)\wedge(r \geqslant 0)\wedge\\ (x = y \cdot q + r)\wedge\\ (r < y)\end{array}}, \quad (5)$$

которая будет теоремой и может быть использована при применении правила прямого прослеживания к тройке Хоара программы `DIV`. Ещё одна тройка T функции `div_rec` будет соответствовать некорректным входным аргументам, её предусловие является отрицанием предусловия тройки (5). Если входные аргументы для функции `div_rec` некорректны, то программа `DIV` сразу же считается некорректной.

Проверим выполнение условия (3). Функция `DIV` передаёт функции `div_rec` список аргументов `(x, y, 0, x)`, тогда в предусловии функции `div_rec` переменные `q1` и `r1` заменяются на `0` и `x` соответственно. Получаем выражение:

$$\begin{array}{l}P_{DIV}(x, y) \Rightarrow P_{div\_rec}(x, y, 0, x) \equiv ((x,\ y \in int) \wedge (x \geqslant 0) \wedge (y > 0)) \Rightarrow\\ \quad \Rightarrow ((x,\ y, 0 \in int) \wedge (x \geqslant 0) \wedge (y > 0) \wedge (0 \geqslant 0) \wedge (x \geqslant 0) \wedge (x = y \cdot 0 + x)) \equiv\\ \quad\quad \equiv ((x,\ y \in int) \wedge (x \geqslant 0) \wedge (y > 0)) \Rightarrow ((x,\ y \in int) \wedge (y > 0) \wedge (x \geqslant 0)\wedge\\ \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \wedge(x = x)) \equiv true.\end{array}$$

Таким образом, тройка Хоара (5) может быть использована в преобразовании тройки Хоара для функции `DIV` по правилу прямого прослеживания (4).

Отметим, что предусловие тройки T невыводимо из предусловия программы, поскольку оно является отрицанием предусловия тройки (5), поэтому тройка T отбрасывается.

Предусловие преобразованной тройки Хоара для функции `DIV` на основе теоремы (5) будет иметь вид:

$$P_{DIV} \Rightarrow P_{div\_rec} \Rightarrow Q_{div\_rec} \equiv ((x,\ y \in int) \wedge (x \geqslant 0) \wedge (y > 0)) \Rightarrow \\ \Rightarrow ((q, r \in int) \wedge (q,\ r \geqslant 0) \wedge (x = y \cdot q + r) \wedge (r < y)).$$

Или, если считать, что предусловие программы `DIV` всегда истинно, то предыдущее выражение можно записать как:

$$(x,\ y \in int) \wedge (x \geqslant 0) \wedge (y > 0) \wedge (q, r \in int) \wedge (q,\ r \geqslant 0) \wedge (x = y \cdot q + r) \wedge (r < y).$$

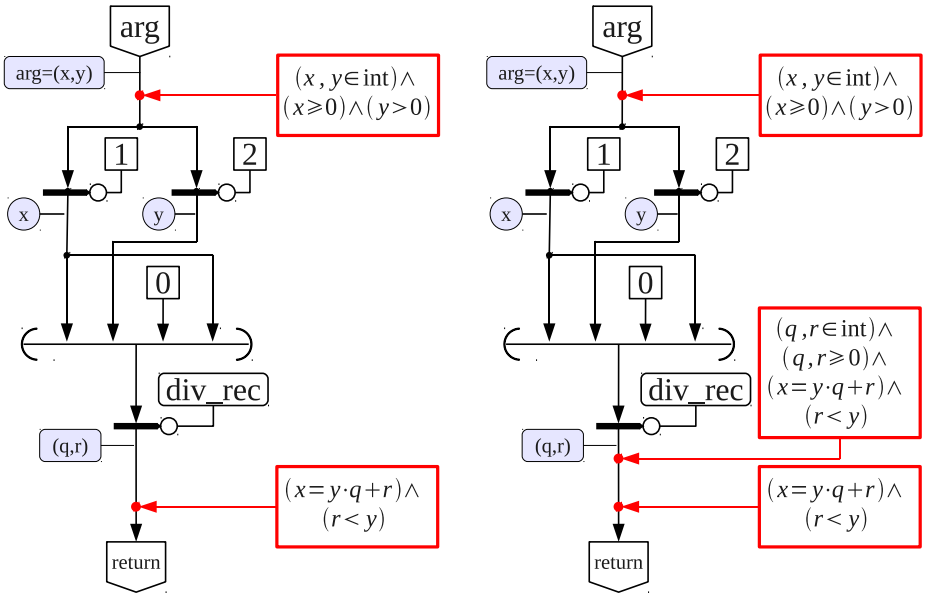

Рис. 4. Преобразование тройки Хоара программы `DIV`: А — тройка Хоара исходной программы, Б — тройка Хоара после преобразования по правилу прямого прослеживания

Преобразование программы `DIV` по правилу прямого прослеживания на основе теоремы (3) изображено на рисунке 4.Б. В результате преобразования получается тройка Хоара с «пустой» программой, которую, в свою очередь, можно преобразовать по правилу (2) в формулу

$$(x,\ y,\ q,\ r \in int) \wedge (x,\ q,\ r \geqslant 0) \wedge (y > 0) \wedge (x = y \cdot q + r) \wedge (r < y) \Rightarrow (x = y \cdot q + r) \wedge (r < y).$$

Очевидно, что полученная формула истинна, а значит, корректна и программа `DIV` (при условии корректности `div_rec`).

Докажем корректность рекурсивной функции `div_rec` с тройкой Хоара (5). Для упрощения изложения присваивание идентификаторов элементам списка входного аргумента упускается и arg сразу представляется в виде списка (x, y, q_1, p_1).

Выделим в коде программы несколько элементов:

1. `(x,y,q1,p1)` — «аргумент текущей итерации». Входной аргумент функции, записанный в виде списка.

2. `(x,y,(q1,1):+,(r1,y):-)` — «аргумент вызова следующей итерации». Выражение, определяющее аргумент для рекурсивного вызова функции.

3. `(y,r1):<=` — «условие вызова следующей итерации». Если значение указанного выражения истинно, то начинает выполняться ветвь программы, приводящая к вызову следующей итерации функции `div_rec`.

4. `(y,r1):>` — «условие завершения функции». Истинность данного выражения приводит к выполнению участка кода, не содержащего итерационный вызов функции. Данное выражение истинно тогда и только тогда, когда ложно условие вызова следующей итерации. Следовательно, функция гарантированно завершится (в программе отсутствуют другие рекурсивные вызовы функций).

Выполнение функции `div_rec` начинается с того, что `y` и `r1` входного аргумента формируют список `(y,r1)`. К этому списку применяются функции «`<=`» и «`>`». Это встроенные функции, их аксиомы одинаковые и отличаются лишь знаком операции. Для функции «`<=`» аксиомы имеют вид:

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

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

Условие (3) выполнено только для первой аксиомы. При применении правила прямого прослеживания на её основе получаются формулы, описывающие «условие вызова следующей итерации» и «условие завершения функции»: $(y \leqslant r_1)$ и $(y > r_1)$ соответственно. Значения этих формул — булевские константы, они формируют список из двух элементов, который является входным аргументом функции «`?`». Согласно своей семантике, функция «`?`» возвращает порядковый номер того элемента входного списка, чьё значение равно *true*, так как «условие вызова следующей итерации» и «условие завершения функции» взаимоисключающие, функция «`?`» возвращает либо «1», либо «2». Полученная константа используется как функция выбора элемента из списка `({(x,y,(q1,1):+,(r1,y):-):div_rec},(q1,r1))`. Это способ, с помощью которого в языке Пифагор реализуется условный выбор, в данном случае выбирается путь выполнения программы: ветвь с рекурсивным вызовом функции или ветвь без рекурсии, приводящая к завершению программы.

Выделение условий «завершения функции» и «вызова следующей итерации» при выполнении условного выбора позволяет «разделить» исходную тройку Хоара на две, а именно, исходная тройка Хоара (5) будет истинной, если истинны две

следующие тройки:

$$\boxed{\begin{array}{l}(x,\ y,\ q_1,\ r_1 \in int)\wedge\\ (x \geqslant 0) \wedge (y > 0)\wedge\\ (q_1 \geqslant 0) \wedge (r_1 \geqslant 0)\wedge\\ (x = y \cdot q_1 + r_1) \wedge (y > r_1)\end{array}} \texttt{(q1,r1):.} \rightarrow (q,r) \boxed{\begin{array}{l}(q,r \in int)\wedge\\ (q \geqslant 0)\wedge(r \geqslant 0)\wedge\\ (x = y \cdot q + r)\wedge\\ (r < y)\end{array}}, \qquad (6)$$

$$\boxed{\begin{array}{l}(x,\ y,\ q_1,\ r_1 \in int)\wedge\\ (x \geqslant 0) \wedge (y > 0)\wedge\\ (q_1 \geqslant 0) \wedge (r_1 \geqslant 0)\wedge\\ (x = y \cdot q_1 + r_1) \wedge (y \leqslant r_1)\end{array}} \texttt{prog} \rightarrow (q,r) \boxed{\begin{array}{l}(q,r \in int)\wedge\\ (q \geqslant 0) \wedge (r \geqslant 0)\wedge\\ (x = y{\cdot}q{+}r)\wedge(r < y)\end{array}}, \qquad (7)$$

где `prog` соответствует код «`{(x,y,(q1,1):+,(r1,y):-):div_rec}:.`». Предусловие первой тройки получается конъюнкцией предусловия программы `div_rec` и «условия завершения функции», а в коде вместо «точки ветвления» подставляется ветвь без рекурсии. Вторая тройка получается аналогично: предусловие — конъюнкция предусловия программы и «условия вызова следующей итерации», а в коде вместо «точки ветвления» подставляется ветвь с рекурсией.

Рассмотрим тройку (7). Первой выполняется функция «`.`», которая снимает задержку с задержанного списка, предусловие и постусловие не изменяются. Далее по правилам прямого прослеживания на основе аксиом функций «`-`» и «`+`», применяемых непосредственно к аргументу, выражения `(q,1):+` и `(r,y):-` заменяются значениями $q_2 \equiv (q_1{+}1)$ и $r_2 \equiv (r_1{-}y)$ соответственно, а тройка Хоара преобразуется к следующему виду:

$$\boxed{\begin{array}{l}(x,\ y,\ q_1,\ r_1 \in int)\wedge\\ (x \geqslant 0) \wedge (y > 0)\wedge\\ (q_1 \geqslant 0) \wedge (r_1 \geqslant 0)\wedge\\ (x = y{\cdot}q_1{+}r_1)\wedge(y \leqslant r_1)\wedge\\ (q_2 = q_1{+}1)\wedge(r_2 = r_1{-}y)\end{array}} \texttt{(x,y,q2,r2):div\_rec} \rightarrow (q,r) \boxed{\begin{array}{l}(q,r \in int)\wedge\\ (q \geqslant 0)\wedge\\ (r \geqslant 0)\wedge\\ (x = y{\cdot}q{+}r)\wedge\\ (r < y)\end{array}}. \qquad (8)$$

Теперь непосредственно к аргументу применяется рекурсивный вызов функции `div_rec`, и «аргумент следующей итерации» имеет вид `(x,y,q2,r2)`.

Зададим ограничивающую функцию для `div_rec`:

$$f(x,y,q_1,r_1) = 1 + x - (y \cdot q_1 + res),$$

$$res = x \mod y,$$

где res — остаток от деления x на y ($(x = q \cdot y + res) \wedge (res < y)$), f — целочисленная функция целочисленных аргументов, определена для всех (x,y,q_1,r_1), удовлетворяющих предусловию функции `div_rec`. Так как $(x,q_1,r_1 \geqslant 0)$, а $(y > 0)$, $f(x,y,q_1,r_1) \geqslant 1$.

Докажем корректность программы `div_rec` с помощью индукции по значениям ограничивающей функции.

База индукции. Программа завершается, если для аргумента (x, y, q_1, r_1) выполнено «условие завершения» $y > r_1$, при этом истинно выражение $x = y \cdot q_1 + r_1$ из предусловия (5). Из определения res следует, что $res = r_1$. Тогда

$$f(x, y, q1, r1) = 1 + x - (y \cdot q1 + res) = 1 + (y \cdot q_1 + r_1) - (y \cdot q1 + res) = 1.$$

Очевидно, что результат выполнения функции в этом случае удовлетворяет постусловию (5). Формальная проверка этого утверждения реализуется с помощью доказательства корректности тройки Хоара (6). Функция «.» не меняет своего аргумента, поэтому, учитывая, что $(q = q_1)$ и $(r = r_1)$, сразу можно применить правило преобразования тройки в формулу:

$$\begin{aligned}((x,\ y,\ q_1,\ r_1 \in int) \wedge (x \geqslant 0) \wedge (y > 0) \wedge (q_1 \geqslant 0) \wedge (r_1 \geqslant 0) \wedge \\ \wedge (x = y \cdot q_1 + r_1) \wedge (y > r_1)) \Rightarrow ((q_1, r_1 \in int) \wedge (q_1 \geqslant 0) \wedge (r_1 \geqslant 0) \wedge \\ \wedge (x = y \cdot q_1 + r_1) \wedge (r_1 < y))\,.\end{aligned}$$

Очевидно, что формула тождественно истинна. Также отметим, что при нарушении «условия завершения функции» имеем $f(x, y, q_1, r_1) > 1$.

Шаг индукции. Возьмём аргумент (x, y, q_1, r_1), удовлетворяющий (5), для которого выполнено «условие вызова следующей итерации» $y \leqslant r_1$ и значение ограничивающей функции равно N: $f(x, y, q_1, r_1) = N$. Предположим, что функция корректна для всех аргументов, у которых значение ограничивающей функции меньше N. Покажем, что значение ограничивающей функции «аргумента следующей итерации» всегда меньше N:

$$\begin{aligned}f(x, y, q_2, r_2) = f(x, y, q_1{+}1, r_1{-}y) = 1{+}x{-}(y{\cdot}(q1{+}1){+}res) = 1{+}x{-}(y{\cdot}q1{+}res){-}y = \\ = f(x, y, q_1, r_1) - y = N - y < N.\end{aligned}$$

Тогда по индуктивному предположению будет верна тройка Хоара для функции `div_rec`, применённой к «аргументу следующей итерации», то есть в предусловии тройки (5) `q1` и `r1` заменяются на `q2` и `r2` соответственно:

$$\boxed{\begin{array}{l}(x,\ y,\ q_2,\ r_2 \in int) \wedge \\ (x \geqslant 0) \wedge (y > 0) \wedge \\ (q_2 \geqslant 0) \wedge (r_2 \geqslant 0) \wedge \\ (x = y \cdot q_2 + r_2)\end{array}} \texttt{(x,y,q2,r2):div\_rec} \rightarrow (q, r) \boxed{\begin{array}{l}(q, r \in int) \wedge \\ (q \geqslant 0) \wedge (r \geqslant 0) \wedge \\ (x = y \cdot q + r) \wedge \\ (r < y)\end{array}}. \quad (9)$$

Полученную тройку можно использовать в качестве теоремы для доказательства корректности тройки (8) с помощью правила прямого прослеживания, считая функцию `div_rec` нерекурсивной. Сначала проверим выполнение условия (3), что эквивалентно проверке того, что предусловие функции `div_rec` выполнено для «аргумента следующей итерации». Для этого необходимо показать, что из истинности предусловия (8) следует истинность предусловия (9), что эквивалентно формуле:

$$\begin{aligned}(x,\ y,\ q_1,\ r_1 \in int) \wedge (x \geqslant 0) \wedge (y > 0) \wedge (q1 \geqslant 0) \wedge (r1 \geqslant 0) \wedge \\ (x = y \cdot q_1 + r_1) \wedge (y \leqslant r_1) \wedge (q_2 = q_1 + 1) \wedge (r_2 = r_1 - y) \Rightarrow \\ \Rightarrow (x,\ y,\ q_2,\ r_2 \in int) \wedge (x \geqslant 0) \wedge (y > 0) \wedge (q_2 \geqslant 0) \wedge (r_2 \geqslant 0) \wedge (x = y \cdot q_2 + r_2).\end{aligned}$$

Данная формула истинна, а значит, предусловие функции `div_rec` выполнено для «аргумента следующей итерации», поэтому можно провести преобразования по правилу прямого прослеживания на основе предположения индукции (9). В результате такого преобразования получается тройка с «пустой программой». После упрощения её предусловия и применения правила преобразования тройки в формулу получаем следующую формулу:

$$\begin{aligned}((x,\ y,\ q_1,\ r_1 \in int) \wedge (x \geqslant 0) \wedge (y > 0) \wedge (q_1 \geqslant 0) \wedge (r_1 \geqslant 0) \wedge \\ \wedge (x = y \cdot q_1 + r_1) \wedge (y \leqslant r_1) \wedge (q \geqslant 0) \wedge (r \geqslant 0) \wedge (x = y \cdot q + r) \wedge (r < y)) \Rightarrow \\ \Rightarrow ((q, r \in int) \wedge (q \geqslant 0) \wedge (r \geqslant 0) \wedge (x = y \cdot q + r) \wedge (r < y))\end{aligned}$$

Полученная формула истинна, а значит, корректность функции `div_rec` доказана.

5. Заключение

В работе рассмотрены особенности формальной верификация ФФП программ, выстраивается аксиоматическая теория для языка Пифагор, а также показано использование этой теории для доказательства корректности программ. Особенности языка программирования Пифагор позволяют представлять программы в виде информационных графов, что облегчает процесс отладки и верификации за счет анализа непосредственной привязки формул к информационным связям между операторами программы. 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 архитектуре. В дальнейшем рассмотренный подход предполагается положить в основу инструментальной среды обеспечивающей поддержку процесса верификации ФПП программ.

Список литературы

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. N. 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. Удалова Ю.В., Легалов А.И., Сиротинина Н.Ю. Методы отладки и верификации функционально-потоковых параллельных программ // Журнал Сибирского федерального университета. Серия: Техника и технологии. 2011. Т. 4. № 2. С. 213–224.

6. Легалов А.И., Казаков Ф.А., Кузьмин Д.А., Привалихин Д.В. Функциональная модель параллельных вычислений и язык программирования «Пифагор»: http://www.softcraft.ru/fppp.shtml

7. Легалов А.И. Использование асинхронно поступающих данных в потоковой модели вычислений / Третья сибирская школа-семинар по параллельным вычислениям. Томск: Изд-во Томского ун-та, 2006. С. 113–120.

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

9. Чень Л. Математическая логика и автоматическое доказательство теорем / Под ред. С.Ю. Маслова. М.: Наука, 1983. 358 с.

10. Harrison J. Handbook Of Practical Logic And Automated Reasoning New York: Cambridge University Press, 2009. 781 p.

Formal Verification of Programs in Functional Dataflow Parallel Language

Kropacheva M.S., Legalov A.I.

Keywords: functional data-flow parallel programming, Pifagor programming language, programs formal verification

The article is devoted to the methods of proving parallel programs correctness that are based on the axiomatic approach. Formal system for functional data-flow parallel programming language Pifagor is described. On the basis of this system programs correctness could be proved.

Сведения об авторах:
Кропачева Мария Сергеевна,
Сибирский федеральный университет, аспирант.
Легалов Александр Иванович,
Сибирский федеральный университет, профессор.