Моделирование и анализ информационных систем. Т. 25, № 6 (2018), с. 637–666
Modeling and Analysis of Information Systems. Vol. 25, No 6 (2018), pp. 637–666

©Шилов Н. В., Кондратьев Д. А., Ануреев И. С., Бодин Е. В., Промский А. В., 2018

DOI: 10.18255/1818-1015-2018-6-637-666

УДК 004.052

Платформенно-независимая спецификация и верификация стандартной математической функции квадратного корня

Шилов Н. В.[1]**, Кондратьев Д. А., Ануреев И. С., Бодин Е. В., Промский А. В.**

Поступила в редакцию 10 сентября 2018
После доработки 15 октября 2018
Принята к публикации 10 ноября 2018

Аннотация. Цель проекта "Платформенно-независимый подход к формальной спецификации и верификации стандартных математических функций" — инкрементальный комбинированный подход к спецификации и верификации стандартных математических функций, таких как `sqrt`, `cos`, `sin` и так далее. Платформенно-независимый подход предполагает простую аксиоматизацию машинной арифметики в терминах вещественной арифметики (то есть арифметики поля $\mathbb{R}$ вещественных чисел), не фиксируя ни основание системы счисления, ни формат машинного слова. Инкрементальность означает, что спецификация и верификация начинается с рассмотрения наиболее "простого" случая – элементарной спецификации и верификации простого алгоритма, работающего с вещественными числами, а заканчивается модификацией элементарной спецификации и алгоритма для машинной арифметики и верификацией алгоритма, работающего в машинной арифметике. А комбинированность подхода означает, что мы начинаем с рассмотрения "базового случая" — "ручной" верификации (с ручкой и бумагой) для алгоритма, работающего в вещественной арифметике, затем выполняем ручную верификацию алгоритма, работающего в машинной арифметике, используя верификацию для базового случая в качестве "конспекта" (proof-outlines), а заканчиваем — верификацией с использованием автоматизированной системы построения/поиска доказательства для того, чтобы исключить апелляцию к "очевидности" в ручной верификации. В статье платформенно-независимый инкрементальный комбинированный подход применяется для спецификации и верификации стандартной математической функции квадратного корня. В настоящий момент автоматизированная верификация разработанных алгоритмов выполнена только частично: с использованием системы ACL2 доказана реализуемость (существование) чисел с фиксированной запятой и таблицы начальных приближений квадратного корня.

Ключевые слова: числа с фиксированной запятой, числа с плавающей запятой, машинная арифметика, формальная верификация, частичная и тотальная корректность, тройки Хоара, метод Флойда, точная функция, квадратный корень, метод Ньютона, справочная таблица

Для цитирования: Шилов Н. В., Кондратьев Д. А., Ануреев И. С., Бодин Е. В., Промский А. В., "Платформенно-независимая спецификация и верификация стандартной математической функции квадратного корня", *Моделирование и анализ информационных систем*, **25**:6 (2018), 637–666.

Об авторах:

Шилов Николай Вячеславович, orcid.org/0000-0001-7515-9647, канд. физ.-мат. наук, доцент
Автономная некоммерческая организация высшего образования "Университет Иннополис",
ул. Университетская, 1, г. Иннополис, Республика Татарстан, 420500, Россия, e-mail: shiloviis@mail.ru

Кондратьев Дмитрий Александрович, orcid.org/0000-0002-9387-6735, аспирант
Институт систем информатики имени А.П. Ершова СО РАН,
пр. акад. Лаврентьева, 6, г. Новосибирск, 630090 Россия, e-mail: apple-66@mail.ru

Ануреев Игорь Сергеевич, orcid.org/0000-0001-9574-128X, канд. физ.-мат. наук, ст. науч. сотр.
Институт систем информатики имени А.П. Ершова СО РАН,
пр. акад. Лаврентьева, 6, г. Новосибирск, 630090 Россия, e-mail: anureev@iis.nsk.su

Бодин Евгений Викторович, orcid.org/0000-0002-5882-0365, науч. сотр.
Институт систем информатики имени А.П. Ершова СО РАН,
пр. акад. Лаврентьева, 6, г. Новосибирск, 630090 Россия, e-mail: bodin@iis.nsk.su

Промский Алексей Владимирович, orcid.org/0000-0002-5963-2390, канд. физ.-мат. наук, ученый секретарь
Институт систем информатики имени А.П. Ершова СО РАН,
пр. акад. Лаврентьева, 6, г. Новосибирск, 630090 Россия, e-mail: promsky@iis.nsk.su

Благодарности:
[1] Работа поддержана грантом РФФИ 17-01-00789.

Введение

В нашей статье речь пойдёт о, так сказать, “верификации в малом”, то есть о верификации отдельных небольших программ и функций (в нашем случае, стандартных математических функций вообще и функции квадратного корня в частности). Но наряду с “верификацией в малом” существует и “верификация в большом” — верификация сложных и больших комплексов программ, критических с точки зрения безопасности или миссии, как, например, программ управления космическим полётом, когда цена однократной программной ошибки может достигать миллиардов рублей.

Так, в декабре 2017 г. “Роскосмос” опубликовал [32] официальные результаты расследования неудачного пуска с космодрома Восточный 28 ноября того же года ракеты-носителя “Союз-2.1б” с космическим аппаратом “Метеор-М” и ещё 18 космическими аппаратами, в результате которого все 19 аппаратов были потеряны. Риски при запуске были застрахованы на сумму 2,6 млрд руб. В официальном расследовании значится следующее:

> *Сложилось такое сочетание параметров стартового стола космодрома, азимутов полета ракеты-носителя и разгонного блока, которое не встречалось ранее. Соответственно, оно не было выявлено при проведённой наземной отработке баллистической траектории согласно действующим методикам.*
>
> *Проведя всесторонний анализ, члены комиссии считают, что проявление этой некорректности алгоритма могло и не произойти при запуске с космодрома Восточный этой же полезной нагрузки с этим же разгонным блоком на этой же ракете. Пуск прошёл бы штатно, например, летом, либо в случае, если бы районы падения отделяемых частей РН лежали в стороне от выбранных.*

Однако верификация в малом так же имеет важное научное, прикладное и экономическое значение, так как “маленькие” ошибки в “маленьких”, но “массовых” (часто используемых) программах могут приводить к тем же миллиардным потерям из-за частого и повсеместного использования таких программ (стандартных функций, в частности) [11].

В настоящей статье платформенно-независимый инкрементальный комбинированный подход применяется для спецификации и верификации стандартной математической функции квадратного корня. Для вычислений в арифметике с фиксированной запятой мы выбрали основанный на методе Ньютона адаптивный алгоритм со справочной (look-up) таблицей начальных приближений квадратного корня; для спецификации этого алгоритма мы используем утверждения тотальной корректности Хоара; для аргументов, больших 1, мы доказываем методом Флойда его завершаемость в машинной арифметике с округлением к ближайшему представимому числу с ошибкой округления не более $\frac{1}{2}ULP$, где ULP — единица наименьшей точности (*Unit of Least Precision* или *Unit in the Last Place*), и с общей ошибкой вычислений не более $\varepsilon + 2ULP$, где $\varepsilon > 0$ - желаемая точность, задаваемая пользователем. Для вычислений квадратного корня в арифметике с плавающей запятой мы используем верифицированный алгоритм для вычисления квадратного корня из "приведённой мантиссы" (к значению большему 1) и деление на 2 (для чётных показателей экспоненты). Корректность алгоритма для арифметики с плавающей запятой следует из корректности алгоритма для арифметики с фиксированной запятой. В качестве автоматизированной системы формализации и поиска доказательства в нашем проекте на данном этапе используется ACL2. В настоящий момент автоматизированная верификация разработанных алгоритмов выполнена только частично: с использованием системы ACL2 доказана реализуемость (существование) чисел с фиксированной запятой и таблицы начальных приближений квадратного корня.

Прежде чем переходить к основному содержанию статьи, становимся на её структуре. В разделе 1.1. будут описаны метод Ньютона применительно к вычислению аппроксимаций квадратного корня, его алгоритмическая реализация в арифметике вещественных чисел, его спецификация условиями тотальной корректности (тройкой Хоара) и ручная верификация частичной корректности (методом Флойда для чисел, больших 1). Затем в разделе 1.2. будет доказана завершаемость нашего алгоритма в арифметике вещественных чисел и установлена квадратичная скорость сходимости этого алгоритма (при некоторых ограничениях на начальные приближения). В разделе 1.3. конкретизируется метод вычисления начальных приближений квадратного корня в арифметике вещественных чисел посредством справочной таблицы (look-up table), удовлетворяющей всем ограничениям для начальных приближений (выявленным в предыдущем разделе 1.2.), доказывается существование такой таблицы и корректность алгоритма с такой таблицей.

Вторая часть посвящена разработке (прототипированию) алгоритма аппроксимации квадратного корня и его спецификации в арифметике с фиксированной запятой и состоит из 5 разделов. В разделе 2.1. мы описываем (аксиоматизируем) нашу версию платформенно-независимой арифметики с фиксированной запятой в терминах вещественной арифметики. В разделе 2.2. мы модифицируем для арифметики с фиксированной запятой алгоритм (со справочной таблицей) аппроксимации квадратного корня и его спецификацию из раздела 1.3. первой части, доказываем существование справочной таблицы начальных приближений квадратного корня в арифметике с фиксированной запятой (при условии, что $ULP < \frac{1}{12}$). Раздел 2.3. посвящён анализу ошибок за одну итерацию цикла алгоритма аппроксимации квадратного корня в арифметике с фиксированной запятой и за несколько итераций; в этом разделе, в частности, доказывается, что при некоторых предположениях

"суммарная" (накопленная за несколько итераций) ошибка округлений не превосходит $2ULP$. В разделе 2.4. мы анализируем условия, гарантирующие завершаемость нашего алгоритма в арифметике с фиксированной запятой, а в разделе 2.5. — варианты модификации прототипа постусловия, первоначально предложенные в разделе 2.1.

Третья часть статьи посвящена спецификации и верификации алгоритмов аппроксимации квадратного корня в машинной арифметике как с фиксированной запятой, так и плавающей запятой. В разделе 3.1. мы (на основе прототипов и анализа, выполненных во второй части) приводим окончательный вариант алгоритма аппроксимации квадратного корня в арифметике с фиксированной запятой, его окончательную спецификацию, затем аннотируем этот алгоритм инвариантом цикла, а в разделе 3.2. — доказываем тотальную корректность этого полностью аннотированного алгоритма. Раздел 3.3. начинается с описания некоторых минимальных предположений об арифметике с плавающей запятой (в терминах арифметики с фиксированной запятой и вещественной арифметики) и посвящён разработке, спецификации (условиями тотальной корректности) и верификации алгоритма аппроксимации квадратного корня в арифметике с плавающей запятой.

В заключительной, четвертой части мы даём краткую сводку полученных результатов и обсуждаем их место среди исследований по верификации программного обеспечения (раздел 4.1.), даём достаточно подробный обзор литературы по верификации стандартных математических функций вообще и квадратного корня в частности, по формализации машинной арифметики (раздел 4.2.), а также намечаем планы дальнейших исследований в рамках нашего проекта "*Платформенно-независимый подход к формальной спецификации и верификации стандартных математических функций*" (раздел 4.3.).

1. Алгоритмы аппроксимации квадратного корня в вещественной арифметике, их спецификация и верификация

1.1. Метод Ньютона, его алгоритмическая реализация, спецификация и частичная корректность

Для вычисления приближённых значений квадратного корня из аргумента $y \geq 0$ широко используется *метод Ньютона* [2,7,31]. Математическое описание этого метода дано на Рис. 1.

Разумеется, что нас не интересует бесконечный процесс вычисления методом Ньютона новых приближений для $\sqrt{y}$, поэтому введем ещё один параметр $\varepsilon > 0$ для контроля точности вычислений и прерывания процесса при достижении желаемой точности. Понятие желаемой точности может быть математически формализовано двумя способами:

- x_i отличается от $\sqrt{y}$ не более чем на ε, то есть *ошибка* $|\sqrt{y} - x_i| \leq \varepsilon$;
- квадрат x_i отличается от y не более чем на ε, то есть *невязка* $|y - x_i^2| \leq \varepsilon$.

Пусть $x_1 = ini(y) > 0$ — произвольное начальное приближение для $\sqrt{y}$;
для любого $i > 0$ если i-ое приближение x_i для $\sqrt{y}$ уже определено,
то пусть $d_i = \frac{y-x_i^2}{2x_i}$ и $x_{i+1} = x_i + d_i$ (то есть $x_{i+1} = \frac{y+x_i^2}{2x_i}$).

Let $x_1 = ini(y) > 0$ be an arbitrary initial approximation for $\sqrt{y}$;
for all $i > 0$ if i-th approximation x_i for $\sqrt{y}$ has been already defined,
then let $d_i = \frac{y-x_i^2}{2x_i}$, and $x_{i+1} = x_i + d_i$ (i.e. $x_{i+1} = \frac{y+x_i^2}{2x_i}$).

Рис. 1. Математическое описание метода Ньютона
Fig. 1. Mathematical description of Newton (Newton-Raphson) method

К сожалению, контроль ошибки невозможен во время вычислений по методу Ньютона (если мы не знаем точного значения квадратного корня). Наоборот, контроль невязки легко осуществим во время вычислений по методу Ньютона. Однако, так как нас интересует вычисление квадратного корня, то по завершении вычислений мы хотели бы гарантировать именно то, что ошибка не превосходит ε. Поэтому возникает естественная идея контролировать абсолютное значение слагаемого d_i и попытаться доказать, что как только абсолютное значение мало, то ошибка не превосходит ε.

В результате мы приходим к представленной на Рис. 2 алгоритмической реализации $ANIR$ (*Adaptive Newton In Reals*) метода Ньютона для вычисления приближённого значения квадратного корня в идеальной арифметике (все операции и значения — в поле $\mathbb{R}$).

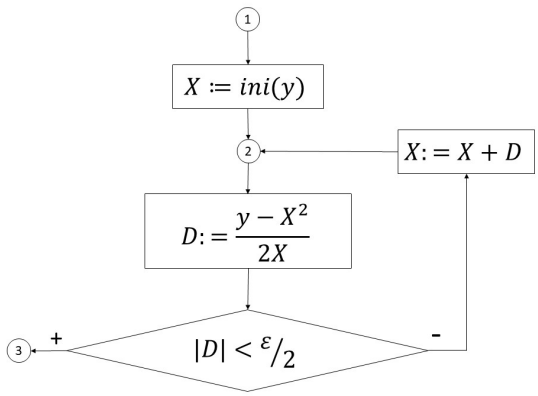

Рис. 2. Блок-схема алгоритма $ANIR$
Fig. 2. The flowchart of the algorithm $ANIR$

Этот и последующие алгоритмы вычисления приближённых значений квадратного корня мы будем применять только к аргументам $y > 1$. Поэтому специфици-

руем алгоритм следующим условием тотальной корректности Хоара:

$$[1 < y \ \&\ 0 < \varepsilon \ \&\ \sqrt{y} \leq ini(y)]\ ANIR\ [|\sqrt{y} - X| \leq \varepsilon]. \tag{1}$$

Доказательство условия (1) разобьем на доказательство утверждения частичной корректности этого же алгоритма с теми же предусловием и постусловием и доказательство завершаемости этого алгоритма, если выполнено предусловие. Доказательство частичной корректности рассмотрим в этом разделе, а доказательство завершаемости — в разделе 1.2.

Для доказательства частичной корректности выберем контрольные точки 1, 2 и 3 так, как показано на Рис. 2, аннотируем точки 1 и 3 предусловием и постусловием из (1), а в качестве инварианта (аннотации) точки 2 возьмём предусловие, усиленное условием $\sqrt{y} \leq X$. Используя метод Флойда доказательства частичной корректности, надо доказать, что

- если в контрольной точке 1 выполнена её аннотация (то есть предусловие) и путь (1..2) завершается, то после выполнения этого пути в контрольной точке 2 выполнена её аннотация (то есть инвариант);
- если в контрольной точке 2 выполнена её аннотация (то есть инвариант) и путь $(2-2)$ завершается, то после выполнения этого пути в контрольной точке 2 выполнена её аннотация (то есть тот же инвариант, но при новых значениях переменных);
- если в контрольной точке 2 выполнена её аннотация (то есть инвариант) и путь $(2+3)$ завершается, то после выполнения этого пути в контрольной точке 3 выполнена её аннотация (то есть постусловие).

Здесь и далее мы используем следующую нотацию для пути по блок-схеме алгоритма между парой контрольных точек i и j: путь заключается в круглые скобки ( ), начинается с указания начальной контрольной точки i, заканчивается указанием конечной контрольной точки j, а между ними указываются знаки рёбер, исходящих из условных операторов на этом пути, по которым прошёл путь, или две точки "..", если путь не содержит условных операторов.

Доказательство всех трёх перечисленных путей носит довольно-таки рутинный характер и было ранее представлено в [26].

1.2. Тотальная корректность алгоритма метода Ньютона

В предыдущем разделе 1.1. мы доказали частичную корректность, остаётся доказать завершаемость в случае, когда выполнено предусловие. Первый участок алгоритма (1..2) выполняется только один раз и поэтому всегда завершается. Пусть, как и в описании метода Ньютона, $x_1, x_2, \ldots\ x_n, x_{(n+1)}, \ldots$ — значения переменной X непосредственно перед первой, второй, $\ldots$ n-й, $(n+1)$-й и так далее итерациями цикла $(2-2)$, и, кроме того, $d_1, d_2, \ldots\ d_n, d_{(n+1)}, \ldots$ — значения переменной D непосредственно сразу после её вычисления на первой, второй, $\ldots$ n-й, $(n+1)$-й и так далее итерациях этого цикла. В частности, $x_1 = ini(y)$ и $d_n = \frac{y - x_n^2}{2x_n}$, $x_{(n+1)} = x_n + d_n$ для всех $n > 0$.

Выразим $d_{(n+1)}$ через d_n:

$$d_{(n+1)} = \frac{y-x^2_{(n+1)}}{2x_{(n+1)}} = \frac{y-(x_n+d_n)^2}{2(x_n+d_n)} = \frac{y-(\frac{y+x_n^2}{2x_n})^2}{2\frac{y+x_n^2}{2x_n}} = -\frac{(y-x_n^2)^2x_n}{4x_n^2(y+x_n^2)} = -\frac{d_n^2x_n}{y+x_n^2} = -\frac{d_n^2}{2x_{(n+1)}}.$$

Заметим, что, в силу инварианта цикла, все значения d_1, d_2, $\ldots$ d_n, $d_{(n+1)}$, $\ldots$ не положительные. Следовательно,

$$\frac{|d_{(n+1)}|}{|d_n|} = \frac{d_{(n+1)}}{d_n} = -\frac{d_n}{2x_{(n+1)}} = \frac{1}{2} \times \frac{x_n^2-y}{x_n^2+y} < \frac{1}{2},$$

то есть $|d_{(n+1)}| < \frac{d_1}{2^n}$. Так как $|D| < \frac{\varepsilon}{2}$ — условие завершения алгоритма $ANIR$, то этот алгоритм завершается, выполнив не более $2 + \log_2 \frac{d_1}{\varepsilon}$ итераций цикла $(2-2)$. Кроме того, мы имеем

$$|d_1| = \frac{x_1^2-y}{2x_1} = (x_1-\sqrt{y}) \times \frac{x_1+\sqrt{y}}{2x_1} < x_1-\sqrt{y} = ini(y)-\sqrt{y}.$$

Поэтому алгоритм $ANIR$ завершается, выполнив не более

$$2 + \log_2 \frac{ini(y)-\sqrt{y}}{\varepsilon} \tag{2}$$

итераций этого цикла. В частности, если начальное приближение “с точностью до целых”, то $(ini(y)-\sqrt{y}) \le 1$, и поэтому число итераций цикла не более $(2-\log_2 \varepsilon)$.

Таким образом, утверждение тотальной корректности (1) полностью доказано.

Можно, однако, обратить внимание на то, что в инварианте алгоритма $ANIR$ речь идет об оценке изменения значения переменной, а в постусловии алгоритма — об ошибке, то есть о значении разности $(X-\sqrt{y})$ после завершения алгоритма. Поэтому возникает идея перейти в инварианте от оценки этого изменения значения переменной к оценке значения самой разности $(X-\sqrt{y})$ перед очередной итерацией.

Для того чтобы найти подходящее условие для инварианта, использующее оценку разности $(X-\sqrt{y})$, проанализируем, как эта оценка меняется за одну итерацию тела цикла $(2-2)$ в алгоритме $ANIR$: пусть a — значение переменной X в начале этого пути, а b — значение этой переменной после завершения этого пути; тогда $b-\sqrt{y} = \frac{y+a^2}{2a} - \sqrt{y} = \frac{(a-\sqrt{y})^2}{2a}$. Так как начальная ошибка (то есть перед первой итерацией тела цикла) — это $(ini(y)-\sqrt{y})$, то, следовательно, в качестве оценки сверху ошибки $(X-\sqrt{y})$ после $m \ge 0$ итераций тела этого цикла можно принять

$$\frac{(ini(y)-\sqrt{y})^{2^m}}{(2\sqrt{y})^m} \tag{3}$$

(которую можно доказать индукцией по m).

Если в предусловие добавить дополнительное ограничение, что $(ini(y)-\sqrt{y}) \le \frac{1}{2}$, и памятуя, что $1 < \sqrt{y}$, оценку сверху (3) ошибки $(X-\sqrt{y})$ после $m \ge 0$ итераций тела цикла $(2-2)$ можно упростить:

$$\frac{(ini(y)-\sqrt{y})^{2^m}}{(2\sqrt{y})^m} \le \frac{1}{2^{(2^m+m)}}. \tag{4}$$

Так как при разработке алгоритма $ANIR$ мы воспользовались дополнительным предположением, что $(ini(y)-\sqrt{y}) \leq \frac{1}{2}$, то спецификация этого алгоритма отличается предусловием от спецификации алгоритма $ANIR$ (1) и может быть представлена следующим утверждением тотальной корректности:

$$[1 < y \ \&\ \& \ 0 < \varepsilon \ \&\ \sqrt{y} \leq ini(y) \leq \sqrt{y} + \frac{1}{2}] \ ANIR \ [|\sqrt{y} - X| \leq \varepsilon]. \tag{5}$$

Оценка (4) говорит, что достаточно "очень небольшого" числа итераций цикла $(2-2)$ для достижения желаемой точности ε приближённого значения квадратного корня $\sqrt{y}$. Действительно, пусть, как и выше, x_1, x_2, $\ldots$ x_n, x_{n+1}, $\ldots$ — значения переменной X непосредственно перед первой, второй, $\ldots$ n-й, $(n+1)$-й и так далее итерациями цикла $(2-2)$, а d_1, d_2, $\ldots$ d_n, d_{n+1}, $\ldots$ — значения переменной D непосредственно сразу после её вычисления на первой, второй, $\ldots$ n-й, , $(n+1)$-й и так далее итерациях этого цикла. Выберем любое $n > 0$ такое, что $\frac{1}{2^{(2^n+n)}} \leq \frac{\varepsilon}{4}$. В силу оценки сверху (4) ошибки $|X - \sqrt{y}|$, верной для любого $m \geq 0$ числа итераций тела цикла $(2-2)$, имеем

$$|d_n| = |x_{n+1} - x_n| = \left|(x_{n+1} - \sqrt{y}) \ + \ (x_n - \sqrt{y})\right| \leq \\ \leq |x_{n+1} - \sqrt{y}| \ + \ |x_n - \sqrt{y}| \leq \frac{\varepsilon}{4} + \frac{\varepsilon}{4} = \frac{\varepsilon}{2}.$$

Но, как показано выше, при верификации пути $(2+3)$, как только $|D| < \frac{\varepsilon}{2}$, то ошибка $|X - \sqrt{y}| < \varepsilon$, то есть $|x_n - \sqrt{y}| < \varepsilon$. Поэтому при том же предположении, что $(ini(y)-\sqrt{y}) \leq \frac{1}{2}$, алгоритм $ANIR$ выполнит не более $\log_2\left(1+|\log_2 \varepsilon|\right)$ итераций цикла $(2-2)$. В качестве иллюстрации скорости $ANIR$ приведём простой пример: если для вычисления $\sqrt{2}$ в качестве начального приближения выбрать $1,5$ (обратите внимание, что $y = 2 > 1$ и $\sqrt{y} < ini(y) = 1,5 \leq \sqrt{y} + \frac{1}{2}$), то

- после $n = 0$ итераций тела цикла $(2-2)$ (то есть перед началом цикла) мы точно знаем $m = (2^n + n) - 1 = 0$ двоичных цифр $\sqrt{2}$ после запятой,
- после $n = 1$ итераций тела этого цикла мы точно знаем $m = (2^n + n) - 1 = 2$ двоичных цифр $\sqrt{2}$ после запятой,
- после $n = 2$ итераций тела этого цикла мы точно знаем $m = (2^n + n) - 1 = 5$ двоичных цифр $\sqrt{2}$ после запятой,
- после $n = 3$ итераций тела этого цикла мы точно знаем $m = (2^n + n) - 1 = 10$ двоичных цифр $\sqrt{2}$ после запятой,
- и так далее.

Заметим, что именно поэтому в нескольких алгоритмах вычисления квадратного корня для компьютерных игр [7] использована явная развертка цикла $(2-2)$ на несколько итераций (на 2–4).

1.3. Адаптивный алгоритм со справочными таблицами, его спецификация и верификация

Конкретизируем способ (метод) $X := ini(y)$ реализации вычисления начального приближения для квадратного корня, использованный в алгоритме $ANIR$, в виде присваивания

$$X := UpRoot\,[SelroundUp(y)]\,,$$

где

- $SelRoundUp : \mathbb{R}^+ \to Sel \subseteq \mathbb{R}^+$ — функция из положительных вещественных чисел в некоторое фиксированное множество “избранных” положительных вещественных чисел Sel, которая для каждого вещественного числа $x > 0$ возвращает некоторое число из Sel, которое не меньше, чем x;

- $UpRoot : Sel \to \mathbb{R}$ — предвычисленная справочная таблица (look-up table, “массив”) приближений с избытком квадратных корней из избранных вещественных чисел.

В результате алгоритм $ANIR$ (Рис. 2) заменяется алгоритмом $LANIR$ (*Look-up table Adaptive Newton In Reals*), представленным на Рис. 3.

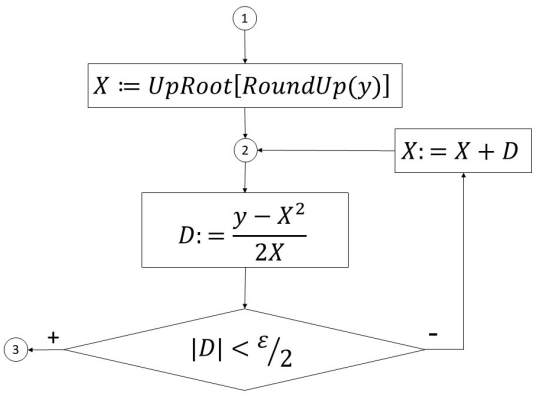

Рис. 3. Блок-схема алгоритма $LANIR$
Fig. 3. The flowchart of the algorithm $LANIR$

Специфицируем этот алгоритм следующим образом:

$$\begin{gathered}\left[1 < y \;\&\; 0 < \varepsilon \;\&\; \sqrt{y} \le UpRoot\big[SelRoundUp(y)\big] \le \sqrt{y} + \tfrac{1}{2}\right] \\ LANIR\left[\left|\sqrt{y} - X\right| \le \varepsilon\right]\end{gathered}\,. \tag{6}$$

Заметим, что корректность спецификации (6) следует из корректности спецификации (5). При этом, однако, необходимо показать, что существуют (могут быть реализованы) функция округления $SelRoundUp$ и справочная таблица $UpRoot$, которые “делают” предусловие в (6) истинным, и, следовательно, алгоритм $LANIR$ завершает работу, после чего выполнено постусловие спецификации (6).

Так как в данной части мы "работаем" с вещественной арифметикой, то функция округления и справочная таблица могут быть реализованы, например, следующим образом[1]:

1. $SelRoundUp : \mathbb{R} \to \mathbb{R}$ — посредством операции $\lambda\ z \in \mathbb{R}.\lceil z \rceil$ округления до ближайшего сверху целого числа (то есть мы принимаем $Sel = \mathbb{N}$);

2. $UpRoot : \mathbb{N}^+ \to \mathbb{R}$ — таблично-заданная функция $\lambda\ z \in \mathbb{R}^+.\sqrt{z}$, составляющая каждому положительному натуральному числу (вещественный) квадратный корень из этого числа.

Покажем, что $\sqrt{y} \leq UpRoot\,[SelRoundUp(y)] \leq \sqrt{y} + \frac{1}{2}$ для любого $y > 1$. Действительно, так как $UpRoot[SelRoundUp(y)] = \sqrt{\lceil y \rceil}$, а $\lceil y \rceil \leq (y+1)$, то

$$0 \leq UpRoot[SelRoundUp(y)] - \sqrt{y} \leq \sqrt{y+1} - \sqrt{y} = \frac{(y+1) - y}{\sqrt{y+1} + \sqrt{y}} < \frac{1}{2}.$$

Таким образом, мы показали, что для арифметики вещественных чисел существуют функция округления $SelRoundUp$ и справочная таблица $UpRoot$, которые делают предусловие спецификации (6) истинным; следовательно, при использовании этих функций и таблиц алгоритм $LANIR$ завершает работу, после чего выполнено постусловие спецификации (6).

2. Разработка и спецификация прототипа алгоритма аппроксимации квадратного корня в машинной арифметике с фиксированной запятой

2.1. Машинная арифметика с фиксированной запятой

Не все вещественные числа представимы в компьютере. Более точно, представимы только некоторые рациональные числа. Для представления существуют два формата: с фиксированной запятой и с плавающей запятой. Все рациональные числа, представимые в формате с фиксированной запятой, будем называть *числами с фиксированной запятой*, а все рациональные числа, представимые в формате с плавающей запятой, — *числами с плавающей запятой*. К формализации чисел с плавающей запятой мы вернёмся позже, а сейчас рассмотрим формализацию чисел с фиксированной запятой.

Мы определим числа с фиксированной запятой как тип данных $\mathbb{D}$, удовлетворяющий следующим аксиомам:

- множество значений этого типа $Val_{\mathbb{D}}$ — это некоторое конечное подмножество рациональных (и, следовательно, вещественных) чисел $\mathbb{Q}$ такое, что
  - оно содержит наименьшее $\inf_{\mathbb{D}} < 0$ и наибольшее $\sup_{\mathbb{D}} > 0$ числа,

[1]Здесь и далее $\lambda\ z \in \mathbb{R}.\lceil z \rceil$ — операции округления до ближайшего сверху целого числа.

  - а также все числа из диапазона $[\inf_{\mathbb{D}}, \sup_{\mathbb{D}}]$ с шагом $\delta_{\mathbb{D}} > 0$,
  - и включает все целые числа $Int_{\mathbb{D}}$ из этого диапазона $[\inf_{\mathbb{D}}, \sup_{\mathbb{D}}]$;

- допустимые операции со значениями этого типа — это
  - машинное сложение $\oplus$ и вычитание $\ominus$. Если результат математического сложения (вычитания) принадлежит диапазону $[\inf_{\mathbb{D}}, \sup_{\mathbb{D}}]$, то результат машинного сложения (соответственно — машинного вычитания) совпадает с результатом математических операций (и в этих случаях сами операции могут обозначаться обычным образом как $+$ and $-$);
  - машинное умножение $\otimes$ и машинное деление $\oslash$. Эти операции возвращают округлённое значение соответствующих математических операций с округлением к ближайшему числу с фиксированной запятой, причём для любых $x, y \in Val_{\mathbb{D}}$
    * если $x \times y \in Val_{\mathbb{D}}$, то $x \otimes y = x \times y$;
    * если $x/y \in Val_{\mathbb{D}}$, то $x \oslash y = x/y$;
    * если $x \times y \in [\inf_{\mathbb{D}}, \sup_{\mathbb{D}}]$, то $|x \otimes y - x \times y| \leq \delta_{\mathbb{D}}/2$;
    * если $x/y \in [\inf_{\mathbb{D}}, \sup_{\mathbb{D}}]$, то $|x \oslash y - x/y| \leq \delta_{\mathbb{D}}/2$;
  - округление вверх $\lceil\ \rceil$ и вниз $\lfloor\ \rfloor$ до ближайшего целого. Обе операции являются всюду определёнными на $Val_{\mathbb{D}}$;

- допустимые бинарные отношения со значениями этого типа — все стандартные равенства и неравенства в рамках диапазона $[\inf_{\mathbb{D}}, \sup_{\mathbb{D}}]$ (и поэтому обозначаются обычным образом как $=$, $\neq$, $\leq$, $\geq$, $<$, $>$).

Легко видеть, что

- если $\inf_{\mathbb{D}} \leq -1$ или $1 \leq \sup_{\mathbb{D}}$, то $\frac{1}{\delta_{\mathbb{D}}} \in \mathbb{N}$;
- $Val_{\mathbb{D}} = \{k\delta_{\mathbb{D}}\ :\ k \in \mathbb{Z},\ \inf_{\mathbb{D}} \leq k\delta_{\mathbb{D}} \leq \sup_{\mathbb{D}}\}$;
- для любых $n \in \mathbb{Z}$ и $v \in Val_{\mathbb{D}}$, если $(n \times v) \in [\inf_{\mathbb{D}}, \sup_{\mathbb{D}}]$, то $n \otimes v = n \times v$;
- так как операции $\lceil\ \rceil$ и $\lfloor\ \rfloor$ всюду определены, то $inf_{\mathbb{D}}, sup_{\mathbb{D}} \in Int_{\mathbb{D}}$.

Вариант реализации описанного выше типа данных (так сказать, виртуальная машина) доступен по ссылке (https://bitbucket.org/ainoneko/lib_verify/src/).

2.2. Прототип алгоритма и спецификации в арифметике с фиксированной запятой

При переходе от “идеальной” арифметики вещественных чисел к арифметике с фиксированной запятой алгоритм $LANIR$ с Рис. 3 должен измениться, так как теперь вместо обычных операций сложения, вычитания, умножения и деления над вещественными числами $\mathbb{R}$ используются операции $\oplus$, $\ominus$, $\otimes$ и $\oslash$. На Рис. 4 представлен вариант такого изменения — алгоритм $LANIFAv1$ (*Look-up table Adaptive Newton In Fix-point Arithmetic, version 1*). В этом изменённом алгоритме

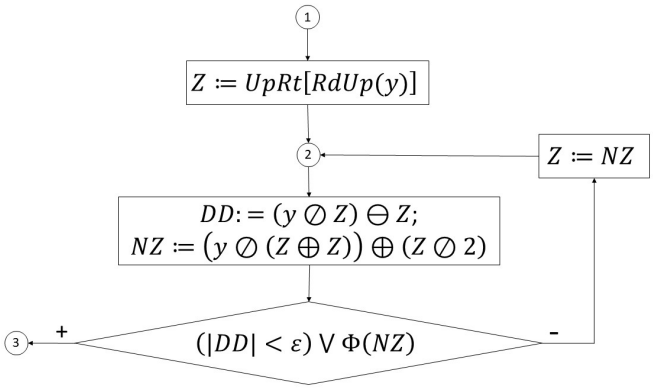

Рис. 4. Блок-схема алгоритма $LANIFAv1$
Fig. 4. The flowchart of the algorithm $LANIFAv1$

- вместо переменной X алгоритма $LANIR$ используется другая переменная Z (чтобы не возникало путаницы, да и типы этих переменных разные: вещественный $\mathbb{R}$ у X и с фиксированной запятой $\mathbb{D}$ у Z);
- присваивание $D := \frac{y-X^2}{2X}$ и следующая за ним проверка $|D| < \frac{\varepsilon}{2}$ должны были бы "превратиться" (поскольку $\frac{y-X^2}{2X} = \frac{y}{2X} - \frac{X}{2}$) в присваивание $D := (y \oslash (Z \oplus Z)) \ominus (Z \oslash 2)$ и проверку $|D| < (\varepsilon \oslash 2)$, но превращаются в присваивание $DD := (y \oslash Z) \ominus Z$ для вычисления удвоенного значения и проверку $(|DD| < \varepsilon)\&\Phi(NZ)$, где $\Phi(NZ)$ — дополнительная проверка (которую тоже ещё предстоит определить) на следующее значение переменной Z, вычисленное и сохранённое в новой переменной NZ;
- присваивание $X := X + D$ превращается (так как $D = \frac{y-X^2}{2X} = \frac{y}{2X} + \frac{X}{2}$) в два присваивания $NZ := \big(y \oslash (Z \oplus Z)\big) \oplus (Z \oslash 2)$ для вычисления следующего значения переменной Z и сохранения его в переменной NZ ($Next\ Z$) и $Z := NZ$.

Кроме того, в этом алгоритме вместо бесконечной таблицы $UpRoot : Sel \to \mathbb{R}$, которая (как было показано выше в разделе 1.3.) может "хранить" график функции $\lambda\ z \in \mathbb{R}^+.\sqrt{z}$, используется "настоящий" (конечный) предвычисленный массив $UpRt : Int^+_{\mathbb{D}} \to Val_{\mathbb{D}}$ для начальных приближений квадратного корня.

Нам бы хотелось специфицировать алгоритм $LANIFAv1$ по аналогии со спецификацией (6) алгоритма $LANIR$, например так:

$$\Big[1 < y\ \&\ 0 < \varepsilon\ \&\ \sqrt{y} \le UpRt\big[\lceil y \rceil\big] \le \sqrt{y} + \tfrac{1}{2}\Big] \\ LANIFAv1\ \Big[|\sqrt{y} - Z| \le g(\varepsilon)\Big], \tag{7}$$

где $g : Val_{\mathbb{D}} \to \mathbb{R}$ — некоторая функция (которую ещё предстоит определить), но, однако, эта спецификация (7) нуждается в модификации и предусловия, и постусловия.

Начнём с модификации предусловия спецификации (7). Во-первых, совершенно естественно, что два первых условия $1 < y$ и $0 < \varepsilon$ в предусловии при переходе к

арифметике с фиксированной запятой должны быть дополнены условиями $y \in Val_{\mathbb{D}}$ и $\varepsilon \in Val_{\mathbb{D}}$, означающими, что входные данные представимы в машине. Во-вторых, совершенно не очевидно, существует ли такой массив $UpRt : Int^{+}_{\mathbb{D}} \to Val_{\mathbb{D}}$, который удовлетворяет третьему условию $\sqrt{y} \le UpRt\big[\lceil y \rceil\big] \le \sqrt{y} + \frac{1}{2}$ из предусловия: если такого массива вообще не существует, то доказывать спецификацию (7) не нужно — она верна, так как из предусловия $FALSE$ следует и завершаемость любого алгоритма, и истинность любого постусловия. Однако в следующем абзаце будет предложено простое достаточное условие на $\delta_{\mathbb{D}}$, гарантирующее существование такого массива.

Достаточным условием для существования массива $UpRt : Int^{+}_{\mathbb{D}} \to Val_{\mathbb{D}}$, удовлетворяющего третьему условию из предусловия спецификации (7), является условие

$$\delta_{\mathbb{D}} \le \frac{1}{12}. \tag{8}$$

Действительно, если $\delta_{\mathbb{D}} \le \frac{1}{12}$, то для любого $m \ge 1$, $m, (m+1) \in Int_{\mathbb{D}}$, имеем

$$\begin{aligned}\tfrac{1}{2} \ge\ \delta_{\mathbb{D}} + \tfrac{1}{2{,}4}\ \ &\ge (\text{так как } \sqrt{m} \ge 1 \text{ и } \sqrt{m+1} \ge \sqrt{2} > 1{,}4)\ \delta_{\mathbb{D}} + \tfrac{1}{\sqrt{m+1}+\sqrt{m}}\ = \\ &= \ \delta_{\mathbb{D}} + \tfrac{(m+1)-m}{\sqrt{m+1}+\sqrt{m}}\ = \ \delta_{\mathbb{D}} + (\sqrt{m+1} - \sqrt{m}),\end{aligned}$$

то есть $(\sqrt{m} + \frac{1}{2}) - \sqrt{m+1} \ge \delta_{\mathbb{D}}$. Следовательно, интервал $[\sqrt{m+1},\ (\sqrt{m} + \frac{1}{2})]$ “шире” $\delta_{\mathbb{D}}$ и поэтому содержит некоторое число a_{m+1} вида $k\delta_{\mathbb{D}}$, которое мы примем в качестве $UpRt[m+1]$: $UpRt : m \mapsto a_m$. Для такого массива $UpRt$ и любого $y > 1$ имеем

$$\sqrt{y} \le \sqrt{\lceil y \rceil} = \sqrt{\lfloor y \rfloor + 1} \le a_{\lfloor y \rfloor + 1} = a_{\lceil y \rceil} = UpLd[\lceil y \rceil] \le \sqrt{\lfloor y \rfloor} + \frac{1}{2} \le \sqrt{y} + \frac{1}{2},$$

что и требовалось доказать для массива $UpRt$.

Доказательство существования типа данных $\mathbb{D}$ и массива $UpRt$ выполнено с использованием системы ACL2 и доступно на Github по ссылке https://github.com/apple2-66/c-light/tree/master/experiments/square-root.

2.3. Ошибки вычислений прототипа алгоритма в арифметике с фиксированной запятой

Проанализируем, как меняется ошибка $|Z - \sqrt{y}|$ за одну итерацию цикла $(2-2)$ в алгоритме $LANIFAv1$. Выберем произвольное $n > 0$, и пусть z_n и z_{n+1} — значения переменной Z непосредственно перед и сразу после n-й итерации этого цикла, а $\Delta_n = z_n - \sqrt{y}$ и $\Delta_{n+1} = z_{n+1} - \sqrt{y}$. Мы выбрали обозначения z_n и z_{n+1}, Δ_n и Δ_{n+1} чтобы не путать эти значения с соответствующими значениями x_n и x_{n+1}, d_n и d_{n+1} переменной X непосредственно перед и сразу после n-й итерации цикла $(2-2)$ в алгоритме $ANIR$ в предыдущей части 1. Имеем

$$\begin{aligned}\Delta_{n+1} = z_{n+1} - \sqrt{y} = &\ \Big(\big(y \oslash (z_n \oplus z_n)\big) \oplus (z_n \oslash 2)\Big) - \sqrt{y}\ = \\ &(\text{если } \big(y \oslash (z_n \oplus z_n)\big) \oplus (z_n \oslash 2) = \big(y \oslash (z_n \oplus z_n)\big) + (z_n \oslash 2)) \\ = &\ \Big(\big(y \oslash (z_n \oplus z_n)\big) + (z_n \oslash 2)\Big) - \sqrt{y}\ =\end{aligned}$$

$$(\text{если } z_n \oplus z_n = 2z_n) = \Big(\big(y \oslash (2z_n)\big) + (z_n \oslash 2)\Big) - \sqrt{y} =$$
$$= \big((\tfrac{y}{2z_n} + \delta') + (\tfrac{z_n}{2} + \delta'')\big) - \sqrt{y} \ (\text{где } |\delta'| \le \tfrac{\delta_{\mathbb{D}}}{2} \text{ и } |\delta''| \le \tfrac{\delta_{\mathbb{D}}}{2}) =$$
$$= \big((\tfrac{y}{2(\sqrt{y}+\Delta_n)} + \tfrac{\sqrt{y}+\Delta_n}{2}) + \delta\big) - \sqrt{y} \ (\text{где } \delta = \delta' + \delta'' \text{ и } |\delta| \le \delta_{\mathbb{D}}) =$$
$$= \big(\tfrac{y}{2(\sqrt{y}+\Delta_n)} - \tfrac{\sqrt{y}}{2}\big) + \tfrac{\Delta_n}{2} + \delta = \big(-\tfrac{\Delta_n\sqrt{y}}{2(\sqrt{y}+\Delta_n)} + \tfrac{\Delta_n}{2}\big) + \delta = \tfrac{\Delta_n^2}{2(\sqrt{y}+\Delta_n)} + \delta = \tfrac{\Delta_n^2}{2z_n} + \delta.$$

Заметим, что в приведённых выше выкладках мы использовали два предположения $z_n \oplus z_n = 2z_n$ и $\big(y \oslash (2z_n)\big) \oplus (z_n \oslash 2) = \big(y \oslash (2z_n)\big) + (z_n \oslash 2)$ о значении z_n, которые можно сформулировать в виде двух следующих условий на переменную Z (так как z_n — это значение переменной Z):

$$\begin{cases} Z \oplus Z = 2Z \\ \big(y \oslash (2Z)\big) \oplus (Z \oslash 2) = \big(y \oslash (2Z)\big) + (Z \oslash 2) \end{cases}. \tag{9}$$

Следовательно,

$$|\Delta_{n+1}| = \big|\tfrac{\Delta_n^2}{2z_n} + \delta\big| \le \tfrac{\Delta_n^2}{2z_n} + \delta_{\mathbb{D}} \le$$
$$(\text{если } 1 \le z_n) \le \tfrac{\Delta_n^2}{2} + \delta_{\mathbb{D}} < (\text{если } |\Delta_n| < 1) < \tfrac{|\Delta_n|}{2} + \delta_{\mathbb{D}},$$

то есть

$$\text{если } |\Delta_n| < 1 \text{ и } 1 \le z_n, \text{ то } |\Delta_{n+1}| < \frac{|\Delta_n|}{2} + \delta_{\mathbb{D}}. \tag{10}$$

Поэтому, если верно условие (8) и $|\Delta_n| < 1$, то $|\Delta_{n+1}| < 1$. В этих выкладках мы использовали предположение $z_n \ge 1$, которое можно сформулировать в виде следующего условия на переменную Z (так как z_n — это значение переменной Z):

$$Z \ge 1. \tag{11}$$

Далее, если $|\Delta_1| = |\sqrt{y} - z_1| < \frac{1}{2}$ и $z_1 \ge 1, \ldots\ z_m \ge 1$, то в силу (10) имеем

$$|\Delta_{m+1}| < \tfrac{|\Delta_m|}{2} + \delta_{\mathbb{D}} < \tfrac{\frac{|\Delta_{m-1}|}{2}+\delta_{\mathbb{D}}}{2} + \delta_{\mathbb{D}} = \tfrac{|\Delta_{m-1}|}{4} + (\tfrac{\delta_{\mathbb{D}}}{2} + \delta_{\mathbb{D}}) <$$
$$< \tfrac{|\Delta_{m-2}|}{8} + (\tfrac{\delta_{\mathbb{D}}}{4} + \tfrac{\delta_{\mathbb{D}}}{2} + \delta_{\mathbb{D}}) < \cdots < \tfrac{|\Delta_1|}{2^m} + \Sigma_{k=0}^{k=(m-1)} \tfrac{\delta_{\mathbb{D}}}{2^k} <$$
$$< \tfrac{|\Delta_1|}{2^m} + \Sigma_{k>0} \tfrac{\delta_{\mathbb{D}}}{2^k} = \tfrac{|\Delta_1|}{2^m} + 2\delta_{\mathbb{D}} < \tfrac{1}{2^{m+1}} + 2\delta_{\mathbb{D}},$$

то есть

$$\text{если } |\Delta_1| < \frac{1}{2} \text{ и } z_1 \ge 1, \ \ldots\ z_m \ge 1, \text{ то } |\Delta_{m+1}| < \frac{1}{2^{m+1}} + 2\delta_{\mathbb{D}}. \tag{12}$$

2.4. Завершаемость работы прототипа алгоритма в арифметике с фиксированной запятой

Таким образом, из (12) следует, что алгоритму $LANIFAv1$ следует остановиться, как только нельзя продолжить “победную” серию $z_1 \ge 1, \ldots\ z_m \ge 1$, то есть как только $z_{m+1} < 1$. Если $z_{m+1} < 1$, то мы не можем применить (10) для оценки ошибки на следующей $(m+1)$-й итерации цикла. Так как по смыслу z_m — это значение переменной Z, то z_{m+1} — это значение переменной NZ, поэтому в качестве “кандидата” для $\Phi(NZ)$ в условии выхода из цикла в алгоритме $LANIFAv1$ можно предложить условие

$$\Phi(NZ) \equiv (NZ < 1). \tag{13}$$

Важное следствие (12) состоит в том, что если $\varepsilon > 5\frac{1}{6}\delta_{\mathbb{D}}$, то цикл в алгоритме $LANIFAv1$ обязательно завершается. Действительно, этот цикл может завершиться

- или потому, что для некоторого $m \geq 0$ верно $z_1 \geq 1, \ldots\ z_m \geq 1$, но $z_{m+1} < 1$, и произойдёт выход по условию $\Phi(NZ)$,
- или потому, что на всех итерациях $z_1 \geq 1, \ldots\ z_m \geq 1, \ldots$, что гарантирует в конечном счёте выход по условию $|DD| < \varepsilon$.

Так как в первом из этих случаев завершение алгоритма очевидно, то сразу перейдём к рассмотрению второго случая. Выберем произвольное $m \geq 0$ такое, что $\frac{1}{2^m} \leq \delta_{\mathbb{D}}$. В силу (12) на m-й итерации цикла $(2-2)$ верно $|z_m - \sqrt{y}| \leq 3\delta_{\mathbb{D}}$ и, следовательно, мы имеем следующее:

$$
\begin{aligned}
|DD| = |(y \oslash z_m) \ominus z_m| = \\
&\text{(если } (y \oslash z_m) \ominus z_m = (y \oslash z_m) - z_m) = |(y \oslash z_m) - z_m| = \\
= |(\tfrac{y}{z_m} + \delta) - z_m| \ \text{(где } |\delta| \leq \tfrac{\delta_{\mathbb{D}}}{2}) = |\tfrac{y - z_m^2}{z_m} + \delta| \leq \\
&\leq |\tfrac{y - z_m^2}{z_m}| + |\delta| = |\tfrac{(\sqrt{y} - z_m)(\sqrt{y} + z_m)}{z_m}| + |\delta| = \\
= |\Delta_m|\big(1 + \tfrac{\sqrt{y}}{z_m}\big) + |\delta| \leq |\Delta_m|\big(1 + \tfrac{\sqrt{y}}{z_m}\big) + \tfrac{\delta_{\mathbb{D}}}{2} \leq \\
&\text{(так как } |z_m - \sqrt{y}| \geq 3\delta_{\mathbb{D}})) \leq |\Delta_m|\big(1 + \tfrac{\sqrt{y}}{\sqrt{y} - 3\delta_{\mathbb{D}}}\big) + \tfrac{\delta_{\mathbb{D}}}{2} \leq \\
\leq |\Delta_m|\big(2 + \tfrac{3\delta_{\mathbb{D}}}{\sqrt{y} - 3\delta_{\mathbb{D}}}\big) + \tfrac{\delta_{\mathbb{D}}}{2} \leq \\
&\text{(так как } \delta_{\mathbb{D}} \leq \tfrac{1}{12} \text{ и } 1 \leq \sqrt{y}) \leq |\Delta_m|\big(2 + \tfrac{\frac{3}{12}}{1 - \frac{3}{12}}\big) + \tfrac{\delta_{\mathbb{D}}}{2} = \\
&= \tfrac{7}{3}|\Delta_m| + \tfrac{\delta_{\mathbb{D}}}{2}.
\end{aligned}
$$

Заметим, что в этих выкладках мы использовали предположение о значении z_m, которое можно сформулировать в виде следующего условия на переменную Z (так как z_m — это опять значение именно переменной Z):

$$(y \oslash Z) \ominus Z = (y \oslash Z) - Z. \tag{14}$$

В силу свойства (12) мы имеем

$$|DD| \leq \frac{7}{3}\big(\frac{1}{2^m} + 2\delta_{\mathbb{D}}\big) + \frac{\delta_{\mathbb{D}}}{2} = \frac{7}{3} \times \frac{1}{2^m} + \frac{31}{6}\delta_{\mathbb{D}} == \frac{7}{3} \times \frac{1}{2^m} + 5\frac{1}{6}\delta_{\mathbb{D}};$$

поэтому для любого

$$\varepsilon > 5\frac{1}{6}\delta_{\mathbb{D}} \tag{15}$$

обязательно найдется такое число m, что $\big(5\frac{1}{6}\delta_{\mathbb{D}} + \frac{7}{3} \times \frac{1}{2^m}\big) < \varepsilon$; следовательно, после m итераций цикла $(2-2)$ в алгоритме $LANIFAv1$ обязательно будет выполнено условие $|DD| < \varepsilon$ завершения этого цикла, причём (забегая несколько вперёд — см. свойство (17)) после выхода из цикла будет верно условие $|\sqrt{y} - Z| < \varepsilon + \frac{\delta_{\mathbb{D}}}{2}$. Поэтому (на основании (10) и (15)) мы можем оценить ошибку вычислений следующим образом:

$$
\begin{aligned}
|\sqrt{y} - NZ| < \\
&< \big(\tfrac{1}{2}|\sqrt{y} - NZ| + \delta_{\mathbb{D}}\big) + \tfrac{\delta_{\mathbb{D}}}{2} \ < \ \big(\tfrac{1}{2}\big(\varepsilon + \tfrac{1}{2}\delta_{\mathbb{D}}\big) + \delta_{\mathbb{D}}\big) + \tfrac{\delta_{\mathbb{D}}}{2} \ < \ \tfrac{1}{2}\varepsilon + \tfrac{7}{4}\delta_{\mathbb{D}} \ < \\
&< \varepsilon + 2\delta_{\mathbb{D}};
\end{aligned}
$$

следовательно, окончательно мы получаем оценку для ошибки вычислений

$$|\sqrt{y} - NZ| < \frac{1}{2}\varepsilon + \frac{7}{4}\delta_{\mathbb{D}} < \varepsilon + 2\delta_{\mathbb{D}}. \tag{16}$$

2.5. Анализ постусловия для прототипа алгоритма в арифметике с фиксированной запятой

Теперь проанализируем, как зависит оценка ошибки $|\sqrt{y} - Z| < g(\varepsilon)$ в постусловии спецификации (7) от того, каким образом была завершена работа алгоритма $LANIFAv1$ —

- или по условию $|DD| < \varepsilon$,
- или по условию $\Phi(NZ)$, то есть $NZ < 1$ (см. (13)).

Начнём со случая выхода по условию $|DD| < \varepsilon$. При доказательстве пути $(2+3)$ для алгоритма $ANIR$ в разделе 1.1. корректность постусловия $|\sqrt{y} - X| \leq \varepsilon$ была следствием истинности условия выхода из цикла и завершения алгоритма $|D| < \frac{\varepsilon}{2}$ (разумеется, при истинности инварианта). Поэтому попробуем применить такой же подход и к алгоритму $LANIFAv1$.

Пусть для любого положительного $\varepsilon \in Val_{\mathbb{D}}$

$$g(\varepsilon) = \varepsilon + \frac{\delta_{\mathbb{D}}}{2}, \tag{17}$$

а $z \in Val_{\mathbb{D}}$ — произвольное положительное значение переменной Z. Тогда из $\varepsilon > |DD|$ следует, что

$$\begin{aligned} g(\varepsilon) = \varepsilon + \tfrac{\delta_{\mathbb{D}}}{2} > |DD| + \tfrac{\delta_{\mathbb{D}}}{2} \geq \\ \geq |DD| + |\delta| \text{ (где } \delta = (y \oslash z) - \tfrac{y}{z} \text{ и } |\delta| \leq \tfrac{\delta_{\mathbb{D}}}{2}) \geq |DD - \delta| = \\ = \Big|\big((y \oslash z) \ominus z\big) - \delta\Big| = (\text{если } (y \oslash z) \ominus z = (y \oslash z) - z) = \Big|\big((y \oslash z) - z\big) - \delta\Big| = \\ \Big|\big((y \oslash z) - \delta\big) - z\Big| = \Big|\tfrac{y}{z} - z\Big| = \Big|\tfrac{y - z^2}{z}\Big| = \Big|\tfrac{(\sqrt{y} - z)(\sqrt{y} + z)}{z}\Big| = \\ = |\sqrt{y} - z| \times \tfrac{\sqrt{y} + z}{z}. \end{aligned}$$

С другой стороны, имеем $g(\varepsilon) \leq g(\varepsilon) \times \frac{\sqrt{y}+z}{z}$; поэтому из приведённой выше цепочки неравенств следует, что $g(\varepsilon) \times \frac{\sqrt{y}+z}{z} > |\sqrt{y} - z| \times \frac{\sqrt{y}+z}{z}$ и что (в случае, когда $z > 0$) $g(\varepsilon) > |\sqrt{y} - z|$. Но это ровно то свойство, которое мы хотим от функции g: при выходе из цикла и завершении алгоритма по условию $\varepsilon > |DD|$ обязательно выполнено постусловие $g(\varepsilon) > |\sqrt{y} - Z|$ и, следовательно, условие (16)

$$|\sqrt{y} - NZ| < \frac{1}{2}\varepsilon + \frac{7}{4}\delta_{\mathbb{D}} < \varepsilon + 2\delta_{\mathbb{D}}.$$

(Заметим, что в приведённом рассуждении мы использовали условие (14), а так же условие $z > 0$, являющееся следствием из более сильного условия (11).)

Теперь рассмотрим случай выхода по условию $\Phi(NZ)$ (то есть $NZ < 1$). Пусть $n > 0$ — номер итерации цикла $(2-2)$, во время выполнения которой произошёл выход из этого цикла по этому условию, и пусть $z_n \geq 1$ и $z_{n+1} < 1$ — значения

переменной Z непосредственно перед и сразу после n-ой итерации этого цикла. Очевидно, что z_{n+1} — это значение переменной NZ перед проверкой условия выхода из цикла. Имеем

$$\begin{aligned} z_{n+1} = & \big(y \oslash (z_n \oplus z_n)\big) \oplus (z_n \oslash 2) \ = \\ & \text{(если } \big(y \oslash (z_n \oplus z_n)\big) \oplus (z_n \oslash 2) = \big(y \oslash (z_n \oplus z_n)\big) + (z_n \oslash 2)) \\ = & \big(y \oslash (z_n \oplus z_n)\big) + (z_n \oslash 2) \ = \\ & \text{(если } z_n \oplus z_n = 2z_n) = \big(y \oslash (2z_n)\big) + (z_n \oslash 2) \ = \\ = & \ (\tfrac{y}{2z_n} + \delta') + (\tfrac{z_n}{2} + \delta'') \text{ (где } |\delta'| \le \tfrac{\delta_{\mathbb{D}}}{2} \text{ и } |\delta''| \le \tfrac{\delta_{\mathbb{D}}}{2}) = \\ & = \tfrac{y}{2z_n} + \tfrac{z_n}{2} + \delta \text{ (где } \delta = \delta' + \delta'' \text{ и } |\delta| \le \delta_{\mathbb{D}}) = \tfrac{y+z_n^2}{2z_n} + \delta. \end{aligned}$$

Заметим, что в этих преобразованиях мы опять использовали условия (9). Так как $z_{n+1} < 1$, то

$$\frac{y + z_n^2}{2z_n} + \delta < 1. \tag{18}$$

Рассмотрим два случая в зависимости от того, какое из следующих двух неравенств имеет место: или $z_n < \sqrt{y}$, или $z_n \ge \sqrt{y}$.

В первом из этих случаев (если $z_n < \sqrt{y}$) вычтем $z_n \ge 1$ из (18):

$$0 = 1 - 1 > \big(\frac{y + z_n^2}{2z_n} + \delta\big) - z_n = \frac{(\sqrt{y} - z_n)(\sqrt{y} + z_n)}{2z_n} + \delta.$$

Следовательно,

$$0 < |\sqrt{y} - z_n| = \sqrt{y} - z_n < (\sqrt{y} - z_n) \times \frac{\sqrt{y} + z_n}{2z_n} < -\delta \le \delta_{\mathbb{D}},$$

то есть в этом случае имеем $|\sqrt{y} - z_n| \le \delta_{\mathbb{D}}$ и, так как z_{n+1} — это значение NZ в момент выхода из цикла, то в силу (10) имеем

$$|\sqrt{y} - NZ| \le 1\frac{1}{2}\delta_{\mathbb{D}} < \frac{1}{2}\varepsilon + \frac{7}{4}\delta_{\mathbb{D}} < \varepsilon + 2\delta_{\mathbb{D}}. \tag{19}$$

Во втором из перечисленных выше случаев (если $z_n \ge \sqrt{y}$) давайте примем z_n как начальное приближение $x_1 = ini(y)$ для квадратного корня $\sqrt{y}$ в алгоритме $ANIR$ (Рис. 2). Тогда $x_2 = \frac{y+x_n^2}{2x_1} = \frac{y+z_n^2}{2z_n} = z_{n+1} - \delta$. Согласно доказательству корректности пути $(2-2)$ алгоритма $ANIR$ (см. раздел 1.1.) $\sqrt{y} < x_2 < x_1$. Следовательно, $\sqrt{y} < (z_{n+1} - \delta) < z_n$. Но, так как $z_{n+1} < 1$ и $1 < \sqrt{y}$, то $z_{n+1} < \sqrt{y} < z_{n+1} - \delta$, то есть (так как z_{n+1} — это значение NZ в момент выхода из цикла) имеем

$$|\sqrt{y} - NZ| \le \delta_{\mathbb{D}} < \frac{1}{2}\varepsilon + \frac{7}{4}\delta_{\mathbb{D}} < \varepsilon + 2\delta_{\mathbb{D}}. \tag{20}$$

Подведём некоторый итог результатов, полученных в данной части 2.:

- в результате анализа прототипа алгоритма $LANIFAv1$ (Рис. 4) в разделе 2.4. мы пришли к выводу (см. (13)), что условие $\Phi(NZ)$ — это $NZ < 1$;
- обновлённая спецификация алгоритма должна учесть в предусловии все свойства, выявленные в части 2. при анализе прототипа спецификации 7.

3. Алгоритмы аппроксимации квадратного корня в машинной арифметике, их спецификация и верификация

3.1. От прототипа к аннотированному алгоритму в арифметике с фиксированной запятой

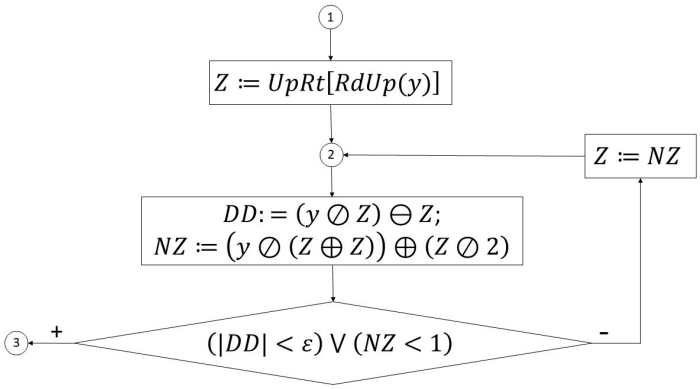

Рис. 5. Блок-схема алгоритма $LANIFAv2$
Fig. 5. The flowchart of the algorithm $LANIFAv2$

Окончательный вариант алгоритма $LANIFAv2$ (*Look-up table Adaptive Newton In Fix-point Arithmetic, version 2*, см. Рис. 5) аппроксимации квадратного корня в машинной арифметике с фиксированной запятой получается из алгоритма $LANIFAv1$ в результате подстановки $NZ < 1$ вместо $\Phi(NZ)$. А спецификация этого алгоритма $LANIFAv2$ получается из спецификации (7) прототипа алгоритма $LANIFAv1$ и выглядит следующим образом:

$$\begin{bmatrix} (a) & 1 < y \in Val_{\mathbb{D}} \ \& \\ (b) & \delta_{\mathbb{D}} \le \frac{1}{12} \ \& \\ (c) & 5\frac{1}{6}\delta_{\mathbb{D}} < \varepsilon \in Val_{\mathbb{D}} \ \& \\ (d) & \sqrt{y} \le UpRt\big[\lceil y \rceil\big] \le \sqrt{y} + \frac{1}{2} \ \& \\ (e) & 2\left(\sqrt{y} + \frac{1}{2}\right) < \sup_{\mathbb{D}} - 2\delta_{\mathbb{D}} \end{bmatrix} \quad LANIFAv2 \quad \left[\left|\sqrt{y} - NZ\right| \le \left(\varepsilon + 2\delta_{\mathbb{D}}\right)\right]. \tag{21}$$

Однако для верификации этой спецификации нам удобно “обогатить” алгоритм $LANIFAv2$ (Рис. 5) вспомогательной переменной M для счётчика итераций. Эта вспомогательная переменная никак не влияет на работу алгоритма, но нужна для аннотации контрольной точки 2 инвариантом цикла $(2-2)$. Соответствующий алгоритм $LANIFAv3$ представлен на Рис. 6.

Аннотируем контрольные точки алгоритма $LANIFAv3$ следующим образом: точки 1 и 3 — предусловием и постусловием в соответствии со спецификацией (21), а точку 2 — инвариантом, являющимся конъюнкцией предусловия спецификации (21) и трёх новых условий (f) $Z \ge 1$, (g) $|\sqrt{y} - Z| < \left(\frac{1}{2^M} + 2\delta_{\mathbb{D}}\right)$ и (h) $M > 0$.

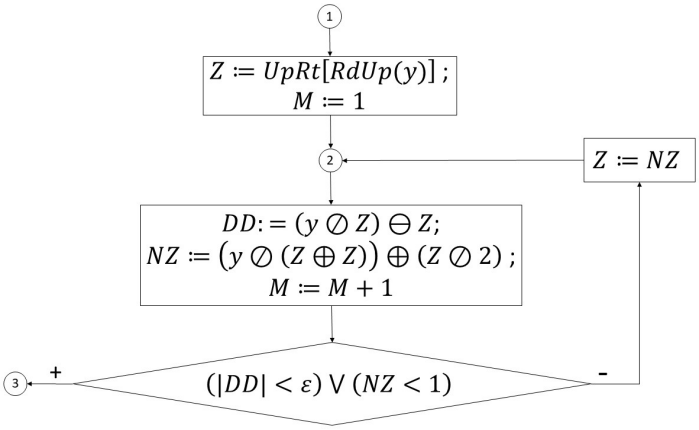

Рис. 6. Блок-схема алгоритма $LANIFAv3$, отличающаяся от $LANIFAv2$ вспомогательной переменной-счётчиком M
Fig. 6. The flowchart of the algorithm $LANIFAv3$ different from $LANIFAv2$ by a fresh counter M

3.2. Верификация аннотированного алгоритма в арифметике с фиксированной запятой

Доказательство утверждения тотальной корректности для алгоритма $LANIFAv3$ с предусловием и постусловием такими же, как в спецификации (21), разобьем на доказательство утверждения частичной корректности и доказательство завершаемости.

Для доказательства частичной корректности методом Флойда надо доказать, что

- если в контрольной точке 1 выполнена её аннотация (то есть предусловие) и путь (1..2) завершается, то после выполнения этого пути в контрольной точке 2 выполнена её аннотация (то есть инвариант);
- если в контрольной точке 2 выполнена её аннотация (то есть инвариант) и путь $(2-2)$ завершается, то после выполнения этого пути в контрольной точке 2 выполнена её аннотация (то есть тот же инвариант, но при новых значениях переменных);
- если в контрольной точке 2 выполнена её аннотация (то есть инвариант) и путь $(2+3)$ завершается, то после выполнения этого пути в контрольной точке 3 выполнена её аннотация (то есть постусловие).

Доказательство пути (1..2) тривиально. Так как значение переменной Z в конце этого пути равно $UpRt\big[\lceil y \rceil\big]$, дополнительные условия (f) и (g) следуют из условий (a) и (d) предусловия. Так как значение переменной M в конце этого пути равно 1, то выполнение условия (h) очевидно.

Доказательство пути $(2+3)$ фактически выполнено в разделе 2.5.

Докажем путь $(2-2)$. Пусть a и m — значения переменных Z и M в начале этого пути, путь завершается, а b и $(m+1)$ — значения этих же переменных после завершения этого пути. Так как этот путь завершается, то b — это и значение переменной

NZ в конце этого пути, а так как на этом пути выполнено условие $NZ \geq 1$, то $b \geq 1$, то есть выполнено условие (f) в конце этого пути. Если условие (h) выполнено в начале пути $(2-2)$, то, очевидным образом, это условие выполнено и в конце этого пути. В силу свойства (g) для a имеет место $1 \leq a < \left(\sqrt{y} + \frac{1}{2}\right)$, и, с учётом свойства (e), $2a < (\sup_{\mathbb{D}} - 2\delta_{\mathbb{D}})$. Следовательно, $2a = (a \oplus a)$. Далее имеем

$$\begin{aligned}\tfrac{y}{2a} + \tfrac{a}{2} - \sqrt{y} &= \tfrac{y}{2(\sqrt{y}+\Delta)} + \tfrac{\sqrt{y}+\Delta}{2} - \sqrt{y} \text{ (где } \Delta = a - \sqrt{y}) = \\ &= \left(\tfrac{y}{2(\sqrt{y}+\Delta)} - \tfrac{\sqrt{y}}{2}\right) + \tfrac{\Delta}{2} = -\tfrac{\Delta\sqrt{y}}{2(\sqrt{y}+\Delta)} + \tfrac{\Delta}{2} = \tfrac{\Delta^2}{2(\sqrt{y}+\Delta)} = \tfrac{\Delta^2}{2a} < \\ &< \tfrac{a-\sqrt{y}}{2} \text{ (в силу } (f) \text{ и } (g) \text{ для } a).\end{aligned}$$

Пусть $\left(\frac{y}{2a} + \delta'\right) = (y \oslash 2a)$ и $\left(\frac{a}{2} + \delta''\right) = (a \oslash 2)$, где $|\delta'| \leq \frac{\delta_{\mathbb{D}}}{2}$ и $|\delta''| \leq \frac{\delta_{\mathbb{D}}}{2}$. Тогда

$$((y \oslash (a \oplus a)) + (a \oslash 2)) = \left(\tfrac{y}{2a} + \tfrac{a}{2}\right) + (\delta' + \delta'') < \sqrt{y} + \tfrac{a-\sqrt{y}}{2} = \tfrac{a+\sqrt{y}}{2} < \sup_{\mathbb{D}},$$

и поэтому

$$b = ((y \oslash (a \oplus a)) \oplus (a \oslash 2)) = ((y \oslash 2a) + (a \oslash 2))$$

и

$$\begin{aligned}|b - \sqrt{y}| = \left|\left(\tfrac{y}{2a} + \tfrac{a}{2} - \sqrt{y}\right) + (\delta' + \delta'')\right| &\leq \left|\tfrac{y}{2a} + \tfrac{a}{2} - \sqrt{y}\right| + |\delta' + \delta''| < \\ &< \tfrac{|a-\sqrt{y}|}{2} + \delta_{\mathbb{D}} < \tfrac{\left(\frac{1}{2^m} + 2\delta_{\mathbb{D}}\right)}{2} + \delta_{\mathbb{D}} = \tfrac{1}{2^{(m+1)}} + 2\delta_{\mathbb{D}}.\end{aligned}$$

Следовательно, условие (g) тоже выполнено в конце пути $(2-2)$.

Таким образом, доказана частичная корректность обоих алгоритмов $LANIFAv2$ (Рис. 5) и $LANIFAv3$ (Рис. 6) для вычислений аппроксимаций квадратного корня в арифметике с фиксированной запятой. Это означает, в частности, что доказан инвариант контрольной точки 2 алгоритма $LANIFAv3$ и его условия (f) и (g). Кроме того, значение переменной M в этом алгоритме $LANIFAv3$ возрастает на каждой легальной итерации цикла $(2-2)$. Поэтому для доказательства завершаемости цикла $(2-2)$ в алгоритме $LANIFAv3$ мы можем применить рассуждения, уже описанные в разделе 2.4. Остаётся повторить замечание, что переменная M никак не влияет на работу алгоритма $LANIFAv3$ и, следовательно, оба алгоритма $LANIFAv2$ и $LANIFAv3$ на одних и тех же значениях y и ε или завершаются одновременно (выполнив одно и то же число итераций цикла $(2-2)$), или оба не завершаются. Так как мы доказали завершаемость алгоритма $LANIFAv3$, если выполнено предусловие (21), тем самым доказана и завершаемость алгоритма $LANIFAv2$, если также выполнено предусловие. Тем самым, мы доказали завершаемость и тотальную корректность обоих алгоритмов $LANIFAv2$ (Рис. 5) и $LANIFAv3$ (Рис. 6).

3.3. Алгоритм аппроксимации квадратного корня в арифметике с плавающей запятой, его спецификация и верификация

Идея алгоритма вычисления квадратного корня в арифметике с плавающей запятой математически может быть изложена очень просто: если вещественное число $a \in \mathbb{R}$, $a \geq 0$ представлено в форме $a = m \times \beta^p$, где $m \in \mathbb{R}$ — “мантисса” этого числа,

$\beta \in \mathbb{R}^+$ — (положительное) “основание” (экспоненты), а $p \in \mathbb{Z}$ — “’экспонента” (вернее, показатель экспоненты) этого числа, то

$$\sqrt{a} = \begin{cases} \sqrt{m} \times \beta^{\frac{p}{2}}, & \text{если } p \text{ — чётное число;} \\ \sqrt{m \times \beta} \times \beta^{\frac{p-1}{2}}, & \text{если } p \text{ — нечётное число.} \end{cases} \tag{22}$$

Алгоритм $FSQRT$ на Рис. 7 реализует эту простую математическую идею для чисел с плавающей запятой с использованием чисел с фиксированной запятой для мантиссы, основания и экспоненты.

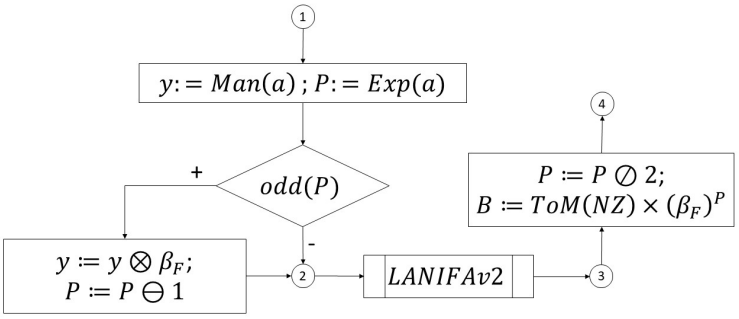

Рис. 7. Блок-схема алгоритма $FSQRT$
Fig. 7. The flowchart of the algorithm $FSQRT$

В отличие от аксиоматизации арифметики с фиксированной запятой, описанной в разделе 2.1., мы не будем аксиоматизировать *всю* арифметику с плавающей запятой, а только аксиоматизируем несколько свойств и операций, которые преобразуют числа с плавающей запятой в числа с фиксированной запятой и наоборот. Пусть $\mathbb{D}$ — тип для чисел с фиксированной запятой, удовлетворяющий аксиоматизации, описанной в 2.1. Определим тип для чисел с плавающей запятой $\mathbb{F}$ следующим образом:

- множество значений этого типа $Val_{\mathbb{F}}$ — это конечное подмножество множества вещественных чисел $\mathbb{R}$, состоящее из некоторых чисел из диапазона $[\inf_{\mathbb{F}}, \sup_{\mathbb{F}}]$, где $\inf_{\mathbb{F}} < 2$, $\sup_{\mathbb{F}} > 2$ и $\{0, \inf_{\mathbb{F}}, \sup_{\mathbb{F}}\} \subseteq Val_{\mathbb{F}}$;
- есть три унарные операции:
  - *мантисса* $Man : Val_{\mathbb{F}} \to Val_{\mathbb{D}}$,
  - *экспонента* $Exp : Val_F \to Int_{\mathbb{D}}$,
  - *преобразование в мантиссу* $ToM : Val_{\mathbb{D}} \to Val_{\mathbb{D}}$,

  а также константы
  - *основание* (целое число) $\beta_T \in Int^+_{\mathbb{D}}$ (натуральное число с фиксированной запятой),

– *дефект* (мантиссы) $\mu_{\mathbb{F}} \in Val^{+}_{\mathbb{D}}$ (положительное число с фиксированной запятой)

такие, что:

– для любого положительного числа $x \in Val_{\mathbb{F}}$
  * $1 < Man(x) < \frac{sup_{\mathbb{D}}}{\beta_{\mathbb{F}}}$, и $x = Man(x) \times \beta_{\mathbb{F}}^{Exp(x)}$ (где "$\times$" — математическая операция умножения);
  * если $inf_{\mathbb{D}}$ нечётное число, то $inf_{\mathbb{D}} < Exp(x)$, и $inf_{\mathbb{D}} \leq Exp(x)$, если $inf_{\mathbb{D}}$ — чётное;
– $|x - ToM(x)| \leq \mu_{\mathbb{F}}$ для любого положительного числа $x \in Val_{\mathbb{D}}$.

Во-первых, заметим, что по нашему определению диапазон изменения мантиссы — это некоторый подынтервал $(1, \frac{sup_{\mathbb{D}}}{\beta_{\mathbb{F}}})$, в то время как общепринятое определение фиксирует диапазон $[0.1, 1)$. Причина нашего выбора диапазона изменения мантиссы простая: в предыдущем разделе 3.1. был описан, а в разделе 3.2. верифицирован алгоритм $LANIFAv2$ (Рис. 5) аппроксимации квадратного корня в арифметике с фиксированной запятой для чисел, больших 1 (см. спецификацию (21)).

Во-вторых, заметим, что свойство $x = Man(x) \times \beta_{\mathbb{F}}^{Exp(x)}$ использует именно *математическую* операцию умножения, а не её "машинный" аналог. Но это использование операции умножения на самом деле соответствует представлению числа с плавающей запятой в компьютере в машинном слове в виде мантиссы и экспоненты, и поэтому операция умножения в данном случае — это просто операция "сборки" числа с плавающей запятой из его мантиссы и экспоненты (без потери точности).

В-третьих, "обратная" операция преобразования в мантиссу по нашему определению может приводить к потере точности мантиссы, что означает, что не все числа с фиксированной запятой являются мантиссами чисел с плавающей запятой (мантисса числа с плавающей запятой имеет меньше разрядов, чем число с фиксированной запятой).

Как уже было сказано, алгоритм аппроксимации квадратного корня для чисел с плавающей запятой $FSQRT$ представлен на Рис. 7. В этом алгоритме

- $LANIFAv2$ — это уже знакомый нам алгоритм с Рис. 5 с входными переменными y и ε и выходной переменной NZ типа $\mathbb{D}$ (с фиксированной запятой),
- переменные a и B типа $\mathbb{F}$ (с плавающей запятой) являются ещё одной входной и, соответственно, единственной выходной переменной этого алгоритма $FSQRT$,
- переменная P имеет тип с фиксированной запятой $\mathbb{D}$ (но принимает только целые значения из $Int_{\mathbb{D}}$),
- операции $\otimes$, $\otimes$ и $\oslash$ — это машинные операции умножения, вычитания и деления для арифметики с фиксированной запятой, описанные в разделе 2.1.),
- а константа $\beta_{\mathbb{F}}$ — это основание экспоненты (то есть число с фиксированной запятой типа $\mathbb{D}$).

Заметим также, что алгоритм $FSQRT$ выглядит ациклическим, но на самом деле он содержит цикл, “спрятанный” в алгоритме $LANIFAv2$.

Так как алгоритм $FSQRT$ основан на алгоритме $LANIFAv2$, то его спецификация получается из спецификации (21), но по отдельности рассматривает случаи чётной и нечётной экспоненты входной переменной a (как это уже сделано в (22)). Для удобства выделим общую часть предусловия

$$(a,b,c) \equiv \begin{cases} (a) & 0 < a \in Val_{\mathbb{F}} \ \& \\ (b) & \delta_{\mathbb{D}} \leq \frac{1}{12} \ \& \\ (c) & 5\frac{1}{6}\delta_{\mathbb{D}} < \varepsilon \in Val_{\mathbb{D}}. \end{cases} \tag{23}$$

Тогда спецификация алгоритма $FSQRT$ выглядит следующим образом:

Чётная экспонента:

$$\begin{array}{l} \left[\begin{array}{ll} (a,b,c) & \& \\ (d) & \sqrt{Man(a)} \leq UpRt\big[\lceil Man(a)\rceil\big] \leq \\ & \qquad\qquad \leq \sqrt{Man(a)} + \frac{1}{2} \ \& \\ \\ (e) & 2\left(\sqrt{Man(a)} + \frac{1}{2}\right) < \sup_{\mathbb{D}} - 2\delta_{\mathbb{D}} \end{array}\right] \\ \qquad\qquad\qquad FSQRT \\ \qquad\qquad\qquad \left[|\sqrt{a} - B| \leq (\varepsilon + 2\delta_{\mathbb{D}} + \mu_{\mathbb{F}}) \times \beta_{\mathbb{F}}^{\frac{Exp(a)}{2}}\right] \end{array} \tag{24}$$

Нечётная экспонента:

$$\begin{array}{l} \left[\begin{array}{ll} (a,b,c) & \& \\ (d) & \sqrt{Man(a) \times \beta_{\mathbb{F}}} \leq UpRt\big[\lceil man(a)\rceil\big] \leq \\ & \qquad\qquad \leq \sqrt{Man(a) \times \beta_{\mathbb{F}}} + \frac{1}{2} \ \& \\ \\ (e) & 2\left(\sqrt{Man(a) \times \beta_{\mathbb{F}}} + \frac{1}{2}\right) < \sup_{\mathbb{D}} - 2\delta_{\mathbb{D}} \end{array}\right] \\ \qquad\qquad\qquad FSQRT \\ \qquad\qquad\qquad \left[|\sqrt{a} - B| \leq (\varepsilon + 2\delta_{\mathbb{D}} + \mu_{\mathbb{F}}) \times \beta_{\mathbb{F}}^{\frac{Exp(a)-1}{2}}\right] \end{array} \tag{25}$$

Фактически в этой спецификации просто явно использовано то, что в алгоритме $FSQRT$ (Рис. 7) на вход алгоритма $LANIFAv2$ (Рис. 5) в качестве значения переменной y передаётся или значение $Man(a)$, или значение $Man(a) \times \beta_{\mathbb{F}}$ (в зависимости от чётности показателя экспоненты), и, следовательно, предусловия в (24) и (25) получаются из предусловия в (21) в результате подстановки $Man(a)$ или $Man(a) \times \beta_{\mathbb{F}}$ вместо y. Аналогично, постусловия в (24) и (25) получаются из постусловия в (21) в результате прямого просачивания постусловия из (21) через присваивания

$$P := P \oslash 2 \ ; \ B := ToM(NZ) \times \beta_{\mathbb{F}}^{P},$$

в результате чего в оценке ошибки в арифметике с фиксированной запятой $\varepsilon + 2\delta_{\mathbb{D}}$ появляется дефект мантиссы $\mu_{\mathbb{F}}$ в качестве “довеска”.

4. Заключение

4.1. Сумма результатов статьи

В качестве эпиграфа к нашей статье (и проекту "*Платформенно-независимый подход к формальной спецификации и верификации стандартных математических функций*") можно было бы взять следующую цитату из аннотации работы [20]:

> *Current critical systems commonly use a lot of floating-point computations, and thus the testing or static analysis of programs containing floating-point operators has become a priority. However, correctly defining the semantics of common implementations of floating-point is tricky, because semantics may change with many factors beyond source-code level, such as choices made by compilers.*

Однако есть существенная разница между [20] и нашей статьей. Цитируемая статья посвящена вопросам представления чисел с плавающей запятой и реализации арифметических операций, а наша статья — представлению платформенно-независимого инкрементального комбинированного подхода к спецификации и верификации стандартных функций и его применению для спецификации и верификации стандартной математической функции квадратного корня.

Платформенная независимость отличает нашу работу от работ по формальной спецификации и верификации стандартных функций для конкретных процессорных архитектур Intel [14,15,17] и Oracle [11], или чётко описанных форматов представления чисел с фиксированной и плавающей запятой [5,13,20]. Мы только формулируем достаточно общие свойства арифметики с фиксированной запятой и только минимально необходимые свойства арифметики с плавающей запятой, которые, как мы надеемся, будет легко проверить для конкретной архитектуры и платформы. Отметим сразу, что на данный момент мы только прототипировали вариант реализации арифметики с фиксированной запятой, а проверка, что существующие реализации арифметики с фиксированной запятой удовлетворяют нашей модели, пока остаётся задачей на перспективу.

Кроме того, подход, представленный в данной статье, отличает инкрементальность, состоящая в том, что сначала мы специфицируем и вручную верифицируем алгоритм для идеальной арифметики вещественных чисел, потом используем эту спецификацию и верификацию как эскиз для спецификации и ручной верификации алгоритма в машинной арифметике с фиксированной запятой, а затем комбинируем ручную верификацию с автоматической, то есть формализуем и проверяем наше доказательство целиком средствами автоматизированной системы построения доказательств. На данный момент мы располагаем только формализованным доказательством существования справочной таблицы начальных приближений для квадратного корня, выполненное с использованием системы ACL2.

Следует обратить внимание на то, что согласно стандарту IEEE-754 [29], аппроксимация квадратного корня должна быть "точной" (exact) в следующем смысле: возвращаемый результат должен отличаться от истинного математического результата не более чем на $\frac{1}{2}ULP$, в то время как лучшая из доказанных нами в настоящей

статье оценок для ошибки (в арифметике с фиксированной запятой)

$$\varepsilon + 2\delta_{\mathbb{D}} \text{ (причём } 5\frac{1}{6}\delta_{\mathbb{D}} < \varepsilon),$$

которая фигурирует в утверждении тотальной корректности (21), в пределе даёт оценку $7\frac{1}{6}\delta_{\mathbb{D}}$. Может сложиться впечатление, что наш алгоритм $LANIFAv2$ для арифметики с фиксированной запятой и, следовательно, алгоритм для арифметики с плавающей запятой $FSQRT$, не являются точными. Однако, если принять, что наш тип данных $\mathbb{D}$, использованный для вычислений с фиксированной запятой, — это "внутренний" тип данных, используемый только для обеспечения нужной точности вычислений во "внешнем" типе $\mathbb{T}$, то ситуация изменится: если всякое число с плавающей запятой типа $\mathbb{T}$ автоматически является числом типа $\mathbb{D}$, а при обратном преобразовании из типа $\mathbb{D}$ в тип $\mathbb{T}$ происходит округление с "потерей" 4 младших двоичных разрядов (то есть $16\delta_{\mathbb{D}}$), то вычисления, выполненные в типе $\mathbb{D}$ с ошибкой $7\frac{1}{6}\delta_{\mathbb{D}}$, становятся точными для типа $\mathbb{T}$ (так как $7\frac{1}{6}\delta_{\mathbb{D}} < 8\delta_{\mathbb{D}} = \frac{16}{2}\delta_{\mathbb{D}}$). Такая интерпретация соответствует практике реализации вычислений со значениями типа `float`, во время которых все вычисления реально проходят с типом `double`, а по завершении вычислений результат типа `double` округляется до значения типа `float`. Именно по этой причине мы выбрали обозначение $\mathbb{D}$ (Double) для "внутреннего" типа с фиксированной запятой. Аналогичная ситуация у нас возникала с мантиссами в разделе 3.3.: всякая мантисса определялась числом с фиксированной запятой, но при обратном преобразовании происходила потеря точности.

4.2. Обзор литературы

Актуальность проблемы спецификации и валидации стандартных функций хорошо осознана научным и инженерным сообществом. Можно, например, сослаться на работы [1,19], в которых всесторонне обсуждается проблема спецификации и тестирования стандартных математических функций, причём во внимание принимается не только точность вычислений (ошибка или невязка), но и такие алгебраические свойства стандартных функций, как, например, чётность и нечётность (cos и sin), нормализация ($\sin^2 x + \cos^2 x = 1$). Образовательное значение проблемы спецификации и верификации стандартных функций также нашло отражение в литературе [24,25].

Достаточно обширная литература посвящена формальной верификации реализации стандартных математических функций в арифметике с плавающей запятой для конкретных платформ и архитектур: в работах [8,15] верифицирована функция (операция) деления, в работах [3,8,15,23] — функция квадратного корня, в [14] — некоторые тригонометрические функции, в [16] — степенная функция, а в [27] — гамма-функция. (Кстати, даже автоматизированная верификация целочисленных стандартных функций является нетривиальной задачей, см., например, работу [3], в которой доказана с использованием системы PVS функция целой части квадратного корня.)

Есть и публикации по аксиоматизации машинной арифметики с плавающей запятой и, прежде всего, бинарной машинной арифметики в соответствии со стандартом IEEE-754, с доказательством основных свойств и корректности некоторых стандартных математических функций [4–6,13,22].

Так, в работе [5] представлена формализация машинной арифметики в Z-нотации и её реализация на языке Occam. В частности, в цитируемой работе описаны следующие операции и функции в арифметике с плавающей запятой: округления, сложения, умножения, квадратного корня, преобразования в целый тип, сравнения и другие. Кроме того, в работе [5] формализованы пять классов чисел с плавающей запятой (NaN, Inf, $zero$, нормализованные и денормализованные числа) и четыре варианта округления, в то время как в нашей статье мы используем только один вариант округления к ближайшему числу. Реализация на языке Occam включает представление чисел с плавающей запятой, операции округления и другие. Шаблон работы с числами с плавающей запятой в [5]) состоит в следующем:

1. "распаковка" числа, то есть вычисление знака, мантиссы и экспоненты числа;
2. денормализация числа (выравнивание экспонент);
3. выполнение операции с денормализованными числами;
4. "упаковка" результатов операции (то есть "сборка" числа из знака, мантиссы и экспоненты).

В работе [13] представлен подход с использованием системы HOL Light к спецификации и верификации нескольких операций с плавающей запятой в архитектуре Intel IA-64:

> *Correctness of the mathematical software starts from the assumption that the underlying hardware floating point operations behave according to the IEEE standard 754 for binary floating point arithmetic. Actually, IEEE-754 doesn't explicitly address floating-point machine arithmetic operations, and it leaves underspecified certain significant questions, e.g. NaN propagation and underflow detection. Thus, we not only need to specify the key IEEE concepts but also some details specific to IA-64.*

В цитируемой статье сначала представлена платформенно-независимая теория чисел с плавающей запятой, которая затем специализируется для архитектуры IA-64: в теории числа с плавающей запятой представляются в весьма общем виде $\pm k \times 2^{E-N}$, но затем используются стандартные форматы представления, распаковки и упаковки, денормализации и нормализации. В работе [13] также обсуждается проблема определения понятия ULP, которое имеет определённые разночтения (но, в конце концов, в цитируемой работе принимается определение ULP из [21]). В [13] (так же, как и в [5]) рассмотрены 4 варианта округления результата выполнения операций над числами с плавающей запятой:

- к ближайшему числу с плавающей запятой (как у нас в статье),
- а также вниз, вверх и по направлению к нулю.

В принципе числа с плавающей запятой в [13] могут иметь неограниченную экспоненту (что отличается от стандарта IEEE-754), но во время выполнения операций переполнение памяти обрабатывается должным образом.

В работе [6] предпринята попытка описать логическую теорию (синтаксис и семантика) арифметики чисел с плавающей запятой и использовать SMT-решатели (Satisfiability Modulo Theories) в качестве разрешающей процедуры. Заметим, что в нашей статье мы не стремились дать сколько-либо полную теорию ни для арифметики с фиксированной запятой, ни для арифметики с плавающей запятой, а ограничились только свойствами, которые нам понадобились для спецификации и верификации конкретной проблемы — аппроксимации квадратного корня.

Верификация аппроксимаций квадратного корня вызывает особый интерес. Так, работа [8] посвящена верификации алгоритма вычисления аппроксимации функции квадратного корня, основанного на так называемом методе "цифра за цифрой" (digit-by-digit или CORDIC — COordinate Rotation DIgital Computer), который вычисляет цифры вещественного числа. В работах [9, 23] доказывается корректность алгоритма аппроксимации квадратного корня, использующего многочлены Чебышёва и реализованного в процессоре Power4 (хотя алгоритм, основанный на многочленах Чебышёва, выполняет больше шагов, чем метод Ньютона для достижения нужной точности). Верификация алгоритма аппроксимации квадратного корня в работах [9, 23] разделена на два этапа: доказательство корректности разложения функции квадратного корня в ряд Тейлора и доказательство свойств функции квадратного корня с использованием этого разложения. Все доказательства в работе [23] выполнены с использованием библиотеки ACL2, формализующей нестандартный вещественный анализ.

В конце литературного обзора мы остановимся на сравнении нашего препринта [26] с настоящей статьей. В препринте [26] описаны, специфицированы и (вручную) верифицированы *неадаптивные* алгоритмы аппроксимации квадратного корня для чисел с фиксированной и плавающей запятой с *накоплением ошибки* вплоть (для чисел с фиксированной запятой) до $2\delta_{\mathbb{D}}(2 + \log_2 \frac{S}{\varepsilon})$, где S — "шаг" справочной таблицы начальных приближений для квадратного корня (в настоящей статье $S = 1$). В настоящей работе описаны, специфицированы и (вручную) верифицированы *адаптивные* алгоритмы аппроксимации квадратного корня, достигающие (для чисел с фиксированной запятой) точности $7\frac{1}{6}\delta_{\mathbb{D}}$, рассмотрено прототипирование модели чисел с фиксированной запятой и автоматизированное доказательство (в системе ACL2) существования массива начальных приближений квадратного корня.

4.3. План дальнейших исследований

Первая ближайшая практическая задача — полная автоматизированная (с использованием, вероятно, системы ACL2) проверка ручного доказательства утверждения тотальной корректности (21) для алгоритма аппроксимации квадратного корня $LANIFAv2$ (Рис. 5) для чисел с фиксированной запятой и утверждений тотальной корректности (25) и (24) для алгоритма аппроксимации квадратного корня $FSQRT$ (Рис. 7) для чисел с плавающей запятой. Напомним, что на данном этапе пока выполнена автоматизированная верификация только существования массива начальных приближений квадратного корня в арифметике с фиксированной запятой.

Вторая ближайшая практическая задача — прототипирование алгоритма $LANIFAv2$ с использованием уже реализованной модели арифметики с фиксированной

запятой, прототипирование модели арифметики с плавающей запятой, а затем — прототипирование алгоритма $FSQRT$ в этой модели. Связанная задача — поиск примеров реальных платформ, соответствующих нашим моделям машинной арифметики.

Ближайшие теоретические задачи — исследование применимости интервальной математики [12] в контексте спецификации и верификации стандартных математических функций (на примере квадратного корня) и выбор метода и алгоритма спецификации и (сначала ручной) верификации стандартных тригонометрических функций sin и cos.

Благодарность: Дмитрию Юрьевичу Надёжину за техническую помощь с доказательством в системе ACL2 и за обсуждение материалов препринта [26].

Список литературы / References

[1] Кулямин В.В., “Стандартизация и тестирование реализаций математических функций, работающих с числами с плавающей точкой”, *Программирование*, **33**:3 (2007), 1–29; English transl.: Kuliamin V.V., “Standardization and testing of implementations of mathematical functions in floating point numbers”, *Programming and Computer Software*, **33**:3 (2007), 154–173.

[2] Никитин В.Ф., *Вариант вычисления квадратного корня (алгоритм Ньютона)*, http://algolist.manual.ru/maths/count_fast/sqrt.php; [Nikitin V.F., *Variant vychisleniya kvadratnogo kornya (algoritm Nyutona)*, http://algolist.manual.ru/maths/count_fast/sqrt.php, (in Russian).]

[3] Шелехов В.И., “Верификация и синтез эффективных программ стандартных функций floor, isqrt и ilog2 в технологии предикатного программирования”, *Проблемы управления и моделирования в сложных системах*, Труды 12-й межд. конф. (Самара, Самарский научный центр РАН), 2010, 622–630; [Shelekhov V.I., “Verifikatsiya i sintez effektivnykh programm standartnykh funktsiy floor, isqrt i ilog2 v tekhnologii predikatnogo programmirovaniya”, *Problemy upravleniya i modelirovaniya v slozhnykh sistemakh*, Trudy 12-y mezhd. konf. (Samara, Samarskiy nauchnyy tsentr RAN), 2010, 622–630, (in Russian).]

[4] Ayad A., Marché C., “Multi-prover verification of floating-point programs”, *Perspectives of System Informatics. PSI 2014*, Lecture Notes in Artificial Intelligence, **6173**, Springer, Berlin, Heidelberg, 2010, 127–141.

[5] Barret G., “Formal Methods Applied to a Floating-Point Number System”, *IEEE Trans. Softw. Eng.*, **15**:5 (1989), 611–621.

[6] Brain M. et al., “An Automatable Formal Semantics for IEEE-754 Floating-Point Arithmetic”, *Computer Arithmetic (ARITH ’15)*, Proc. of the 2015 IEEE 22nd Symposium (IEEE Computer Society), 2015, 160–167.

[7] El-Magdoub M.H., *Best Square Root Method – Algorithm – Function (Precision VS Speed)*, https://www.codeproject.com/Articles/69941/Best-Square-Root-Method-Algorithm-Function-Precisi.

[8] Ferguson W.E. et al., “Digit serial methods with applications to division and square root (with mechanically checked correctness proofs)”, 2017, 102–114, arXiv: arXiv:1708.00140.

[9] Gamboa R.A., *Square Roots in Acl2: a Study in Sonata Form*, Technical Report. University of Texas at Austin, USA, 1997.

[10] Грис Д., *Наука программирования*, Мир, М., 1986; in English: Gries D., *The Science of Programming*, Springer-Verlag, New York, 1981.

[11] Grohoski G., “Verifying Oracle’s SPARC Processors with ACL2 (Slides of the Invited talk)”, *The ACL2 Theorem Prover and Its Applications*, Proc. Int. Workshop, 2017,

http://www.cs.utexas.edu/users/moore/acl2/workshop-2017/slides-accepted/grohoski-ACL2_talk.pdf.

[12] Gutowski M.W., "Power and beauty of interval methods", arXiv: arXiv:physics/0302034.

[13] Harrison J., "A Machine-Checked Theory of Floating Point Arithmetic", Lecture Notes in Computer Science, **1690**, Springer, Berlin, Heidelberg, 1999, 113–130.

[14] Harrison J., "Formal Verification of Floating Point Trigonometric Functions", Lecture Notes in Computer Science, **1954**, Springer, Berlin, Heidelberg, 2000, 217–233.

[15] Harrison J., "Formal Verification of IA-64 Division Algorithms", Lecture Notes in Computer Science, **1869**, Springer, Berlin, Heidelberg, 2000, 233–251.

[16] Harrison J., "Floating Point Verification in HOL Light: The Exponential Function", *Formal Methods in System Design*, **16**:3 (2000), 271–305.

[17] Harrison J., "Formal Verification of Square Root Algorithms", *Formal Methods in System Design*, **22**:2 (2003), 143–153.

[18] Hoare C.A.R., "The Verifying Compiler: A Grand Challenge for Computing Research", Lecture Notes in Computer Science, **2890**, Springer, Berlin, Heidelberg, 2003, 1–12.

[19] Kuliamin V.V., "Standardization and Testing of Mathematical Functions in floating point numbers", Lecture Notes in Computer Science, **5947**, Springer, Berlin, Heidelberg, 2010, 257–268.

[20] Monniaux D., "The pitfalls of verifying floating-point computations", *ACM Transactions on Programming Languages and Systems*, **30**:3 (2008), 1–41.

[21] Muller J.-M., *Elementary Functions: Algorithms and Implementation*, Birkhaüser, 2005.

[22] Muller J.-M. et al., *Handbook of Floating-Point Arithmetic, 2nd eddition*, Birkhüser, 2018.

[23] Sawada J., Gamboa R., "Mechanical Verification of a Square Root Algorithm Using Taylor's Theorem", Lecture Notes in Computer Science, **2517**, Springer, Berlin, Heidelberg, 2002, 274–291.

[24] Shilov N.V., "On the need to specify and verify standard functions", *The Bulletin of the Novosibirsk Computing Center (Series: Computer Science)*, **38** (2015), 105–119.

[25] Shilov N.V., Promsky A.V., "On specification andd verification of standard mathematical functions", *Humanities and Science University Journal*, **19** (2016), 57–68.

[26] Shilov N.V. et al., "Towards platform-independent verification of the standard mathematical functions: the square root function", arXiv: arXiv:abs/1801.00969.

[27] Siddique U., Hasan O., "On the Formalization of Gamma Function in HOL", *J. Autom. Reason.*, **53**:4 (2014), 407–429.

[28] *C reference. Sqrt, sqrtf, sqrtl*, http://en.cppreference.com/w/c/numeric/math/sqrt.

[29] *IEEE 754-2008*, http://ieeexplore.ieee.org/document/4610935.

[30] *ISO/IEC/IEEE 60559:2011. Information technology -- Microprocessor Systems -- Floating-Point arithmetic*, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=57469.

[31] *Алгоритм нахождения корня n-ной степени*, https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%BD%D0%B0%D1%85%D0%BE%D0%B6%D0%B4%D0%B5%D0%BD%D0%B8%D1%8F_%D0%BA%D0%BE%D1%80%D0%BD%D1%8F_n-%D0%BD%D0%BE%D0%B9_%D1%81%D1%82%D0%B5%D0%BF%D0%B5%D0%BD%D0%B8; in English: *nth root algorithm*, https://en.wikipedia.org/wiki/Nth_root_algorithm.

[32] *"Роскосмос" назвал причину неудачного запуска с космодрома Восточный*, https://www.rbc.ru/politics/12/12/2017/5a2ebcd59a79479d29667115; [*"Roskosmos" nazval prichinu neudachnogo zapuska s kosmodroma Vostochnyy*, https://www.rbc.ru/politics/12/12/2017/5a2ebcd59a79479d29667115, (in Russian).]

Shilov N. V., Kondratyev D. A., Anureev I. S., Bodin E. V, Promsky A. V., "Platform-independent Specification and Verification of the Standard Mathematical Square Root Function", *Modeling and Analysis of Information Systems*, **25**:6 (2018), 637–666.

DOI: 10.18255/1818-1015-2018-6-637-666

Abstract. The project "Platform-independent approach to formal specification and verification of standard mathematical functions" is aimed onto the development of incremental combined approach to specification and verification of standard Mathematical functions like `sqrt`, `cos`, `sin`, etc. Platform-independence means that we attempt to design a relatively simple axiomatization of the computer arithmetics in terms of real arithmetics (i.e. the field $\mathbb{R}$ of real numbers) but do not specify neither base of the computer arithmetics, nor a format of numbers representation. Incrementality means that we start with the most straightforward specification of the simplest case to verify the algorithm in real numbers and finish with a realistic specification and a verification of the algorithm in computer arithmetics. We call our approach combined because we start with manual (pen-and-paper) verification of the algorithm in real numbers, then use this verification as proof-outlines for a manual verification of the algorithm in computer arithmetics, and finish with a computer-aided validation of the manual proofs with a proof-assistant system (to avoid appeals to "obviousness" that are common in human-carried proofs). In the paper, we apply our platform-independent incremental combined approach to specification and verification of the standard Mathematical square root function. Currently a computer-aided validation was carried for correctness (consistency) of our fix-point arithmetics and for the existence of a look-up table with the initial approximations of the square roots for fix-point numbers.

Keywords: fix-point numbers, floating-point numbers, computer arithmetic, formal verification, partial and total correctness, Hoare triples, Floyd verification method of inductive assertions, exact function, square root function, Newton–Raphson method, look-up table

On the authors:
Nikolay V. Shilov, orcid.org/0000-0001-7515-9647, PhD
Autonomous noncommercial organization of higher education "Innopolis University",
1 Universitetskaya str., Innopolis, Tatarstan Republic, 420500, Russia, e-mail: shiloviis@mail.ru

Dmitry A. Kondratyev, orcid.org/0000-0002-9387-6735, postgraduate student
A.P. Ershov Institute of Informatics Systems,
6, Acad. Lavrentjev pr., Novosibirsk 630090, Russia, e-mail: apple-66@mail.ru

Igor S. Anureev, orcid.org/0000-0001-9574-128X, PhD, senior researcher
A.P. Ershov Institute of Informatics Systems,
6, Acad. Lavrentjev pr., Novosibirsk 630090, Russia, e-mail: anureev@iis.nsk.su

Eugene V. Bodin, orcid.org/0000-0002-5882-0365, researcher
A.P. Ershov Institute of Informatics Systems,
6, Acad. Lavrentjev pr., Novosibirsk 630090, Russia, e-mail: bodin@iis.nsk.su

Alexei V. Promsky, orcid.org/0000-0002-5963-2390, PhD, scientific secretary
A.P. Ershov Institute of Informatics Systems,
6, Acad. Lavrentjev pr., Novosibirsk 630090, Russia, e-mail: promsky@iis.nsk.su

Acknowledgments:
The research has been supported by Russian Foundation for Basic Research (grant 17-01-00789).