С.П. Ковалёв

ПРИМЕНЕНИЕ ЛОГИКИ ЛУКАСЕВИЧА ДЛЯ РАЗРАБОТКИ АЛГОРИТМОВ

Abstract. *Practical problems associated with engineering efficient robust algorithms for real world computers lay beyond the traditional scope of mathematical theory of algorithms. Special mathematical methods are required to formally reflect and verify empirical approaches routinely used by technicians. Such methods based on model theory and multiple-valued logics are presented here. Specifically, it is proven that Łukasiewicz logic language is capable to express operations used in computer implementations of integral arithmetic. This allows proposing novel regard of the nature of Łukasiewicz logic.*

Введение

Важнейшим средством решения задач в различных областях знания в настоящее время служит компьютер. Для того чтобы машина смогла решить задачу, решение следует представить в виде алгоритма – явно заданного пошагового процесса преобразования исходных данных в искомый объект. Однако не для всякого объекта существует алгоритм его построения. По существу, представлению в форме алгоритмов не поддаются конструкции, уровень сложности которых качественно превосходит арифметические вычисления в натуральных числах. Поэтому объект, который можно построить при помощи некоторого алгоритма, называется вычислимым или рекурсивным.

Методы выявления вычислимых объектов разрабатываются в рамках специального раздела математики – теории алгоритмов. Предлагается ряд эквивалентных математических моделей алгоритмов, в том числе допускающих представление в виде (абстрактных) технических устройств. Ярким примером служит машина Тьюринга – автомат, перерабатывающий последовательности символов неограниченного размера согласно заданной программе, реализующей алгоритм. Состояние автомата характеризуется значением q и номером n текущего символа последовательности S. Команды, из которых состоит программа, записываются в виде $qa \to pb\varepsilon$, что означает возможность перехода из состояния q при условии, что $S_n = a$, в состояние p с заменой значения S_n на b и последующим перемещением от n-го символа к $n + \varepsilon$-му. Допускается перемещение не более чем на один шаг, т. е. $\varepsilon \in \{0, +1, -1\}$.

Такую машину можно представить в виде головки, движущейся вдоль бесконечной ленты, на которой записаны символы перерабатываемой последовательности.

Однако доказательство вычислимости некоторого объекта и даже предъявление алгоритма его построения вовсе не гарантирует практической отдачи от его реализации на компьютере. Дело в том, что реальные вычислительные устройства основываются на совершенно иных концепциях, чем математические модели теории алгоритмов. Для представления чисел и операций используются характеристики разнообразных физических и технических объектов, выходящих за рамки чистой математики. Например, в настоящее время широко распространены электронные устройства, реализующие вычислительный процесс путем преобразования значений электрического напряжения. Необходим специальный математический аппарат, позволяющий адекватно учитывать особенности устройств в целях создания алгоритмов, выдающих практически полезные результаты при выполнении на них. По существу, такой аппарат должен обеспечивать адекватное формальное выражение хода конструкторской мысли разработчика алгоритма, языка программирования или вычислительного устройства. В настоящей статье представлен подход к формированию такого аппарата, основанный на теории моделей и многозначной логике Лукасевича.

По сравнению с теорией алгоритмов предлагаемый аппарат обладает определенной громоздкостью. Это вызвано сложностью технических закономерностей, эмпирически нащупанных проектировщиками вычислительных машин. Подобная ситуация встречается и в других областях знания, сочетающих высокоразвитую теорию с техническими приложениями. Например, теоретическим фундаментом механики служат вариационные принципы Лагранжа и Гамильтона. Они позволяют строить и анализировать абстрактные модели механических явлений, используя изящный и мощный математический аппарат. Однако они не могут непосредственно применяться для решения конкретных технических проблем, например, расчета прочности зданий. Проблемы такого рода изучаются в рамках специальных технических дисциплин, таких как сопротивление материалов. Эти дисциплины предлагают громоздкие математические методы, обоснованные зачастую лишь частными эмпирическими наблюдениями.

Практические проблемы разработки алгоритмов

Итак, хотя теория алгоритмов предоставляет математический фундамент для их разработки, практическое применение ее результатов наталкивается на принципиальные ограничения. В частности, значительные затруднения связаны с ограниченностью объема ресурсов, которыми оснащены вычислительные машины. Например, множество натуральных чисел тривиально вычислимо – его характеристической функцией служит постоянная 1. Однако на практике вычислить все его элементы невозможно: этот процесс займет бесконечное время и потребует бесконечного количества ячеек памяти. Так что вычислимость объекта вовсе не гарантирует возможности представить его средствами реального компьютера. Более того, компьютер способен вычислять только функции, заданные на конечных множествах. Они образуют довольно бедный подкласс класса рекурсивных функций. В связи с этим программисты вкладывают в понятие «рекурсивная функция» совершенно другой смысл, чем установленный в теории: они называют так функцию, вызывающую саму себя в ходе выполнения. Здесь можно усмотреть не более чем условную аналогию с формой записи правила примитивной рекурсии, задающего зависимость значения функции $f(x + 1)$ от значения $f(x)$.

Ограничения на объем ресурсов памяти, доступных для представления чисел, приводят к тому, что результаты выполнения арифметических операций на компьютере могут отличаться от ожидаемых. Такие различия проявляются в форме всевозможных переполнений, переносов, ошибок округления. Например, современные персональные компьютеры, как правило, поддерживают только 2^{32} различных числовых значений. Поэтому, если в переменную i, имеющую целочисленный тип (int), поместить значение 65536 (т. е. 2^{16}), то произведение i*i будет равняться 0. При этом будет установлен специальный статусный флаг, сигнализирующий о переполнении. В то же время если результаты операций не выходят за пределы диапазона поддерживаемых чисел, то никаких отличий быть не должно. Так, если в приведенном примере значение i равно 2, то выражение i*i, разумеется, равняется 4. При реализации алгоритмов необходимо уметь определять, может ли произойти переполнение на очередном шаге, и задавать действия по его обработке.

Как известно, аппарат теории алгоритмов позволяет не только выявлять вычислимые задачи, но и оценивать их вычислительную сложность. С точки зрения теории, сложность алгоритма – это минимальное количество актов применения элементарных правил

вычисления (например, шагов головки машины Тьюринга), требуемых для его выполнения. Обычно сложность задают как функцию от длины входной последовательности машины Тьюринга, т.е. от величины входных параметров задачи. Известно, что чем выше теоретическая сложность алгоритма, тем больше ресурсов (в первую очередь времени и памяти) следует отводить под выполнение его программной реализации. Однако получать конкретные количественные оценки эффективности решения задачи на реальном компьютере обычно не удается. Положение усугубляется тем, что для практически важных алгоритмов имеются лишь асимптотические оценки сложности, записываемые при помощи O-символов. Например, алгоритм бинарного поиска в массиве из n элементов имеет сложность $O(\log_2 n)$. Это означает, что его сложность выражается формулой $C \log_2 n$, причем константа C определяется правилом сравнения элементов и может иметь очень большое значение. Здесь показательным является пример одного алгоритма из теории графов, сложность которого зависит от числа вершин n как $10^{150}n^3$ [1]. Кубическая зависимость от параметра задачи считается признаком невысокого уровня сложности, однако величина постоянного множителя делает бессмысленной практическую реализацию такого алгоритма.

Чтобы получить практически значимые оценки эффективности алгоритма, необходимо принять во внимание архитектуру компьютера, на котором он реализуется. Дело в том, что в основе функционирования арифметических модулей компьютеров лежат совершенно иные физические процессы, чем те, которыми моделируется машина Тьюринга. Поэтому операции, имеющие значительную вычислительную сложность, могут быть реализованы при помощи быстрых и компактных устройств. В свою очередь, выполнение некоторых операций, элементарных с точки зрения теории алгоритмов, обычно требует значительного расхода ресурсов. В качестве примера рассмотрим многопроцессорную сеть, состоящую из большого количества арифметических устройств. Она способна быстро выполнять параллельные переборные алгоритмы, сложность которых экспоненциально зависит от количества перебираемых элементов. В то же время линейный по сложности, но строго последовательный алгоритм итеративного вычисления примитивно рекурсивной функции будет выполняться медленно и неэффективно, поскольку на каждом шаге работать будет только один процессор сети.

Большое значение имеет также «обратная задача» оценки сложности. Она состоит в построении такого алгоритма решения данной вычислительной задачи, реализация которого была бы

оптимальной для данной архитектуры. Такая постановка известна как проблема отображения вычислительных задач на архитектуру компьютерных систем [2]. Теория алгоритмов здесь практически бессильна, поскольку она не предлагает математических средств для учета архитектурных особенностей компьютеров. В частности, требуются формальные языки для записи реализации вычислительного процесса, способы проверки его корректности, методы оценки потребляемых им объемов времени и памяти.

В последнее время большую актуальность приобрела задача оптимизации алгоритмов с точки зрения количества электроэнергии, затрачиваемого на его выполнение. Она вызвана распространением мобильных и переносных компьютерных устройств, емкость автономного электропитания которых очень ограничена в силу физико-химических закономерностей. Архитектура этих устройств такова, что действия, имеющие сходную структуру и одинаковые показатели эффективности, могут радикально отличаться по энергопотреблению. Например, при модификации числового значения, хранящегося в двоичной памяти, энергия затрачивается на переключение значения двоичного разряда из 0 в 1 и обратно. Предположим, что в памяти хранится число 15 (в двоичном представлении 1111). Его увеличение на единицу дает 16 (в двоичном представлении 10000), что требует переключения значений 5 разрядов. Увеличение еще на единицу дает 17 (в двоичном представлении 10001), изменяет всего один разряд и поэтому оказывается в 5 раз экономичнее.

В целях решения перечисленных проблем необходимо построить математическую модель компьютерной реализации арифметики. При этом очень важно определить адекватный уровень абстракции. С одной стороны, модель не должна быть привязанной к конкретным техническим решениям и физическим принципам, иначе она не будет обладать универсальностью и простотой. С другой стороны, она должна отражать ключевые особенности реализации арифметики при помощи доступных нам технических средств. Исходя из изложенного выше, мы выделяем следующие особенности такого рода:

(1) функционирование компьютера сводится к выполнению арифметических операций над числовыми кодами различных информационных объектов (самих чисел, текстов и т. д.);
(2) все допустимые числовые коды образуют конечное множество;
(3) существуют средства для обнаружения переполнения в ходе выполнения вычислений;

(4) все возможные элементарные операции, из которых составляются вычисления, образуют фиксированное конечное множество;
(5) скорость выполнения вычислений определяется правилами развертывания вычислительного процесса вдоль стрелы реального времени;
(6) необходима техника доказательства правильности реализации вычислительных алгоритмов.

Частичная интерпретация

Хорошо известно, что арифметика может быть формализована как теория, т. е. множество предложений языка первого порядка. Поэтому естественным кандидатом на роль математического объекта, способного служить моделью ее реализации, является алгебраическая система – стандартная семантическая конструкция первого порядка. В силу указанных выше требований (2) и (4) она должна иметь конечный универсум (основное множество) и конечную сигнатуру. Отсюда следует, что на ней не могут быть истинными все аксиомы арифметики, поскольку всякая модель последней бесконечна. Машинная реализация арифметики способна верифицировать только такие предложения арифметической теории, которые характеризуют поведение чисел из множества, поддерживаемого компьютером. Необходим математический метод построения алгебраических систем, на которых эти предложения истинны, и оценки их «близости» к стандартной модели арифметики.

В теоретическом программировании алгебраическая система, описывающая поведение информационных объектов определенного типа, называются абстрактным типом данных (АТД) [3]. Числовые АТД обычно строятся *ad hoc*, например, посредством той или иной модификации аксиом арифметики. Однако подобная модификация заставляет заново доказывать основные метатеоремы непротиворечивости, адекватности и т. д. Кроме того, аксиоматические теории, как правило, очень неустойчивы: небольшое изменение аксиом может привести к кардинальному изменению свойств теории. Поэтому следует модифицировать не саму теорию, а способ построения ее модели. Необходимая модификация теоретико-модельного подхода предложена автором в работе [4]. Она названа частичной интерпретацией теории первого порядка T и заключается в построении алгебраической системы, которая должна верифицировать только такие предложения теории T, для которых любые входящие в них термы могут быть заменены

константами из заданной конечной подсигнатуры ее сигнатуры. Таким образом, формальной моделью ресурса, доступного для представления объектов теории T, является явно заданное множество константных символов. Вне этого множества даже стандартные аксиомы равенства могут не выполняться.

Точные определения этих конструкций выглядят следующим образом.

Определение 1. Будем рассматривать языки первого порядка без равенства. Пусть σ – сигнатура, содержащая символ равенства $\equiv$, T – теория сигнатуры σ, содержащая стандартные аксиомы равенства, σ_0 – конечная подсигнатура сигнатуры σ. *Проекцией* $T \downarrow \sigma_0$ называется множество таких бескванторных предложений φ сигнатуры σ_0, выводимых из T, что для любого терма t, входящего в φ, существует константный символ $c \in \sigma_0$, такой, что предложение $t \equiv c$ выводимо из T. *Частичной интерпретацией* теории T в сигнатуре σ_0 называется пара $<A, \sigma_0>$, где A – алгебраическая система сигнатуры $\sigma(A) \supseteq \sigma_0$, являющаяся моделью проекции $T \downarrow \sigma_0$.

Определение 2. Пусть $C(\sigma_0) = \{c_1, \ldots, c_n\}$ – множество константных символов сигнатуры σ_0. *Релятивизацией* формулы φ на σ_0 называется формула $\varphi \uparrow \sigma_0$, получающаяся из φ заменой всех подформул вида $\forall x_0\, \psi(x_0, x_1, \ldots, x_k)$ конъюнкциями $\bigwedge_{i=1, \ldots, n} \psi(c_i, x_1, \ldots, x_k)$ и вида $\exists x_0\, \psi(x_0, x_1, \ldots, x_k)$ – дизъюнкциями $\bigvee_{i=1, \ldots, n} \psi(c_i, x_1, \ldots, x_k)$. Будем говорить, что пара $<A, \sigma_0>$ *поддерживает* предложение φ, если предложение $\varphi \uparrow \sigma_0$ выполняется в алгебраической системе A.

В качестве математических моделей компьютерной реализации арифметики выступают частичные интерпретации арифметической теории с универсумом $E_{n+1} = \{0, 1, \ldots, n\}$, причем значение n кодирует флаг переполнения. Степень их сходства со стандартной моделью арифметики определяется способностью поддерживать (в смысле определения 2) основные арифметические аксиомы, в частности схему индукции и инъективность функции прибавления единицы s. Для случая арифметики целых чисел, как показано в работе [4], наиболее адекватными являются следующие частичные интерпретации.

Определение 3.

(i) *Начальным отрезком множества неотрицательных целых чисел с переполнением* называется алгебраическая система

$$\mathrm{OA}_{n+1} = <E_{n+1}, 0, 1, \ldots, n-1, =, |+|, |-|, |\times|>,$$
$$x\ |+|\ y = \min(n, x + y),$$
$$x\ |-|\ y = \max(0, x - y),$$

$x \,|\times|\, y = \min(n, xy)$.

(ii) *(Почти) симметричным отрезком целых чисел по модулю* называется алгебраическая система

$$\begin{aligned}
&\mathrm{MA}_{n+1} = \langle E_{n+1}, 0, 1, \ldots, [(n-1)/2], -[n/2], \ldots, -1,\\
&\qquad (=), (+), (-), (\times), \mathrm{Carry}\rangle,\\
&x\,(=)\,y = (x = y) \vee (x, y) \in \{(0, n), (n, 0)\},\\
&x\,(+)\,y = (x + y) \bmod n,\\
&\quad (-)\,x = n - x,\\
&x\,(\times)\,y = xy \bmod n,\\
&\mathrm{Carry}(x) = (x = n).
\end{aligned}$$

Арифметика вещественных чисел строится путем их аппроксимации парами целых чисел: целой и дробной частью либо мантиссой и порядком. Операции над ними представляют собой определенные модификации операций из определения 3. Их рассмотрение выходит за рамки настоящей работы, оно приведено в работе [4].

Аппарат логики Лукасевича

Естественным инструментом исследования свойств операций, заданных на конечных множествах, является многозначная логика. Действительно, универсум можно рассматривать как совокупность логических констант, а сигнатурные функции – как базисные логические связки. Как показано автором настоящей работы в статье [5], адекватный аппарат для моделирования компьютерной реализации арифметики предоставляет логика Лукасевича $Ł_{n+1}$. Следует отметить, что она функционально неполна – существуют логические функции, не выразимые через ее связки. Именно из-за неполноты она обычно не применяется для анализа арифметики. Однако в силу теоремы Эванса - Шварца [6] неполнота устраняется путем добавления самих чисел (константных функций), так что логика Лукасевича позволяет выявлять структурные свойства арифметических операций, не зависящие от значений их аргументов. Кроме того, не выразимыми оказываются именно функции, выходящие за рамки арифметики целых чисел как таковой (например, деление).

Для целей настоящей статьи многозначную логику следует представить в виде матрицы – алгебраической системы с универсумом E_{n+1} [7]. Сигнатура логической матрицы L состоит из некоторого набора функциональных символов, обозначающих связки, и одноместного предиката D, выделяющего истинностные значения данной логики. Тавтологиям логики отвечают такие и только

такие термы $t(x_1, \ldots, x_k)$, что формула $D(t(x_1, \ldots, x_k))$ выполняется на L. Логике Лукасевича соответствует следующая матрица:

$$Ł_{n+1} = \langle E_{n+1}, \sim, \to, \{n\}\rangle,$$
$$\sim x = n - x,$$
$$x \to y = \min(n, n - x + y).$$

Через ее связки выражаются дизъюнкция и конъюнкция многозначной логики:

$$x \vee y = \max(x, y) = (x \to y) \to y,$$
$$x \wedge y = \min(x, y) = \sim(\sim x \vee \sim y) = \sim(x \to \sim(x \to y)).$$

Рассмотрим свойства логики Лукасевича как класса функций на E_{n+1}, замкнутого относительно суперпозиции. Обозначим через P_{n+1} класс всех функций на E_{n+1}, образующий логику Поста. Для произвольного непустого подмножества $X \subset E_{n+1}$ положим

$$C^X_{n+1} = \{f \in P_{n+1} \mid f(X, \ldots, X) \subseteq X\},$$
$$D^X_{n+1} = \{f \in P_{n+1} \mid f(E_{n+1}, \ldots, E_{n+1}) \subseteq X\}.$$

Класс C^X_{n+1} является *предполным* в P_{n+1}, т.е. он замкнут и замыкание его объединения с любой функцией, не выразимой в нем, равно P_{n+1} [8]. Как установили Эванс и Шварц, логика Лукасевича является *слабо полным* классом [8], т.е. в результате его объединения с множеством всех константных функций на E_{n+1} получается система функций, полная в P_{n+1}. В частности, с одной стороны, $Ł_{n+1}$ содержится в предполном классе $C^{\{0, n\}}_{n+1}$, причем совпадает с ним тогда и только тогда, когда n является простым числом. С другой стороны, $Ł_{n+1}$ содержит класс $D^{\{0, n\}}_{n+1}$, причем не совпадает с ним при $n > 1$ (здесь мы считаем, что это условие выполнено). Отметим, что существует взаимно однозначное соответствие между функциями из $D^{\{0, n\}}_{n+1}$ и предикатами на E_{n+1}, задающими арифметические отношения. Они строятся при помощи дизъюнкции и конъюнкции из функций

$$J_i(x) = \begin{cases} n, x = i, \\ 0, x \neq i. \end{cases}$$

Применим аппарат логики Лукасевича для анализа свойств операций компьютерных реализаций арифметики, описанных в определении 3. Здесь получены следующие результаты.

Предложение 1. *Выражение, вычисление которого задается формулой A, не дает переполнения тогда и только тогда, когда* $\sim J_n(A)$ *– тавтология* $Ł_{n+1}$.

Лемма 1. *Имеют место следующие равенства.*

(i) $(x = y) = J_n((x \to y) \wedge (y \to x))$;
(ii) $x \,|+|\, y = {\sim}x \to y$;
(iii) $x \,|-|\, y = {\sim}(x \to y)$;
(iv) $x \,|\times|\, y = \bigvee_{i \in E_{n+1}} ({\sim}x \to^i 0) \wedge J_i(y)$.

Теорема 1.
(i) *Операции* $=, |+|, |-|, |\times|$ *выразимы в* $Ł_{n+1}$.
(ii) *Система функций* $\{|+|, |-|\}$ *образует базис в классе* $Ł_{n+1} \cap C^{\{0\}}{}_{n+1}$, *предполном в* $Ł_{n+1}$.
(iii) *Системы функций* $\{{\sim}, |+|\}$, $\{n, |-|\}$, $OAŁ_{n+1} = \{=, |-|\}$ *являются базисами в* $Ł_{n+1}$.

Следствие 1.1. *Система констант и функций арифметики с переполнением* OA_{n+1} *полна в* P_{n+1}.

Лемма 2. *Имеют место следующие равенства.*
(i) $x \bmod n = x \,|-|\, J_n(x)$;
(ii) $x \,(+)\, y = ((x \,|+|\, y) \bmod n) \,|+|\, ((x \,|-|\, (n \,|-|\, y)) \bmod n)$;
(iii) $(-)x = n \,|-|\, x$;
(iv) $x \,(\times)\, y = \bigvee_{i \in E_{n+1}} (0 \,(+)^i\, x) \wedge J_i(y)$;
(v) $\mathrm{Carry}(x) = J_n(x)$;
(vi) $x \,(=)\, y = \mathrm{Carry}((-)(x \,(+)\, ((-)y)))$.

Обозначим через M_{n+1} класс функций, сохраняющих отношение равенства по модулю (=) системы MA_{n+1}. Известно [8], что класс M_{n+1} является предполным, но не слабо полным. Обозначим через $M^{\Lambda}{}_{n+1}$ класс функций, сохраняющих четырехместное отношение

$$\Lambda_{n+1} = \{\langle x, x, x, x\rangle \mid x \in E_{n+1}\} \cup \left(\{0, n\}^4 \cap \{\langle x_1, x_2, x_3, x_4\rangle \mid x_1 + x_2 = x_3 + x_4 \,(\mathrm{mod}\, 2n)\}\right).$$

Далее, положим

$$MP_{n+1} = \{f \in P_{n+1} \mid f(x_1, \ldots, x_{s-1}, 0, x_{s+1}, \ldots, x_k) = f(x_1, \ldots, x_{s-1}, n, x_{s+1}, \ldots, x_k) \text{ для всех } s = 1, \ldots, k\},$$
$$J_{n+1} = \{id, {\sim}, J_n, J_0, J_0J_n, J_0J_0\},$$
$$R_{n+1} = (MP_{n+1} \cap (D^{\{0, n\}}{}_{n+1} \cup D^{E_{n+1}\setminus\{n\}}{}_{n+1} \cup D^{E_{n+1}\setminus\{0\}}{}_{n+1})) \cup J_{n+1}.$$

Все эти классы замкнуты, причем $R_{n+1} \subset M^{\Lambda}{}_{n+1} \subset M_{n+1}$.

Теорема 2.
(i) *Операции* (=), (+), (–), (×), Carry *выразимы в* $Ł_{n+1}$.
(ii) *Система функций* $\{(+), (-), (\times), \mathrm{Carry}\}$ *образует базис в классе* $MAC_{n+1} \subseteq Ł_{n+1} \cap R_{n+1} \subset Ł_{n+1} \cap M^{\Lambda}{}_{n+1} \subset Ł_{n+1} \cap M_{n+1} \subset Ł_{n+1}$, *причем* MAC_{n+1} *совпадает с* $Ł_{n+1} \cap R_{n+1}$ *тогда и только тогда, когда n является простым числом.*
(iii) *Система функций*

$$\text{МАЛ}_{n+1} = \begin{cases} \{(-), \text{Carry}, \vee\}, & n=2, \\ \{(+), (-), \text{Carry}, \vee\}, & n>2 \end{cases}$$

образует базис в $Ł_{n+1}$.

Следствие 2.1. *Система констант и функций арифметики по модулю* МА_{n+1} *неполна и не предполна в* P_{n+1}*, но становится полной при обогащении функцией* $\max(x, y)$*,* $(x, y) \in E_{n+1} \times E_{n+1}$.

Логический анализ алгоритмов

Покажем, каким образом представленный аппарат применятся при решении задач, описанных в разделе 1. В частности, он позволяет строить, преобразовывать и проверять логические модели алгоритмов, имеющие следующую форму. Арифметические и логические операции записываются явными выражениями в виде суперпозиций базисных логических связок. Управляющие конструкции (по возможности) записываются в форме дизъюнктов, что позволяет придать модели структурное сходство с однородным потоком вычислений. Например, цикл, вычисляющий функцию $F(x, y)$ путем последовательного применения y итераций функции $f(x)$, задается дизъюнктом вида $\bigvee_{i \in E_{n+1}} f^i(x) \wedge J_i(y)$ (ср. утверждения (iv) лемм 1 и 2).

Такое представление алгоритмов обладает высокой степенью гибкости и адаптивности с точки зрения отображения на архитектуру различных компьютеров. При выполнении отображения машинные команды целевой архитектуры также представляются словами языка многозначной логики. Это позволяет выполнить абстрактный аналог процедуры трансляции путем сопоставления подвыражений. Правильность выполнения этой процедуры можно доказать строго, привлекая технику доказательства логики Лукасевича (в том числе автоматизированную [9]).

К такой модели можно применить традиционные процедуры оптимизации: удаление повторяющихся подвыражений, необязательных промежуточных переменных, мертвого кода и т.д. В результате получатся, по существу, абстрактные инварианты поведения алгоритмов, для которых разработаны методы оценки производительности, не зависящие от выбора целевой аппаратуры [10]. В основе этих методов лежит трактовка базовых конструкций (в данном случае элементов синтаксиса языка многозначной логики) как обращений к некоторому логическому вычислительному ресурсу. Использование этого ресурса имеет определенную стоимость (меру вычислительной работы), отражающую показатели его эффективности.

Универсальность, присущая языку многозначной логики, позволяет использовать его при отображении алгоритмов на различные классы машинных архитектур. Так, подавляющее большинство современных компьютеров ориентированы на представление чисел в позиционной системе счисления по основанию 2. Основной моделью вычислений, реализованной в современных аппаратных средствах и воспроизводимой языками программирования, является арифметика MA_{2^q+1} (обычно q равно 16, 32 или 64). Однако предикат Carry (флаг переноса) не имеет программной реализации, что усложняет контроль целочисленного переполнения. Еще большую проблему составляет функциональная неполнота системы MA_{n+1}, приводящая к потребности в таких неэффективных операциях, как условный переход (см. следствие 2.1). Любопытную альтернативу представляют многозначные арифметические устройства, например, аппаратная реализация операции $|-|$ арифметики OA_{n+1} на токопроводящих полимерах [11]. В силу утверждения (iii) теоремы 1 этой операции достаточно для построения полнофункционального арифметического устройства.

Заключение

Основываясь на представленном подходе к построению машинной арифметики, мы хотели бы предложить новую интерпретацию природы логики Лукасевича. А именно, поскольку логика Лукасевича предоставляет адекватный формальный язык для описания инструментов решения вычислительных задач, можно заключить, что она определяет формы и ограничения, лишь в рамках которых конечные информационные системы способны работать с таким инфинитным объектом, как множество целых чисел. Логика Лукасевича оказывается своего рода «дистиллятом» тех явлений информационной реальности, которые порождают брауэровскую «изначальную интуицию натурального числа».

В частности, трактовка выделенного истинностного значения n как переполнения означает, что класс тавтологий $Ł_{n+1}$ содержит в точности такие арифметические термы, значения которых не могут быть вычислены в машинной арифметике, поддерживающей n чисел. Согласно предложению 1, этот факт открывает путь к использованию техники доказательств логики Лукасевича в задачах верификации вычислительных программ на предмет отсутствия переполнения. Отметим также, что слабость логики как класса теорем оборачивается ее выразительной силой как класса числовых функций – с ростом n выражений с неопределимым значением становится все меньше, так что $Ł_{n+1}$ как класс теорем ослабевает

(например, согласно условию Линденбаума [7], $Ł_{kn+1} \subset Ł_{n+1}$ при всяком $k > 1$).

ЛИТЕРАТУРА

1. *Demetrescu C., Finocchi I., Italiano G.F.* Algorithm engineering // Bull. EATCS. 2003. Vol. 79. P. 48-63.
2. *Воеводин В.В.* Отображение проблем вычислительной математики на архитектуру вычислительных систем // Вычислительные методы и программирование. 2000. Т.1. С. 37-44.
3. *Liskov B.H., Zilles S.N.* Specification techniques for data abstractions // IEEE Trans. on Software Engineering. 1975. SE-1, 1. P. 7-19.
4. *Ковалёв С.П.* Аналитические модели машинной арифметики // Сибирский журнал индустриальной математики. 2003. Т. 6, №3. С. 88-102.
5. *Ковалёв С.П.* Логика Лукасевича как архитектурная модель арифметики // Сибирский журнал индустриальной математики. 2003. Т. 6, № 4. С. 32-50.
6. *Evans T., Schwartz P.B.* On Slupecki T-functions // J. Symbolic Logic. 1958. Vol. 23. P. 267-270.
7. *Карпенко А.С.* Логики Лукасевича и простые числа. М.: Наука, 2000.
8. *Rosenberg I.G.* Completeness properties of multiple-valued logic algebras // Computer Science and Multiple-Valued Logic. Amsterdam: North Holland, 1977. P. 144-186.
9. *Beavers G.* Automated theorem proving for Łukasiewicz logics // Studia Logica. 1993. Vol. 52, N 2. P. 183-195.
10. *Смелянский Р.Л.* Методы анализа и оценки производительности вычислительных систем. М.: МГУ, 1990.
11. *Mills J.W.* Polymer Processors. Indiana University, Computer Science Dept, Technical Report TR580. Indiana University, 2003. http://www.cs.indiana.edu/pub/techreports/TR580.pdf.