ISSN 2658-5014 (Print) СИИТ ISSN 2686-7044 (Online)
http://siit.ugatu.su 2026. Т. 8, № 2(26)
СИСТЕМНАЯ ИНЖЕНЕРИЯ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ
Научная статья

УДК 510.649 DOI: 10.54708/SIIT-2026-no2-p132 EDN: PEVKBH

Тестирование в качестве доказательства разрешимости конструктивных задач

М. Джудакизаде • А. П. Бельтюков

Удмуртский государственный университет

Предлагается подход к решению конструктивных задач методом построения конструктивных доказательств их разрешимости, при котором доказательство оказывается программой, а его проверка – тестированием этой программы. При этом исчерпывающее тестирование проводится за конечное время. Более того, структура тестового комплекса автоматически выводится из структуры программы. Метод сочетает подходы Д. Гильберта, Ж. Эрбрана и Г. Генцена и является развитием трёхуровневой реализации логических формул. Для полного набора логических связок и кванторов позитивной логики предикатов без функций вводятся правила тестирования секвенций в нотации $f{:}(\Gamma{\to}B){:}f'$, где f – обосновывающая конструкция (реализация), f' – тестирующая конструкция. Для каждого правила явно описывается операционная семантика соответствующей программы.

Конструктивная логика; синтез программ; исчисление секвенций; реализуемость; универсум Эрбрана; тестирование; трёхуровневая реализация; дедуктивный синтез; временная сложность.

Введение

Цель настоящей работы – предложить способ проверки программ, синтезированных с помощью конструктивного логического вывода. В этом случае программа алгоритмически извлекается из постановки задачи, записываемой в виде конструктивно понимаемой логической формулы. Обычно полномасштабным тестированием программ занимаются в случае их ручного создания. Если программа создана автоматически надёжным логическим выводом из строгой постановки решаемой задачи, то, казалось бы, тестирование не нужно: программа и так должна решать поставленную задачу. На самом деле это не так: в постановке задачи могут быть ошибки, программу могут пытаться применить в условиях, не соответствующих постановке. Если это происходит при реальной практической эксплуатации, то мы рекомендуем применять специальную процедуру «разбора полётов» [Jou24]. Но иногда нужно убедиться в правильности работы программы заранее, чтобы исключить ошибки в постановке задачи. В этой статье предлагается такой подход, при котором тестирование можно провести, не только не зная конкретных данных, но и не уточнив ещё даже конкретизации использованных абстрактных типов значений. Заодно это даёт план тестирования синтезированной программы в период опытной эксплуатации, чтобы исключить возможное некорректное описание предметной области в постановке задачи. На самом деле это приводит к тому, что вместо того, чтобы выполнять формальную проверку доказательства того, что поставленная задача разрешима, мы можем протестировать работу этой программы на некотором идеальном универсуме. Если тестирование не выявит ошибок, то это значит, что программа действительно

Рекомендовано к публикации программным комитетом XI Международной научной конференции ITIDS’2025 «Информационные технологии интеллектуальной поддержки принятия решений», Уфа, 13–15 ноября 2025 г.

Джудакизаде М., Бельтюков А. П. Тестирование в качестве доказательства разрешимости конструктивных задач // СИИТ. 2026. Т. 8, № 2(26). С. 132-143. DOI: 10.54708/SIIT-2026-no2-p132. EDN: PEVKBH.

Joudakizade M., Beltiukov A. P. “Testing as a proof of solvability of constructive problems” // SIIT. 2026. Vol. 8, no. 2(26), pp. 132-143. DOI: 10.54708/SIIT-2026-no2-p132. EDN: PEVKBH. (In Russian).

решает поставленную задачу. Вместе с тем этот процесс тестирования может служить прототипом проверки программы при опытной эксплуатации. Далее это тестирование объясняется подробнее.

Если мы хотим представить существование некоторой конструкции как подтверждение некоторого утверждения, то можно идти (по крайней мере) двумя путями:

1) построение формального доказательства (условно говоря, это – путь Давида Гильберта [Hil99]);

2) построение подтверждающего примера (условно говоря, это – путь Жака Эрбрана [Her30]).

Мы рассматриваем оба пути. Если с первым всё более-менее понятно, то для второго далее написано некоторое разъяснение.

Жак Эрбран для обоснования утверждений на абстрактном уровне предлагал строить некоторую искусственную предметную область – «Универсум Эрбрана» [Her30]. Для наших языков, рассуждений и их интерпретаций мы можем также породить свой универсум. Поскольку, в отличие от Эрбрана, в наших языках не используются предметные функции, то в отношении предметов этот универсум будет прост: он состоит из абстрактных имён. С другой стороны, в нашей конструктивной логике появляются реализации и реализационные конструкции [Kle45; Kle52]. Со структурными конструкциями всё ясно: достаточно воспользоваться операцией упорядоченной пары и двумя метками (0 и 1). Для абстрактных реализационных конструкций истинных постоянных атомов предлагается использовать сами эти атомы. Функциональные конструкции – абстрактные функции, реализующие заданные истинные импликативные и универсальные (постоянные) формулы, – записываются в виде самих этих формул, снабжённых некоторыми указателями конкретных вариантов исполнения (описанных далее).

Например, если f – реализация формулы $(D \Rightarrow (B \vee C))$, где D, B и C имеют по одной возможной реализации, то можно считать, что

$$f = f_0 = (D \Rightarrow (B \vee C))\ [0] \quad \text{или} \quad f = f_1 = (D \Rightarrow (B \vee C))\ [1],$$

где $f_0(D) = (0, B)$, $f_1(D) = (1, C)$. В подобных же условиях формула $((D \vee H) \Rightarrow (B \vee C))$ имеет 4 возможных реализации и т. д.

Если мы теперь соединим этот подход с секвенциальным подходом Герхарда Генцена [Gen35a; Gen35b], то следует задаться вопросом о роли секвенций в этих обстоятельствах. Здесь мы будем тестировать не сами формулы, а секвенции – утверждения о том, что одни формулы следуют из других. Будем считать, что для тестирования секвенции $\Gamma \to B$, где Γ – цепочка формул, B – формула, следует подобрать, по крайней мере, две конструкции: обосновывающую (подтверждающую) конструкцию f (то есть реализацию секвенции: программу, решающую задачу, сформулированную в виде этой секвенции) и проверяющую конструкцию f' (тест или набор тестов для этой реализации).

Утверждение о том, что данная секвенция обосновывается этими конструкциями, записывается в виде:

$$f : (\Gamma \to B) : f'.$$

Структура конструкций f и f', а также процедура проверки этого утверждения описывается далее в разделе 2. В разделе 2 также объясняется, что такое обосновывающая конструкция, как её можно выполнять на конкретной предметной области и тестировать в связанном с ней абстрактном универсуме.

Предлагаемый подход обладает важным вычислительным свойством: исчерпывающее тестирование программы f при помощи тестовой конструкции f' проводится за конечное время. По описанию приведённого ниже алгоритма нетрудно видеть, что временна́я сложность этого процесса является полиномиальной от размеров программы и формулы. Кроме того,

структура тестового комплекса f' однозначно выводится из структуры самой программы f и реализуемой формулы, что позволяет полностью автоматизировать процесс тестирования.

1. Обзор связанных работ

Предлагаемый подход опирается на несколько классических направлений исследований, которые мы кратко рассматриваем ниже.

Реализуемость Клини. Семантика реализуемости, введённая С. Клини в 1945 году [Kle45] и систематически изложенная в [Kle52], связывает истинность формул интуиционистской логики с существованием эффективных процедур, подтверждающих эту истинность. В рамках этого подхода каждой формуле сопоставляется реализатор – алгоритм, конструктивно свидетельствующий об её истинности. Реализуемость Клини стала основой для многих последующих конструктивных интерпретаций логики и заложила фундамент для автоматической экстракции программ из доказательств. Настоящая работа использует идею реализатора как программы, однако существенно расширяет её за счёт явного тестирующего компонента и конкретной операционной семантики каждого оператора.

Интерпретация «диалектика» Гёделя. В 1958 году К. Гёдель предложил функциональную интерпретацию арифметики Гейтинга [Göd58], известную под названием «диалектика». В этой интерпретации каждой формуле сопоставляется игра между «предлагающим» и «оппонентом»: истинность формулы означает наличие у предлагающего выигрышной стратегии. Данный подход оказал значительное влияние на теорию доказательств и теорию вычислений; его применения в современной математике систематически изложены в [Koh08]. В нашей работе роль оппонента частично берёт на себя тестирующая конструкция f', задающая условия, при которых реализация должна работать корректно.

Исчисление секвенций Генцена. Г. Генцен разработал исчисление секвенций LK и LJ [Gen35a; Gen35b], которое стало одним из основных инструментов структурной теории доказательств. Секвенциальный формализм позволяет явно управлять структурой доказательства и естественным образом соответствует структуре синтезируемых программ. В настоящей работе исчисление секвенций служит основой для описания правил построения обосновывающих и тестирующих конструкций. Естественный вывод, развитый в [Pra06], предоставляет альтернативный, но тесно связанный формализм.

Изоморфизм Карри–Говарда. Фундаментальная связь между доказательствами и программами, известная как изоморфизм Карри–Говарда [Cur72; How95], утверждает, что типы соответствуют формулам, а программы – доказательствам. Систематическое изложение этой связи и её следствий содержится в [Sør06]. Настоящая работа опирается на этот изоморфизм: конструкция f, обосновывающая секвенцию $\Gamma \to B$, является одновременно программой, вычисляющей реализацию B из реализаций Г, а её операционная семантика явно задаётся в каждом правиле.

Интуиционистская теория типов и системы доказательств. Теория типов Мартина–Лёфа [Mar84] и исчисление конструкций Кокана–Юэ [Coq86] обеспечивают мощные формальные основания для доказательно-корректного программирования. Система Nuprl [Con86] реализует эти идеи в виде интерактивной среды разработки программ с доказательствами. В отличие от этих систем наш подход ориентирован не на интерактивное доказательство, а на автоматическое тестирование извлечённых программ с полиномиальной сложностью.

Дедуктивный синтез программ. Манна и Уолдингер [Man80] разработали систематический дедуктивный подход к синтезу программ, в котором программа извлекается из конструктивного доказательства её спецификации. Верификационные подходы Хоара [Hoa69] и Дейкстры [Dij75] обеспечивают формальные гарантии корректности программ через аксиоматическую семантику и предусловия-постусловия. Современное состояние области синтеза программ изложено в обзоре [Gul17]; синтаксически управляемый синтез рассматривается в [Alu13]. Настоящая работа вписывается в эту традицию, предлагая механизм автоматической верификации синтезированных программ через структурное тестирование.

Двухуровневая реализация. Непосредственным предшественником настоящей работы является [Jou24], в которой была предложена двухуровневая семантика реализации, сочетающая элементы реализуемости Клини и интерпретации «диалектика» Гёделя. В этой работе для каждой формулы вводились пара (реализация, поддержка) и арбитражная процедура ar(b'', b, B, b'), проверяющая корректность реализации b при поддержке b'' и противодействии b'. Настоящая работа существенно развивает этот подход: тестирующая конструкция f' становится явным синтаксическим объектом исчисления с конкретной операционной семантикой, а не скрытой метапроцедурой; при этом обеспечивается автоматическое построение тестового комплекса.

2. Логический язык и универсум Эрбрана

2.1. Синтаксис логического языка

В качестве языка спецификаций используется простой вариант позитивной логики предикатов без функций и без отрицания. Выбор такого языка обусловлен задачами дедуктивного синтеза надёжных программ: в этом контексте недопустимо отбрасывать какие-либо варианты входных данных посредством отрицания; вместо этого все возможные ошибки должны быть явно предусмотрены и обработаны. Отсутствие функциональных символов упрощает построение универсума и делает тестовое пространство обозримым.

Синтаксис языка задаётся следующей грамматикой в форме Бэкуса–Наура:

```
<формула>   ::= <предикат> '(' <имя> {',' <имя>} ')'
        | '(' <формула> <связка> <формула> ')'
        | <квантор> <имя> <формула>
 <связка>    ::= '&' | 'v' | '=>'
<квантор>   ::='∀'|'∃'
```

Предполагается, что языки для понятий «*предикат*» и «*имя*» фиксированы и не порождают синтаксических неоднозначностей. Конкретное содержание предикатов зависит от предметной области применения.

Для логического вывода методом тестирования используются секвенции вида:

$$G_1,\ldots,G_n \;\rightarrow\; B,$$

где G_1, …, G_n – формулы-посылки; B – формула-заключение. Смысл такой секвенции близок к смыслу формулы (G_1 & … & G_n => B).

2.2. Подобие универсума Эрбрана

Наряду с реальными конкретными предметными областями, для которых предназначены решения исследуемых конструктивных задач, записанных в виде конструктивно понимаемых логических формул, нам понадобится некоторая абстрактная предметная область – подобие известного универсума Эрбрана, предложенное в своё время для обоснования формул, понимаемых классически. Классический универсум Эрбрана [Her30] строится на основе термов языка первого порядка. В нашем случае язык не содержит предметных функций, что существенно упрощает его предметную составляющую. Тем не менее конструктивная логика требует учёта не только предметных объектов, но и реализационных конструкций – программ, подтверждающих истинность формул. Надо отметить, что наш универсум не есть что-то глобальное, как эрбрановский – он может пополняться от секвенции к секвенции по мере построения рассуждения.

Предметный компонент универсума. Предметный универсум состоит из абстрактных имён. Новые абстрактные имена появляются по мере продвижения процесса тестирования.

Реализационный компонент универсума. Реализационные конструкции строятся индуктивно по структуре формул:

- Для истинного атома $P(\boldsymbol{c})$ реализационной конструкцией служит сам атом $P(\boldsymbol{c})$.

- Для конъюнкции (B & C) реализация есть упорядоченная пара (b, c), где b – реализация B; c – реализация C. Операция спаривания реализуется оператором cg (описанном далее в разделе 3.2).
- Для дизъюнкции ($B_0 \vee B_1$) реализация есть помеченная пара (i, b), где $i \in \{0, 1\}$, b – реализация B_i. Метки 0 и 1 указывают на часть формулы, которая реализована. Пометка создается оператором dg (описанном далее в разделе 3.3).
- Для импликации (B => C) в посылке секвенции реализация есть абстрактная функция вида (B=>C)[k], где k – указатель конкретного варианта исполнения. Например, если D, B и C имеют по одной реализации, то (D=>(B∨C)) [0] есть функция, возвращающая (0, B), а (D=>(B∨C)) [1] – функция, возвращающая (1, C). В подобных условиях ((D∨H)=>(B∨C)) имеет 4 возможных реализации. Задержанное вычисление реализуется оператором ig (описанном далее в разделе 3.4). Надо заметить, что для реализации доказываемых формул, в отличие от реализаций посылок секвенций, используются не абстрактные функции, а конкретные алгоритмические конструкции на языке, описанном ниже.
- Для универсальной формулы $\forall x B(x)$ в посылке секвенции реализация есть абстрактная функция, ставящая в соответствие каждому предмету a_i универсума реализацию $B(a_i)$. Если на самом деле таких реализаций может быть несколько, это учитывается при последующем тестировании путём применения нескольких тестов, полученных при использовании и разборе этой формулы. Создание новых предметов в абстрактном универсуме описано далее в разделе 3.5 и 3.11. Для универсальной формулы в заключении секвенции используются не абстрактные функции, а конкретные алгоритмические конструкции на языке, описанном далее.
- Для экзистенциальной формулы $\exists x B(x)$ реализация есть пара (a_i, b), где a_i – предмет из универсума; b – реализация $B(a_i)$. Создание такой пары реализуется оператором eg (описанном далее в разделе 3.6).

Аналогично естественным образом возникает также и понятие реализации секвенции.

3. Правила построения программ и их тестирования

Для точного описания того, что значит «секвенция обосновывается конструкцией f при тестировании f'», необходимо дать соответствующие правила тестирования. Далее приводится полный набор таких правил. Для каждого правила описывается: (1) условие применимости и тестирующая конструкция, (2) операционная семантика соответствующей программы – явная формула вычисления её результата на конкретной предметной области. Через $\boldsymbol{g}$ обозначается вектор значений переменных и формул из Γ; g_i – i-й элемент этого вектора. Правила приводятся для различных видов секвенций в зависимости от структур их правых и левых частей и различных видов обосновывающих конструкций. Ниже приводится список этих правил.

3.1. Правило тестирования проекции (pr – projection)

Простейшая обосновывающая конструкция – проекция. Она сопоставляет элемент левой части и правую часть секвенции. Правило записывается в виде:

$$\mathrm{pr}_i : (\Gamma \rightarrow P(\boldsymbol{c})) : \mathrm{pr}'(P(\boldsymbol{c}))$$
$$<=$$
$$G_i = P(\boldsymbol{c}),$$

где G_i – i-й элемент списка Γ; $P(\boldsymbol{c})$ – постоянный атом. Такая запись правила означает, что указанное в первой строке действие заключается в выполнении действия, указанного в третей строке. Если G_i не имеет указанного вида, то тест считается проваленным.

Здесь pr_i – программа i-й проекции, выдающая i-й элемент списка значений:

$$\mathrm{pr}_i\,(\boldsymbol{g}) = g_i,$$

pr'(B) – тривиальный тестирующий объект, который сравнивает результат применения программы к списку $\boldsymbol{g}$ с объектом B. Тест проходит тогда и только тогда, когда G_i синтаксически совпадает с $P(\boldsymbol{c})$.

3.2. Тестирование спаривающего оператора (cg – conjunction generation)

Спаривающий оператор создаёт реализацию пары. Правило для него записывается в виде:

$$\mathrm{cg}(f_0, f_1) : (\Gamma \to (B_0 \,\&\, B_1)) : \mathrm{cg}'(f'_0, f'_1) = \{(0, f_0'), (1, f_1')\}$$
$$<=$$
$$\text{для всех } i \in \{0,1\} \quad f_i : (\Gamma \to B_i) : f_i'.$$

Здесь выполняются два теста: для левой и правой частей пары. Операционная семантика программы $\mathrm{cg}(f_0, f_1)$ – следующая (как для конкретного, так и для абстрактного универсума):

$$\mathrm{cg}(f_0, f_1)(\boldsymbol{g}) = (f_0\,(\boldsymbol{g}), f_1(\boldsymbol{g})).$$

Результатом является упорядоченная пара значений, вычисленных применением f_0 и f_1 к одному и тому же вектору $\boldsymbol{g}$. Тестирующая конструкция $\mathrm{cg}'(f_0', f_1')$ применяет f_0' для проверки левой компоненты и f_1' – для проверки правой компоненты, независимо друг от друга.

3.3. Тестирование помечающих операторов (dg – disjunction generation)

Помечающий оператор создаёт реализацию дизъюнкции. Правило для него записывается в виде

$$\mathrm{dg}_i\,(f) : (\Gamma \to (B_0 \vee B_1)) : \mathrm{dg}'(f')$$
$$<=$$
$$f : (\Gamma \to B_i) : f',$$

где i – 0 или 1. При иных значениях i тест считается проваленным. Операционная семантика программы $\mathrm{dg}_i\,(f)$:

$$\mathrm{dg}_i\,(f)\,(\boldsymbol{g}) = (i, f\,(\boldsymbol{g})).$$

Результатом является помеченная пара: метка i фиксирует выбранную часть дизъюнкции, $f\,(g)$ – ее реализация. Тест $\mathrm{dg}'(f')$ проверяет корректность реализации выбранной части.

3.4. Тестирование оператора задержанного вычисления (ig – implication generation)

Оператор задержанного вычисления создаёт реализацию импликации. Правило для него записывается в виде:

$$\mathrm{ig}(f) : (\Gamma \to (B \Rightarrow C)) : \mathrm{ig}'(f')$$
$$<=$$
$$f : (\Gamma,\, B \to C) : f'.$$

Операционная семантика программы $\mathrm{ig}(f)$:

$$\mathrm{ig}(f)\,(\boldsymbol{g})\,(b) = f\,(\boldsymbol{g}, b).$$

Здесь выражение $\mathrm{ig}(f)(\boldsymbol{g})$ сразу не вычисляется, а остаётся в таком виде до попытки применить его к некоторому аргументу b. При этом применении и выполняется программа f с расширенным аргументом $(\boldsymbol{g}, b)$.

3.5. Тестирование оператора создания нового предмета (ag – ∀-generation)

Новый предмет универсума, создаётся как абстрактный аргумент (формальный параметр) подпрограммы. Это совершенно новый предмет, про который ничего не известно. При применении этой программы этот предмет будет заменён на конкретный аргумент (фактический параметр). Правило для проверки этого оператора записывается в виде

$$\text{ag}(f, a) : (\Gamma \to \forall x B(x)) : \text{ag}'(f')$$
$$<=$$
$$f : (\Gamma \to B(a)) : f',$$

где a – имя нового предмета, добавляемого в универсум (формальную предметную область). Впоследствии при дальнейшей обработке программы это имя может быть заменено. Операционная семантика программы ag(f, a):

$$\text{ag}(f, a)\ (\boldsymbol{g})\ (c) = f\,[a \to c](\boldsymbol{g}).$$

Это означает, что в программе f имя a заменяется на значение c. Здесь выражение ag(f, a)($\boldsymbol{g}$) также сразу не вычисляется, а остаётся в таком виде до попытки применить его к некоторому аргументу c. При конкретизации a значением c выполняется программа f с этой заменой. Тест ag'(f') проверяет корректность нового предмета универсума, про который ничего не было известно; отсюда следует, что этот тест пройдёт и для всех остальных предметов универсума.

3.6. Тестирование оператора конкретизации (eg – ∃-generation)

Оператор конкретизации помечает значение объектом из предметной области. Это правило служит для реализации существования и записывается в виде:

$$\text{eg}(b, f) : (\Gamma \to \exists x B(x)) : \text{eg}'(f')$$
$$<=$$
$$f : (\Gamma \to B(b)) : f',$$

где b – некоторый предмет из предметной области.
Операционная семантика программы eg(b, f):

$$\text{eg}(b, f)\ (\boldsymbol{g}) = (b, f\,(\boldsymbol{g})).$$

Результатом является пара: свидетельство b и реализация $B(b)$, вычисленная программой f. Тест eg'(f') проверяет корректность f для выбранного свидетельства b.

3.7. Тестирование оператора расщепления (cu – conjunction usage)

Оператор расщепления разделяет пару, реализующую конъюнкцию. Правило для его проверки записывается в виде:

$$\text{cu}_i(f) : (\Gamma \to D) : \text{cu}'(f')$$
$$<=$$
$$f : (\Gamma, B, C \to D) : f',\ G_i = (B\ \&\ C).$$

Если G_i не имеет указанного вида, то тест считается проваленным. Операционная семантика программы $\text{cu}_i(f)$:

$$\text{cu}_i\,(f)\ (\boldsymbol{g}) = f\,(\boldsymbol{g}, b, c), \text{ где } g_i = (b, c).$$

Реализация конъюнкции $G_i = (B\&C)$ расщепляется на компоненты b и c, которые добавляются в вектор аргументов для f. Если G_i не имеет вида $(B\&C)$, то тест провален.

3.8. Тестирование оператора разбора случаев (du – disjunction usage)

Оператор разбора случаев анализирует пометки и служит для использования реализации дизъюнкции. Правило для его проверки записывается в виде

$$\text{du}_i\,(f_0, f_1) : (\Gamma \to \text{D}) : \text{du}'(f_0', f_1') = \{(0, f_0'), (1, f_1')\}$$
$$<=$$
$$\text{для всех } j \in \{0,1\} : f_j : (\Gamma, B_j \to D) : f_j',\ G_i = (B_0 \vee B_1).$$

Если G_i не имеет указанного вида, то тест считается проваленным. Здесь тоже требуется выполнить два теста. Операционная семантика программы $\text{du}_i\,(f_0, f_1)$:

$$\text{du}_i\,(f_0, f_1)\ (\boldsymbol{g}) = f_j\ (\boldsymbol{g}, b), \text{ где } g_i = (j, b).$$

Дизъюнкция $G_i = (B_0 \vee B_1)$ разбирается по метке j: если $j = 0$, то применяется f_0 с добавлением b в качестве реализации B_0; если $j=1$, то применяется f_1 с добавлением b в качестве реализации B_1. Тест $\mathrm{du}'(f_0', f_1')$ состоит из двух тестов, проверяющих две ветви программы соответственно.

3.9. Тестирование оператора сложного вызова подпрограммы (iu – implication usage)

Оператор сложного вызова использует функцию, реализующую импликацию, передавая ей аргумент. Правило проверки этого оператора записывается в виде:

$$\mathrm{iu}_i(f, h) : (\Gamma \to D) : \mathrm{iu}'(f', h') = \{(0, f'), (1, h')\}$$
$$<=$$
$$f : (\Gamma \to B) : f',$$
$$h : (\Gamma, C \to D) : h',$$
$$G_i = (B => C).$$

Если G_i не имеет указанного вида, то тест считается проваленным. Здесь также выполняются два теста: тест получения посылки (f') и тест использования заключения (h'). Операционная семантика программы $\mathrm{iu}_i(f, h)$:

$$\mathrm{iu}_i(f, h)(\boldsymbol{g}) = h(\boldsymbol{g}, g_i(f(\boldsymbol{g}))).$$

Программа f применяется к $\boldsymbol{g}$ для получения реализации формулы B; затем реализация импликации $G_i = (B => C)$ применяется к этой реализации, давая реализацию c формулы C; наконец, h применяется к расширенному вектору $(\boldsymbol{g}, c)$ для получения реализации D.

3.10. Тестирование оператора простого вызова подпрограммы (au – ∀-usage)

Простой вызов подпрограммы используется для выполнения реализации формулы, начинающейся с квантора всеобщности. Правило для его проверки записывается в виде:

$$\mathrm{au}_i(b, f) : (\Gamma \to D) : \mathrm{au}'(f')$$
$$<=$$
$$f : (\Gamma, B(b) \to D) : f', \quad G_i = \forall x B(x).$$

Если G_i не имеет указанного вида, то тест считается проваленным. Операционная семантика программы $\mathrm{au}_i(b, f)$:

$$\mathrm{au}_i(b, f)(\boldsymbol{g}) = f(\boldsymbol{g}, g_i(b)).$$

Универсальная формула $G_i = \forall x B(x)$ применяется к конкретному предмету b, давая реализацию $B(b)$; эта реализация добавляется в вектор аргументов для f.

3.11. Тестирование оператора извлечения объекта (eu – ∃-usage)

Оператор извлечения объекта применяется для расщепления пары, содержащей предмет, и используется для применения реализации пары. Правило проверки этого правила записывается в виде:

$$\mathrm{eu}_i(a, f) : (\Gamma \to D) : \mathrm{eu}'(f)$$
$$<=$$
$$f : (\Gamma, B(a) \to D) : f', G_i = \exists x B(x),$$

где a – новый предмет, добавляемый в универсум. Если G_i не имеет указанного вида, то тест считается проваленным. Операционная семантика программы $\mathrm{eu}_i(a, f)$:

$$\mathrm{eu}_i(a, f)(\boldsymbol{g}) = f[a \to c](\boldsymbol{g}, b), \text{ где } \quad g_i = (c, b).$$

Реализация экзистенциальной формулы $G_i = \exists x B(x)$ содержит свидетельство c и реализацию $B(c)$, обозначенную b. В программе f формальное имя a заменяется на конкретное свидетельство c, после чего f выполняется с расширенным вектором аргумента $(\boldsymbol{g}, b)$.

4. Примеры

В данном разделе приводятся примеры, иллюстрирующие применение правил исчисления из раздела 3. Для каждого примера указывается доказываемая секвенция, строится обосновывающая конструкция f, тестирующая конструкция f', а также приводится пошаговое выполнение программы на конкретных данных.

4.1. Пример 1: проекция (правило pr)

Пусть доказываемая секвенция имеет вид:

$$P(a), Q(b) \rightarrow Q(b).$$

Требуется показать, что из двух посылок $P(a)$ и $Q(b)$ выводится $Q(b)$. Строим обосновывающую конструкцию f. Применяем правило тестирования проекции 3.1 с i=2, поскольку $Q(b)$ есть вторая формула в списке посылок:

$$f = \mathrm{pr}_2.$$

Условие применимости: $G_2 = Q(b)$ при $\Gamma = (P(a), Q(b))$. Условие выполнено.
Строим тестирующую конструкцию f':

$$f' = \mathrm{pr}'(Q(b)).$$

Тестирующая конструкция $\mathrm{pr}'(Q(b))$ проверяет, что вторая формула в списке посылок действительно есть $Q(b)$. Полная запись обоснования выглядит так:

$$\mathrm{pr}_2: (P(a), Q(b) \rightarrow Q(b)) : \mathrm{pr}'(Q(b)).$$

Рассмотрим процессы тестирования и выполнения. Пусть конкретные значения посылок задаются так: $\boldsymbol{g} = (p_a, q_b)$, где p_a – реализация $P(a)$, а q_b – реализация $Q(b)$.

При тестировании на абстрактном универсуме программа pr_2 извлекает второй элемент вектора $\Gamma = (P(a), Q(b))$:

$$\mathrm{pr}_2(\Gamma) = G_2 = Q(b).$$

Далее тестирующая конструкция $\mathrm{pr}'(Q(b))$ проверяет, что $\mathrm{G}_2 = Q(\mathrm{b})$:

$$\mathrm{pr}'(\mathrm{Q(b)}): \text{проверка } G_2 = Q(b) \Rightarrow \text{тест пройден.}$$

При выполнении программы на конкретных значениях результат выполнения программы: q_b – реализация $Q(b)$. Значит, программа корректна.

4.2. Пример 2: введение конъюнкции (правила pr и cg)

Пусть доказываемая секвенция имеет вид

$$P(a), \mathrm{Q}(b) \rightarrow (Q(b) \,\&\, P(a)).$$

Требуется из посылок $P(a)$ и $Q(b)$ вывести их конъюнкцию в обратном порядке.
Построим обосновывающую конструкцию f. Заключение есть конъюнкция $(Q(b)\&P(a))$. Значит, применяем правило введения конъюнкции 3.2:

$$f = \mathrm{cg}(f_0, f_1),$$

где f_0 обосновывает левую часть $Q(b)$, а f_1 обосновывает правую часть $P(a)$. Для f_0:$Q(\mathrm{b})$ есть вторая формула в посылках:

$$f_0 = \mathrm{pr}_2.$$

Для f_1: $P(a)$ есть первая формула в посылках:

$$f_1 = \mathrm{pr}_1.$$

Итоговая обосновывающая конструкция:

$$f = \mathrm{cg}(\mathrm{pr}_2, \mathrm{pr}_1).$$

Построение тестирующей конструкции f'.

По правилу 3.2 тестирующая конструкция есть пара тестов для каждого компонента:

$$f' = \mathrm{cg}'(f_0', f_1') = \{(0, \mathrm{pr}'(Q(b))), (1, \mathrm{pr}'(P(a)))\}.$$

Полная запись обоснования выглядит так:

$$\mathrm{cg}(\mathrm{pr}_2, \mathrm{pr}_1) : (P(a), Q(b) \rightarrow (Q(b) \,\&\, P(a))) : \{(0, \mathrm{pr}'(Q(b))), (1, \mathrm{pr}'(P(a)))\}.$$

Рассмотрим процессы тестирования и выполнения. Пусть $\Gamma = (P(a), Q(b))$, $\boldsymbol{g} = (p_a, q_b)$.

Шаг 1. Получаем левую часть конъюнкции:

$$f_0(\Gamma) = \mathrm{pr}_2(\Gamma) = G_2 = Q(b),$$
$$f_0(\boldsymbol{g}) = \mathrm{pr}_2(\boldsymbol{g}) = \mathrm{g}_2 = q_b.$$

Шаг 2. Получаем правую часть конъюнкции:

$$f_1(\Gamma) = \mathrm{pr}_1(\Gamma) = \mathrm{G}_1 = P(a),$$
$$f_1(\boldsymbol{g}) = \mathrm{pr}_1(\boldsymbol{g}) = \mathrm{g}_1 = p_a.$$

Шаг 3. Программа cg формирует пару:

$$\mathrm{cg}(\mathrm{pr}_2, \mathrm{pr}_1)(\Gamma) = (f_0(\Gamma), f_1(\Gamma)) = (Q(b), P(a)),$$
$$\mathrm{cg}(\mathrm{pr}_2, \mathrm{pr}_1)(\boldsymbol{g}) = (f_0(\boldsymbol{g}), f_1(\boldsymbol{g})) = (q_b, p_a).$$

Шаг 4. Тест (0, pr'(Q(b))) проверяет первый компонент:

$$G_2 = Q(b) \Rightarrow \text{тест пройден.}$$

Шаг 5. Тест (1, pr'($P(a)$)) проверяет второй компонент:

$$G_1 = P(a) \Rightarrow \text{тест пройден.}$$

Результат вычисления: (q_b, p_a) – корректная реализация $(Q(b) \,\&\, P(a))$.

5. Корректность абстрактного тестирования

Заметим, что правила тестирования секвенций на абстрактном универсуме были изложены так, что было показано, что корректность реализаций формул, выявленная в процессе их тестирования на абстрактном универсуме, означает их корректную работу на конкретных универсумах, что может быть сформулировано в виде следующей теоремы.

Теорема. Если секвенция $\Gamma \rightarrow B$ обосновывается конструкциями f и f':

$$f : (\Gamma \rightarrow B) : f',$$

то f – реализация секвенции $\Gamma \rightarrow B$.

Заключение

В работе предложен подход к обоснованию конструктивных задач, в котором доказательство разрешимости является программой, а проверка этого доказательства – структурным тестированием программы. Метод сочетает классические подходы: формальное исчисление секвенций Генцена [Gen35a; Gen35b] как основу структуры доказательства, универсум Эрбрана [Her30] как инструмент построения конечного тестового пространства, реализационную семантику в духе Клини [Kle45; Kle52] как связь между доказательствами и программами, а также элементы диалектической интерпретации Гёделя [Göd58] через роль тестирующей конструкции как «оппонента».

Основные результаты работы состоят в следующем.

Во-первых, для конструктивной позитивной логики предикатов без функций строятся аналоги универсума Эрбрана, учитывающие как предметные объекты, так и реализационные конструкции.

Во-вторых, введена схема обоснования секвенций в нотации

$$f : (\Gamma \to B) : f',$$

где тестирующая конструкция f' задаёт явный набор проверочных процедур, автоматически выводимых из структуры f и секвенции.

В-третьих, для каждого правила тестирования явно описана операционная семантика соответствующей программы – формула вычисления результата на конкретной предметной области. Это делает подход непосредственно реализуемым, что подтверждается примерами раздела 4.

По сравнению с предшествующей работой [Jou24], настоящий подход делает тестирующую конструкцию явным синтаксическим объектом исчисления с конкретной операционной семантикой, а не скрытой метапроцедурой. Связь с изоморфизмом Карри–Говарда [Cur72; How95; Sør06] обеспечивает возможность непосредственной экстракции исполняемого кода из доказательств.

Дальнейшие направления исследований включают: изучение вопросов полноты введённого исчисления; расширение подхода на логику с функциональными символами; практическую реализацию системы автоматического тестирования синтезированных программ; исследование связи с теорией типов Мартин–Лёфа [Mar84] и исчислением конструкций [Coq86]; применение предложенного подхода к конкретным задачам дедуктивного синтеза надёжных программ в критических областях.

СПИСОК ЛИТЕРАТУРЫ | REFERENCES

[Alu13] Alur R. et al. Syntax-guided synthesis // 2013 Formal Methods in Computer-Aided Design. Portland: IEEE, 2013. P. 1–8.

[Coq86] Coquand T., Huet G. The calculus of constructions // INRIA. 1986.

[Con86] Constable R. L. et al. Implementing mathematics // Nuprl Proof Development System. 1986.

[Cur72] Curry H. B., Hindley J. R., Seldin J. P. Combinatory Logic. Volume II. North-Holland Publishing Company, 1972.

[Dij75] Dijkstra E. W. Guarded commands, nondeterminacy and formal derivation of programs // Communications of the ACM. 1975. V. 18, No. 8. P. 453–457.

[Gen35a] Gentzen G. Untersuchungen über das logische Schließen. I // Mathematische Zeitschrift. 1935. V. 39, No. 1. P. 176–210.

[Gen35b] Gentzen G. Untersuchungen über das logische Schließen. II // Mathematische Zeitschrift. 1935. V. 39, No. 1. P. 405–431.

[Göd58] Gödel V. K. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes // Dialectica. 1958. V. 12, No. 3–4. P. 280–287.

[Gul17] Gulwani S., Polozov O., Singh R. Program synthesis // Foundations and Trends in Programming Languages. 2017. V. 4, No. 1–2. P. 1–119.

[Her30] Herbrand J. Recherches sur la théorie de la demonstration // J. Dziewulski. 1930. V. 33. P. 33–160.

[Hil99] Hilbert D., Ackermann W. Principles of mathematical logic // American Mathematical Soc. 1999. V. 69.

[Hoa69] Hoare C. A. R. An axiomatic basis for computer programming // Communications of the ACM. 1969. V. 12, No. 10. P. 576–580.

[How95] Howard W. A. The Formulae-as-Types Notion of Construction / Philippe De Groote, The Curry-Howard isomorphism. Louvain-la-Neuve: Academia. 1995. https://philpapers.org/rec/HOWTFN

[Jou24] Joudakizadeh M., Bel'tyukov A. P. Two-level realization of logical formulas for deductive program synthesis // Bulletin of Udmurt University. Mathematics. Mechanics. Computer Science. 2024. V. 34, No. 4. P. 469–485. EDN: JLCQGU.

[Kle45] Kleene S. C. On the interpretation of intuitionistic number theory // The Journal of Symbolic Logic. 1945. V. 10, No. 4. P. 109–124.

[Kle52] Kleene S. C. Introduction to Metamathematics. Van Nostrand, Princeton, 1952.

[Koh08] Kohlenbach U. Applied Proof Theory: Proof Interpretations and their Use in Mathematics. Berlin, Heidelberg: Springer, 2008.

[Man80] Manna Z., Waldinger R. A deductive approach to program synthesis // ACM Transactions on Programming Languages and Systems. 1980. V. 2, No. 1. P. 90–121.

[Mar84] Martin-Löf P., Sambin G. Intuitionistic type theory. Naples: Bibliopolis, 1984. V. 9. P. 136.

[Pra06] Prawitz D. Natural deduction: A proof-theoretical study. Courier Dover Publications, 2006.

[Sør06] Sørensen M. H., Urzyczyn P. Lectures on the Curry-Howard isomorphism. Elsevier, 2006. T. 149.

ОБ АВТОРАХ | ABOUT THE AUTHORS

ДЖУДАКИЗАДЕ Милад
Удмуртский государственный университет, Россия.
joudakizadeh@mail.ru ORCID: 0000-0002-6167-6237.
Аспирант, кафедра вычислительных технологий и интеллектуальных систем больших данных.

БЕЛЬТЮКОВ Анатолий Петрович
Удмуртский государственный университет, Россия.
belt.udsu@mail.ru ORCID: 0000-0002-3433-9067.
Д-р физ.-мат. наук, проф., кафедра вычислительных технологий и интеллектуальных систем больших данных.

JOUDAKIZADEH Milad
Udmurt State University, Russia.
joudakizadeh@mail.ru ORCID: 0000-0002-6167-6237.
PhD Candidate, Department of Computing Intellectual Systems.

BELTIUKOV Anatoly Petrovich
Udmurt State University, Russia.
belt.udsu@mail.ru ORCID: 0000-0002-3433-9067.
Doctor of Physics and Mathematics, Professor, Department of Computing Intellectual Systems.

МЕТАДАННЫЕ | METADATA

Заглавие: Тестирование в качестве доказательства разрешимости конструктивных задач.

Авторы: Джудакизаде М., Бельтюков А. П.

Аннотация: Предлагается подход к решению конструктивных задач методом построения конструктивных доказательств их разрешимости, при котором доказательство оказывается программой, а его проверка – тестированием этой программы. При этом исчерпывающее тестирование проводится за конечное время. Более того, структура тестового комплекса автоматически выводится из структуры программы. Метод сочетает подходы Д. Гильберта, Ж. Эрбрана и Г. Генцена и является развитием трёхуровневой реализации логических формул. Для полного набора логических связок и кванторов позитивной логики предикатов без функций вводятся правила тестирования секвенций в нотации $f{:}(\Gamma{\to}B){:}f'$, где f — обосновывающая конструкция (реализация), f' – тестирующая конструкция. Для каждого правила явно описывается операционная семантика соответствующей программы.

Ключевые слова: Конструктивная логика; синтез программ; исчисление секвенций; реализуемость; универсум Эрбрана; тестирование; трёхуровневая реализация; дедуктивный синтез; временная сложность.

Язык: Русский.

Статья поступила в редакцию 2 февраля 2026 г.

Title: Testing as a proof of solvability of constructive problems.

Authors: Joudakizade M., Beltiukov A. P.

Abstract: An approach is proposed to solving constructive problems by constructing constructive proofs of their decidability, in which the proof turns out to be a program and its verification amounts to testing that program. Exhaustive testing is carried out in finite. Moreover, the structure of the test suite is automatically derived from the structure of the program. The method combines the approaches of D. Hilbert, J. Herbrand, and G. Gentzen and constitutes a development of the three-level realization of logical formulas. Sequent testing rules are introduced in the notation $f{:}(\Gamma{\to}B){:}f'$, where f is a justifying construction (realization together with support) and f' is a testing construction, covering the full set of logical connectives and quantifiers of positive predicate logic without functions. For each rule the operational semantics of the corresponding program is explicitly described.

Key words: Constructive logic, program synthesis, sequent calculus, realizability, Herbrand universe, testing, three-level realization, deductive synthesis, time complexity.

Language: Russian.

The article was received by the editors on 2 February 2026.