*ПРОГРАММИРОВАНИЕ*, *Рабочий полный вариант*, *№ версии 2*, *с. 1–33*



*УДК 519.681 + 510.649*

# МЕТОД СХЕМ ПРОГРАММ ДЛЯ ПРОПОЗИЦИОНАЛЬНЫХ ПРОГРАММНЫХ ЛОГИК ЗА 30 ЛЕТ

**© Рабочий полный вариант г. Н. В. Шилов*, С. О. Шилова*, А. Ю. Бернштейн****

**Институ систем информатики им. А. П. Ершова*
*630090 Новосибирск, пр. Лаврентьева, 6*
***Новосибирский государственный университет*
*630090 Новосибирск, ул. Пирогова, 2*
*E-mail: shilov@iis.nsk.su, shilov61@inbox.ru, bahton@gmail.com*

Поступила в редакцию ??.??.????

В настоящей статье представлен обзор так называемого схемного метода доказательства разрешимости пропозициональных программных логик. Этот метод основан на сведении к проблеме относительной тотальности недетерминированных схем Янова. Работа выполнена в рамках проекта РФФИ № 13-01-00645-а.

## 1. ВВЕДЕНИЕ

### 1.1 Актуальность темы и краткий исторический обзор

Семейство программных логик включает динамические и темпоральные логики, а также логики процессов. Особо популярны пропозициональные варианты этих логик в связи с их широкой применимостью для спецификации и автоматической верификации как отдельных программ, так и программных систем (включая распределённые и мультиагентные системы). Поэтому исследование и разработка алгоритмов (разрешающих процедур) для валидации (проверки тождественной истинности), поиска или генерации доказательств, верификации в моделях (model checking) формул программных логик является важным направлением теории программирования.

Разрешимость программной логики — это существование алгоритма валидации формул этой логики. Суть схемного метода [13, 38, 18] доказательства разрешимости программных логик состоит в следующем. Формулы исследуемой логики транслируются в так называемые недетерминированные схемы Янова (которые являются естественным обобщением классических схем Янова [45, 46, 32, 33]) таким образом, что транслируемая формула истинна тогда и только тогда, когда соответствующая схема является тотальной (т.е. всегда останавливается) в специальном классе интерпретаций. Так как такая проблема тотальности разрешима (т.е. имеется алгоритм проверки тотальности в специальном классе интерпретаций), то исследуемая пропозициональная программная логика также оказывается разрешимой. (Более того, при аккуратном анализе сложности алгоритма трансляции и разрешающей процедуры для проблемы относительной тотальности схемный метод позволяет получить верхнюю оценку сложности для проблемы разрешимости исследуемой пропозициональной программной логики.)

Первоначальный вариант схемного метода был разработан в 1983-1988 гг. В.А. Непомнящим и Н.В. Шиловым [13, 38] для доказательства разрешимости Пропозициональной Динамической Логики (Propositional Dynamic Logic, PDL) [10, 11, 31], ее строгого и/или детерминированного вариантов. В первоначальном варианте схемного метода для разрешимости PDL и её вариантов использовалась так называемая *относительная тотальность* недетермини-

рованных схем Янова. В 1993-1997 схемный метод был распространён [18] на так называемое $\mu$-Исчисление ($\mu$-Calculus, $\mu$C) [12, 31, 3], которое можно считать расширением PDL за счет использования неподвижных точек по пропозициональным переменным (для выражения, например, зацикливания программ). В версии схемного метода из [18] была использована так называемая *обобщённая тотальность* (допускающая квантификацию пропозициональных переменных). Следует отметить, что в работах 1983-1997 гг. для доказательства разрешимости обобщённой тотальности в терминах недетерминированных схем Янова были разработаны специальные алгоритмы (обобщающие аналогичный метод А.П. Ершова [32, 33] для классических схем Янова [45, 46, 32, 33]).

Схемный метод развивался одновременно и независимо от теоретико-автоматного метода [22, 27, 9] доказательства разрешимости пропозициональных программных логик. Вообще говоря, связь между конечными автоматами и схемами Янова была установлена в [16, 17], где было показано, что схемы Янова — это конечные автоматы над специальным алфавитом (состоящим из означенных пропозициональных переменных). Однако, во-первых, в теоретико-автоматном методе используются не конечные автоматы, а автоматы на бесконечных словах или деревьях, и, во-вторых, теоретико-автоматный метод решает проблему выполнимости формул (сводя её к проблеме непустоты языка автомата), в то время как схемный метод решает проблему тождественной истинности (сводя её к проблеме относительной тотальности схемы).

В настоящей статье для полноты изложения описаны: сведение проблемы разрешимости Элементарной Пропозициональной Динамической Логики к проблеме относительной тотальности, новый вариант сведения проблемы разрешимости $\mu$-Исчисления к проблеме относительной тотальности[1], и исправленный алгоритм проверки относительной тотальности недетерминированных схем Янова [13, 38]. В заключении статьи мы кратко обсудим новый вариант схемного метода [20] для $\mu$-Исчисления, основанный на методе верификации специальной формулы $\mu$-Исчисления в конечных моделях, заданных блок-схемами недетерминированных схем Янова.

[1] вместо проблемы *обобщённой* относительной тотальности из [18]

## 1.2 Логическое введение

Для однозначного понимания некоторых общих терминов начнём с определения синтасиса и семантики пропозициональной булевской логики BOOL.

**Определение 1.** *Пусть* $Con = \{true, false\}$ *— алфавит символов булевских констант, а* $Var$ *— не пересекающийся с ним счетный алфавит пропозициональных переменных. Синтаксис BOOL состоит из формул, которые определяются структурной индукцией:*

- *Константы из* $Con$ *и переменные из* $Var$ *являются формулами.*
- *Любая пропозициональная комбинация формул является формулой: отрицание* $(\neg\phi)$*, конъюнкция* $(\phi \wedge \psi)$ *и дезъюнкция* $(\phi \vee \psi)$ *формул* $\phi$ *и* $\psi$ *являются формулами.*

В записи формул BOOL мы будем опускать самые внешние скобки и те скобки внутри формул, которые можно восстановить, исходя из стандартных правил приоритета: отрицание имеет высший приоритет, затем следует конъюнкция, а дизъюнкция имеет низший приоритет. Например, формула $\big((\neg p) \vee (q \wedge true)\big)$, где $p, q \in Var$, записывается как $\neg p \vee q \wedge true$.

Для записи формул BOOL и других логик, расширяющих BOOL, будем использовать производные связки $\rightarrow$ для импликации и $\leftrightarrow$ для эквивалентности. Мы вводим импликацию и эквивалентность как макросы:

- $(\phi \rightarrow \psi)$ — макрос для $((\neg\phi) \vee \psi)$;
- $(\phi \leftrightarrow \psi)$ — макрос для $\big((\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)\big)$ или (если произвести окончательную макроподстановку) для $\Big(\big((\neg\phi) \vee \psi\big) \wedge \big((\neg\psi) \vee \phi\big)\Big)$.

В тех случаях, когда мы будем использовать макросы и опускать часть скобок, будем считать, что импликация имеет меньший приоритет, чем дизъюнкция, а эквивалентность — меньший приоритет, чем импликация.

Семантика формул пропозициональной булевской логики определяется в терминах оценок пропозициональных переменных.

**Определение 2.** *Оценка пропозициональных переменных — это произвольное отображение* $V : Var \to \{0,1\}$*, которое сопоставляет каждой пропозициональной переменной* $p \in Var$ *её булевское значение:* $V(p) \in \{0,1\}$*. Произвольная оценка* $V$ *может быть структурной индукцией распространена на все формулы BOOL:* $V(true) = 1$*,* $V(false) = 0$*, для отрицания* $V(\neg\phi) = 1 - V(\phi)$*, а для конъюнкции и дизъюнкции — на основании стандартных таблиц истинности:*

| *конъюнкция* | | | | |
|---|---|---|---|---|
| $V(\phi)$ | 0 | 1 | 0 | 1 |
| $V(\psi)$ | 0 | 0 | 1 | 1 |
| $V(\phi \wedge \psi)$ | 0 | 0 | 0 | 1 |

| *дизъюнкция* | | | | |
|---|---|---|---|---|
| $V(\phi)$ | 0 | 1 | 0 | 1 |
| $V(\psi)$ | 0 | 0 | 1 | 1 |
| $V(\phi \vee \psi)$ | 0 | 1 | 1 | 1 |

*Формула* $\phi$ *называется*

- *выполнимой, если существует такая оценка* $V$*, что* $V(\phi) = 1$*;*
- *(тождественно) истинной, если* $V(\phi) = 1$ *для любой оценки* $V$*.*

Приведём также следующие определения общелогических понятий. *Исчисление* — это формальный язык с синтаксически-управляемой системой вывода; если эта система вывода имеет аксиомы (т.е. правила вывода без посылок), то доказуемые предложения (данного языка) — это те, которые можно вывести из аксиом. Система вывода с аксиомами называется аксиоматической системой. Говорят, что формальный язык, для предложений которого определено семантическое (т.е. с использованием моделей) понятие (тождественной) истинности,

- *разрешим*, если существует алгоритм (рекурсивная функция), решающий проблему истинности для предложений этого языка;
- *перечислим*, *частично разрешим* или *аксиоматезируем* если существует алгоритм (частично рекурсивная функция), перечисляющий в каком-либо порядке все истинные предложения этого языка.

Говорят, что исчисление (в частности, аксиоматическая система), для предложений которого определено семантическое понятие истинности

- *непротиворечиво* или *совместно* (consistent), если любое доказуемое предложение выполнимо;
- *корректно* (sound), если всякое доказуемое предложение является тождественно истинным;
- *полно* (complete), если всякое тождественно истинное предложение является доказуемым.

Формальный язык, для предложений которого определено семантическое понятие истинности, называется *(синтаксически) аксиоматизируемым*, если существует корректная и полная аксиоматическая система для истинных предложений этого языка[2].

Проиллюстрируем эти понятия на примере логики BOOL. Рассмотрим в качестве формального языка множество всех формул BOOL. Если этот язык снабдить *пустой* системой вывода[3], то получится совместное и корректное, но не полное исчисление. Если же этот язык снабдить системой вывода, состоящей из аксиом, заданных одной общей схемой $\frac{}{\phi}$ (постулирующей выводимость за один шаг любой формулы), то получится противоречивое, некорректное, но полное исчисление. В качестве непротиворечивой, корректной и полной системы вывода для BOOL можно рассмотреть следующую аксиоматическую систему:

$$\frac{}{\phi,\neg\phi,\Gamma} \quad \frac{\phi,\psi,\Gamma}{\phi\vee\psi,\Gamma} \quad \frac{\phi,\Gamma \qquad \psi,\Gamma}{\phi\wedge\psi,\Gamma}$$

$$\frac{\phi,\Gamma}{\neg\neg\phi,\Gamma} \quad \frac{\neg\phi\vee\neg\psi,\Gamma}{\neg(\phi\wedge\psi),\Gamma} \quad \frac{\neg\phi\wedge\neg\psi,\Gamma}{\neg(\phi\vee\psi),\Gamma} \quad ,$$

которая аксиоматизирует логику BOOL. А разрешимость этой логики с экспоненциальной верхней оценкой сложности $\exp(n)$ (где $n$ — размер записи входной формулы) следует из определения её семантики: достаточно

[2] Как видно из этих определений, есть некоторая неоднозначность в употреблении термина *аксиоматезируемость*: он может означать только перечислимость множества истинных предложений каким-либо алгоритмом, а может — существование корректной и полной аксиоматизации, т.е. перечислимость множества посредством логического вывода.

[3] т.е. не имеющей ни одного правила вывода

построить таблицу истинности для входной формулы[4].

## 2. ПРОПОЗИЦИОНАЛЬНЫЕ ДИНАМИЧЕСКИЕ ЛОГИКИ

### 2.1 Элементарная Пропозициональная Динамическая Логика

Знакомство с пропозициональными динамическими логиками начнем с так называемой Элементарной ПДЛ (Elementary PDL, EPDL) [31, 3], её синтаксиса и семантики[5]. Синтаксис EPDL расширяет синтаксис пропозициональной булевской логики BOOL следующим образом.

**Определение 3.** *Пусть Con и Var — это алфавит символов те же алфавиты булевских констант и пропозициональных переменны, что и в BOOL, а Act — непересекающийся с ними (счетный) алфавит программных переменных. Синтаксис EPDL состоит из формул, которые определяются структурной индукцией следующим образом.*

- *Константы из Con и переменные из Var являются формулами.*
- *Любая пропозициональная комбинация формул является формулой.*
- *Для любой программной переменной $a \in Act$ и формулы $\phi$ модальные конструкции возможности $(\langle a\rangle\phi)$ и необходимости $([a]\phi)$ являются формулами.*

Таким образом, синтаксис EPDL допускает дополнительно к синтаксису BOOL модальности возможности и необходимости. Как обычно, для удобства записи формул EPDL будем опускать самые внешние скобки и те скобки внутри формул, которые можно восстановить, исходя из расширенных правил приоритета: отрицание и модальности имеют высший приоритет, затем следует конъюнкция, дизъюнкция имеет низший приоритет[6]. Например, формула $\Big(\big(\neg([a]p)\big) \vee \big((\langle b\rangle q) \wedge true\big)\Big)$, где $a, b \in Act$ и $p, q \in Var$, должна быть записана как $\neg[a]p \vee \langle b\rangle q \wedge true$.

Семантика программных логик определяется в моделях, которые называются системами размеченных переходов (labeled transition systems, LTS) в компьютерных науках, терминологическими интерпретациями (terminological interpretations) в искусственном интеллекте и представлении знаний, и системами/моделями Крипке (Kripke systems/structures) в математической и философской логике. Мы резервируем в рамках данной статьи термин *модель*, но иногда будем говорить о системах размеченных переходов и использовать сокращение LTS.

**Определение 4.** *Каждая модель $M$ — это тройка $(D, R, E)$ где*

- *область (или носитель) $D \neq \varnothing$ — непустое множество состояний[7];*
- *интерпретация $R : Act \to 2^{D\times D}$ — это функция, которая сопоставляет каждой программной переменной $a \in Act$ бинарное отношение $R(a) \subseteq D \times D$;*
- *означивание $E : Var \to 2^D$ — функция, которая сопоставляет каждой пропозициональной переменной $x \in Var$ одноместный предикат (т.е. подмножество) $E(x) \subseteq D$.*

*Если $M$ — это модель $(D, R, E)$, то область модели $D$ будем обозначать $D_M$, интерпретацию $R$ — $R_M$, а означивание $E$ — $E_M$.*

Таким образом, в каждой модели $M = (D, R, E)$ каждое состояние $s \in D$ можно рассматривать как оценку пропозициональных

[4] Мы здесь не обсуждаем ни проблему **P** *vs.* **NP**, ни практичность алгоритма построения таблиц истинности.

[5] Для специалистов по математической и философской логике заметим, что EPDL — это полимодальный вариант базовой логики **K** [6, 39] с несколько особенным синтаксисом, а для специалистов по искуственному интеллекту и представлению знаний EPDL — базовая дискрипционная логика Attribute Language with Compliment $\mathcal{ALC}$ [24, 41]; только нотация (т.е. синтаксис и терминология) $\mathcal{ALC}$ сильно отличаются от EPDL и полимодального варианта **K**.

[6] Если используются макросы для импликации и эквивалентности, то (как и в BOOL) они имеют приоритет ниже дизъюнкции.

[7] Термин принят в компьютерных науках; в искусственном интеллекте и представлении знаний используют термин *обвекты*, а в математической и философской логике — *миры*.

переменных $V_s : Var \to \{0,1\}$, определенную по правилу

$$V_s(p) = \text{если } s \in E(p) \text{ то } 1 \text{ иначе } 0$$

для любой $p \in Var$.

Семантика EPDL определяется посредством распространения означивания (заданного моделью для пропозициональных переменных) на все формулы EPDL (структурной индукцией по построению формул).

**Определение 5.** *Для произвольной модели $M = (D, R, E)$ имеем*

- $M(true) = D$, $M(false) = \varnothing$;
- $M(x) = E(x)$ *для любой пропозициональной переменной* $x \in Var$;
- $M(\neg\phi) = D \setminus M(\phi)$,
  $M(\phi \wedge \psi) = M(\phi) \cap M(\psi)$,
  $M(\phi \vee \psi) = M(\phi) \cup M(\psi)$;
- $M(\langle a\rangle\phi) = \{s \in D \ : \ \exists t \in D((s,t) \in R(a) \text{ и } t \in M(\phi))\}$
  *для любой программной переменной* $a \in Act$;
- $M([a]\phi) = \{s \in D \ : \ \forall t \in D((s,t) \in R(a) \text{ влечёт } t \in M(\phi))\}$
  *для любой программной переменной* $a \in Act$.

Отношение выполнимости $\models$ — это трёхместное отношение между состояниями, моделями и формулами: $s \models_M \phi$, если $s \in M(\phi)$.

**Определение 6.** *Пусть $\phi$ — формула EPDL.*

- *Пусть $M$ — модель; если $M(\phi) = D_M$, говорят, что $\phi$ истинна в $M$ и пишут $\models_M \phi$.*
- *Если $\phi$ истинна в любой модели, то говорят, что $\phi$ (тождественно) истинна и пишут $\models \phi$.*
- *Пусть $M$ — модель; если существует состояние $s \in D_M$, в котором $s \models_M \phi$, то говорят что формула $\phi$ выполнима в модели $M$.*
- *Если $\phi$ выполнима в какой-либо модели, то говорят, что $\phi$ выполнима.*

## 2.2 Пропозициональная Динамическая Логика

Синтаксис PDL расширяет синтаксис EPDL за счет использования (структурированных недетерминированных схем) программ следующим образом.

**Определение 7.** *Пусть Con, Var и Act — это те же алфавиты символов булевских констант, пропозициональных и программных переменных, что и в EPDL. Синтаксис PDL состоит из формул и программ, которые определяются взаимной структурной индукцией:*

***Формулы:***

- *Константы из Con и переменные из Var являются формулами, любая пропозициональная комбинация формул является формулой.*
- *Для любой PDL-программы $\alpha$ и формулы $\phi$ модальные конструкции возможности ($\langle\alpha\rangle\phi$) и необходимости ($[\alpha]\phi$) являются формулами.*

***Программы:***

- *Любая программная переменная $a \in Act$ является (элементарной) программой.*
- *Для любой PDL-формулы $\phi$ тест $\phi?$ является программой.*
- *Для любых программ $\alpha$ и $\beta$ (последовательная) композиция $(\alpha;\beta)$, (недетерминированный) выбор $(\alpha;\beta)$ и (недетерминированная) итерация $(\alpha^*)$ являются программами.*

Таким образом, синтаксис EPDL — фрагмент синтаксиса PDL, в котором можно использовать только элементарные программы. Для удобства записи формул PDL мы будем использовать те же правила опускания скобок, что и для EPDL. Кроме того, для удобства записи программ PDL будем опускать самые внешние скобки и те скобки внутри формул, которые можно восстановить, исходя из следующих правил приоритета: тест и итерация имеют высший приоритет, затем следует композиция, а выбор имеет низший приоритет. Например, программа $\Big(\big((p \vee q)?; a\big) \cup$

$\Big(b;(true?^*)\big)\Big)$, где $a,b \in Act$ и $p,q \in Var$, должна быть записана как $p \vee q?; a \cup b; true?^*$ .

Как сказано выше, семантика программных логик определяется в моделях: семантика PDL-формул определяется посредством распространения означивания (заданного моделью для пропозициональных переменных) структурной индукцией по построению формул аналогично EPDL; семантика PDL-программ определяется посредством распространения интерпретации (заданной моделью для программных переменных) структурной индукцией по построению программ. Иначе говоря, интерпретация PDL-программы в модели — это бинарное отношение вход-выход на множестве состояний.

**Определение 8.** *Для произвольной модели $M = (D, R, E)$ имеем:*

***Означивание формул:***

- *аналогично определению 5, $M(true) = D$, $M(false) = \varnothing$, $M(x) = E(x)$ для любой пропозициональной переменной $x \in Var$, $M(\neg\phi) = D \backslash M(\phi)$, $M(\phi \wedge \psi) = M(\phi) \cap M(\psi)$, $M(\phi \vee \psi) = M(\phi) \cup M(\psi)$;*
- $M(\langle\alpha\rangle\phi) = \{s \in D \ :$ $\exists t \in D((s,t) \in M(\alpha)$ *и* $t \in M(\phi))\}$ *для любой PDL-программы $\alpha$;*
- $M([\alpha]\phi) = \{s \in D \ :$ $\forall t \in D((s,t) \in M(\alpha)$ *влечёт* $t \in M(\phi))\}$ *для любой PDL-программы $\alpha$.*

***Интерпретация программ:***

- *$M(a) = R(a)$ для любой программной переменной $a \in Act$;*
- *$M(\phi?) = \{(s,s) \in D^2 \ : \ s \in M(\phi)\}$ для любой PDL-формулы;*
- *$M(\alpha;\beta) = M(\alpha) \circ M(\beta)$, где '$\circ$' — композиция бинарных отношений*[8]*;*
  *$M(\alpha \cup \beta) = M(\alpha) \cup M(\beta)$, где '$\cup$' в правой части равенства — теоретико-множественное объединение бинарных отношений;*
  *$M(\alpha^*) = M(\alpha)^*$, где '$*$' в правой части равенства — рефлексивное и транзитивное замыкание бинарного отношения*[9]*.*

Определим так называемую *строгую* Пропозициональную Динамическую Логику (Strict PDL, SPDL).

**Определение 9.**

***Синтаксис*** *SPDL отличается от синтаксиса PDL только тем, что в SPDL-программах вместо недетерминированных выбора и итерации можно использовать только детерминированные выбор $if\ \phi\ then\ \alpha\ else\ \beta\ fi$ и цикл $while\ \phi\ do\ \alpha\ od$, где $\phi$ — SPDL-формула, а $\alpha$ и $\beta$ — SPDL-программы.*

***Семантика*** *SPDL в целом аналогична семантике PDL, а для детерминированных выбора и цикла определяется следующим образом:*

- $M(if\ \phi\ then\ \alpha\ else\ \beta\ fi) =$
  $= \{(s,t) \in D^2 \ :$ *если* $s \in M(\phi)$
  *то* $(s,t) \in M(\alpha)$ *иначе* $(s,t) \in M(\alpha)\}$,
- $M(while\ \phi\ do\ \alpha\ od) =$
  $= \{(s,t) \in D^2 \ :$ *существует* $n \geq 0$
  *и последовательность* $s_0, \ldots s_n \in D^n$
  *что* $s = s_0,\ t = s_n,\ t \notin M(\phi)$,
  *и* $(s_k, s_{(k+1)}) \in M(\alpha),\ s_k \in M(\phi)$
  *для любого* $k \in [0..(n-1)]\}$,

*где $M = (D, R, E)$ — произвольная модель.*

Для PDL и SPDL отношение выполнимости $\models$, выполнимость и истинность в моделях, тождественная истинность определяются аналогично EPDL. Однако для PDL (и её вариантов) определяется ещё четырёхместное отношение вход-выход между моделями, парами состояний и программами: $s\langle\alpha\rangle_M t$, если $(s,t) \in M(\alpha)$.

[8] Мы используем композицию слева направо, т.е. для бинарных отношений $P$ и $Q$ на некотором множестве $D$ имеем $P \circ Q = \{(d',d'') \in D^2 \ :$ существует $d \in D$, что $(d',d) \in P$ и $(d,d'') \in Q\}$. Напомним, что композиция бинарных отношений — ассоциативная операция, поэтому мы можем использовать выражения типа $P \circ Q \circ S$, не заботясь о расстановке скобок.

[9] Напомним, что для бинарного отношения $P$ на некотором множестве $D$ рефлексивное и транзитивное замыкание $P^*$ — это наименьшее (по включению множеств $\subseteq$) рефлексивное и транзитивное бинарное отношение, содержащее $P$; хорошо известно, что $P^* = \bigcup_{k \geq 0} P^k$, где $P^0$ — это тождественое бинарное отношение (диагональ $D$) и $P^k = \underbrace{P \circ \cdots \circ P}_{k-times}$ если $k > 0$.

## 2.3 Эквивалентность формул и программ

**Определение 10.** *Пусть $\rho$ и $\tau$ — пара однотипных синтаксических объектов (например, пара формул или пара программ и тому подобное), а $\mathbb{M}$ — класс моделей для этих объектов. Будем говорить, что $\rho$ и $\tau$ эквивалентны в этом классе $\mathbb{M}$, если $M(\phi) = M(\psi)$ для любой модели $M$ из этого класса. В частном случае, когда $\mathbb{M}$ — класс всех моделей для $\rho$ и $\tau$, $\rho$ и $\tau$ называются эквивалентными. Если же класс $\mathbb{M}$ состоит из единственной модели $M$, то говорят, что $\rho$ и $\tau$ эквивалентны в этой модели.*

Например, для любых пропозициональной и программных переменных $p \in Var$, $a, b \in Act$ следующие программы эквивалентны:

- $if\ p\ then\ a\ else\ b\ fi$ и $(p?; a\ \cup \neg p?; b)$;
- $while\ p\ do\ a\ od$ и $((p?; a)^*; \neg p?)$.

Из этого наблюдения следует, что всякая SPDL-программа и SPDL-формула эквивалентны PDL-программе и, соответственно, PDL-формуле. Более того, это наблюдение позволяет использовать в рамках PDL конструкции $if - then - else - fi$ и $while - do - od$ как макросы:

- $if\ \phi\ then\ \alpha\ else\ \beta\ fi$ — макрос для $(\phi?; \alpha\ \cup \neg\phi?; \beta)$;
- $while\ \phi\ do\ \alpha\ od$ — макрос для $((\phi?; \alpha)^*; \neg\phi?)$.

Приведём ещё несколько важных и полезных примеров эквивалентных PDL-программ.

- Недетерминированный выбор в вариантах PDL коммутативен, т.е. $M(\alpha \cup \beta) = M(\beta \cup \alpha)$ для любых программ $\alpha$ и $\beta$ и любой модели $M$.
- Последовательная композиция и недетерминированный выбор в вариантах PDL ассоциативны, т.е. $M\big((\alpha; \beta); \gamma\big) = M\big(\alpha; (\beta; \gamma)\big)$ и $M\big((\alpha \cup \beta) \cup \gamma\big) = M\big(\alpha \cup (\beta \cup \gamma)\big)$ для любых программ $\alpha$, $\beta$ и $\gamma$ и любой модели $M$.

Из этих эквивалентностей (из ассоциативности) следует, что мы можем опускать скобки в программах, которые получаются из нескольких подпрограмм в результате (только) последовательной композиции или (только) недетерминированного выбора; более того (в силу коммутативности и ассоциативности), мы можем использовать выражения типа $\bigcup_{\gamma \in \Gamma} \gamma$, где $\Gamma$ — конечное множество программ.

**Определение 11.** *Говорят, что формула находится в нормальной форме, если все случаи отрицания в этой формуле находятся на уровне литералов (т.е. $\neg$ в этой формуле применяется только к пропозициональным переменным).*

Следующее утверждение является хорошо известным обобщением законов де Моргана для модальных логик и позволяет преобразовать любую формулу любого из вариантов пропозициональной динамической логики к эквивалентной нормальной формуле того же варианта пропозициональной динамической логики за квадратичное время, но на линейной памяти.

**Утверждение 1.** *Для любой программы $\alpha$ и произвольных формул $\phi$ и $\psi$ пропозициональной динамической логики EPDL, PDL или SPDL пары формул из следующей таблицы эквивалентны в любом классе моделей:*

| | |
|---|---|
| $\neg\neg\phi$ *и* $\phi$ | |
| $\neg(\phi \wedge \psi)$ *и* $\neg\phi \vee \neg\psi$ | $\neg(\phi \vee \psi)$ *и* $\neg\phi \wedge \neg\psi$ |
| $\neg\langle\alpha\rangle\psi$ *и* $[\alpha]\neg\psi$ | $\neg[\alpha]\psi$ *и* $\langle\alpha\rangle\neg\psi$ |

## 2.4 Детерминированные, конечные и эрбрановы модели

**Определение 12.** *Детерминированная модель — это такая модель $M = (D, R, E)$, в которой всякая программная переменная $a \in Act$ интерпретируется графиком функции $R(a) : D \to D$; тотальная детерминированная модель — это детерминированная модель, в которой интерпретация всякой программной переменной — график тотальной (всюду определенной) на $D$ функции. Для любой логики $L$ (например, EPDL, PDL, SPDL) её вариант с тем же синтаксисом, что и сама логика, но с семантикой, в которой используются только (тотальные) детерминированные модели, будет называться (тотальным) детерминированным вариантом логики $L$ и обозначаться $DL$ ($TL$ соотвественно), например: DEPDL, DPDL и TSPDL.*

Заметим, что в утверждении 1 идёт речь об эквивалентности формул в любом классе моделей; следовательно, это утверждение остаётся в силе и для (тотальных) детерминированных вариантов DEPDL, DPDL, DSPDL, TEPDL, TPDL, TSPDL. Но для TEPDL имеет место еще одна важная эквивалентность: для любой программной переменной $a \in Act$ и произвольной TEPDL-формулы $\phi$ формулы $[a]\phi$ и $\langle a\rangle\phi$ эквивалентны. Поэтому для TEPDL утверждение 1 влечёт, что каждая TEPDL-формула эквивалентна некоторой нормальной формуле без модальности необходимости [...], которая может быть построена за квадратичное время на линейной памяти.

**Определение 13.** *Говорят, что формулы (возможно, разных логик) равноистинны, если все они одновременно (т.е. каждая) тождественно истинны (каждая в соответствии со своей семантикой) или одновременно все не являются тождественно истинными. Аналогично: говорят, что формулы равновыполнимы, если все они одновременно выполнимы или одновременно все не являются выполнимыми.*

Следующее утверждение было впервые доказано в [13, 38].

**Утверждение 2.** *Пусть $\phi$ — произвольная фиксированная PDL-формула. Для каждой программной переменной $a \in Act$, которая встречается в $\phi$, пусть $f_a$ и $g_a$ — новые*[10] *программные переменные, а $p_a$ — новая пропозициональная переменная (индивидуальные для каждого $a$). Пусть формула $totdet(\phi)$ получается в результате замены в $\phi$ всех вхождений каждой программной переменной $a \in Act$ на программу $(f_a^*; p_a?; g_a)$; тогда имеем: формула $\phi$ равноистинна/равновыполнима с формулой $totdet(\phi)$ тотальной детерминированной PDL (т.е. TPDL).*

*Доказательство.* Пусть даны произвольные "начальные" формула $\phi$, модель $M = (D, R, E)$ и состояние $s \in D$.

Сначала построим конечную модель $M'$, содержащую начальное состояние, такую, что $s \in M(\phi)$ тогда и только тогда, когда $s \in M'(\phi)$. Пусть $OBL$ — специальная переменная[11] для хранения конечных множеств пар вида (состояние, формула); инициализируем её как $\{(s, \phi)\}$. Теперь, пока значение переменной $OBL$ — непустое множество, выполняем следующий цикл:

- пусть $(t, \psi)$ — произвольная пара из $OBL$; удалим эту пару из $OBL$, т.е. выполним присваивание $OBL := OBL \setminus \{(t, \psi)\}$;
- если $\psi$ не является формулой BOOL, то:
  - если $\psi$ — формула вида $\neg\xi$, то добавляем к $OBL$ пару $(t, \xi)$, т.е. выполним присваивание $OBL := OBL \cup \{(t, \xi)\}$;
  - если $\psi$ — формула вида $\xi \wedge \chi$ или $\xi \vee \chi$, то добавляем к $OBL$ две пары $(t, \xi)$ и $(t, \chi)$, т.е. выполним присваивание $OBL := OBL \cup \{(t, \xi), (t, \chi)\}$;
  - если $\psi$ — формула вида $\langle\alpha\rangle\xi$ или $[\alpha]\xi$ и $t \models_M \psi$, то пусть $(t \dots v)$ — произвольная вычислительная трасса[12] $\alpha$ в $M$ такая, что $v \models_M \xi$; добавим в $OBL$ пару $(v, \xi)$ и все пары $(u, \chi)$ такие, что состояние $u$ встречается в трассе $(t \dots v)$, $\chi?$ — тест из программы $\alpha$, и в состоянии $u$ проверялся тест $\chi$;
  - если $\psi$ — это формула вида $\langle\alpha\rangle\xi$ или $[\alpha]\xi$ и $t \not\models_M \psi$, то пусть $(t \dots v)$ — произвольная вычислительная трасса $\alpha$ в $M$ такая, что $v \not\models_M \xi$; добавим в $OBL$ пару $(v, \xi)$ и все пары $(u, \chi)$ такие, что состояние $u$ встречается в трассе $(t \dots v)$, $\chi?$ — тест из программы $\alpha$, и в состоянии $u$ проверялся тест $\chi$.

Представленный алгоритм работает конечное время, так как каждый раз во время цикла мы убираем из множества $OBL$ одну пару

[10] Здесь и далее в этой статье термин "новые" означает "не использованные до сих пор и попарно различные".

[11] Аббривиатура OBL от английского OBLigations.

[12] Мы не определяли это понятие формально, но интуитивно оно совершенно ясно: это любая конечная последовательность состояний, которую "прошла" программа при её некотором (в силу недетерминизма) исполнении от начала до конца. Формально трассы для недетерминированных схем Янова определяются в тотальных детерминированных моделях далее в части 3, и, если вспомнить, что PDL-программы — это структурированные недетерминированные схемы (см. начало раздела 2.2), то определение трассы может быть легко перенесено на PDL-программы в произвольных моделях.

(состояние, формула) и (возможно) помещаем новые пары, но с более простыми формулами. Поэтому множество состояний, которые появлялись во время работы этого алгоритма, конечно. Обозначим это множество $D'$. Обозначим $R'$ следующую интерпретацию: $R'(a) = \{(t,u) \in (D')^2 \ : \ (t,u) \in R(a)\}$ для любой программной переменной $a \in Act$. Обозначим $E'$ следующее означивание: $E'(p) = \{t \in D' \ : \ t \in E(p)\}$ для любой пропозициональной переменной $p \in Var$. И, наконец, обозначим $M'$ модель $(D', R', E')$. Очевидно (может быть доказано индукцией по структуре формул), что для любой формулы $\psi$, для любого состояния $t \in D'$, если пара $(t, \psi)$ возникала во время выполнения алгоритма, то $t \in M(\psi) \ \Leftrightarrow \ t \in M'(\psi)$. В частности, для начальных состояния и формулы имеем $s \in M(\phi) \ \Leftrightarrow \ s \in M'(\phi)$.

Теперь заметим, что можно построить такую (не более чем счётную) древовидную модель $M''$ содержащую начальное состояние $s$, что $s \in M'(\phi)$ тогда и только тогда, когда $s \in M''(\phi)$. (Здесь под древовидной моделью понимается любая модель $(D'', R'', E'')$, в которой множество вершин $D''$ вместе с множеством ребер $\{R''(a) : a \in Act$ встречается в $\phi\}$ образуют ориентированное дерево; таким образом, древовидную модель можно представлять как ориентированное дерево, вершины которого помечены пропозициональными переменными из $\phi$ в соответствии с означиванием пропозициональных переменных, а ребра — программными переменными из $\phi$, в соответствии с интерпретацией этих переменных.) Для этого призведём "развёртку" конечной модели $M'$ в (возможно) бесконечное дерево путём бесконечного обхода модели как размеченного графа в ширину, начиная с начального состояния $s$, с сохранением пометок вершин и ребер. В результате развёртки конечного графа модели $M'$ получается не более чем счётная древовидная модель $M''$, корень которой — начальное состояние $s$. Очевидно (может быть доказано индукцией по структуре формул), что для любой формулы $\psi$, для любого состояния $t \in D'$ и любой копии этого состояния $t'' \in D''$ имеем $t \in M(\psi) \ \Leftrightarrow \ t'' \in M''(\psi)$. Так, для начального состояния и формулы имеем $s \in M'(\phi) \ \Leftrightarrow \ s \in M''(\phi)$.

Заметим, что если начальная модель $M$ была детерминированной моделью, то и конечная модель $M'$, и древовидная модель $M''$ будут детерминированными. Однако в случае, когда начальная модель не была детерминированной, $M'$ и (следовательно) $M''$ могут не быть детерминированными. Но мы можем "трансформировать" не более чем счётную древовидную модель $M''$ для $\phi$ в (не более чем счётную) древовидную тотальную детерминированную модель $M'''$ для $totdet(\phi)$.

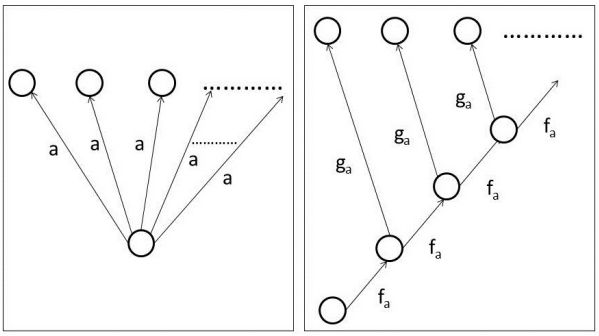


Рис. 1.

Для этого промоделируем интерпретацию каждой программной переменной $a \in Act$, подобрав тотальную детерминированную интерпретацию программных переменных $f_a$ и $g_a$ и означивание пропозициональной переменной $p_a$ так, как это показано на Рис. 1: для любого состояния (вершины дерева) $t$

- добавим в дерево счётную последовательность вспомогательных вершин $t_1, t_2, \ldots$,
- интерпретируем $f_a$ как перечисление этой последовательности, начиная с $t_0 = t$,
- означим $p_a$ истиной только на префиксе, длина которого равна числу $a$-наследников $t$,
- интерпретируем $g_a$ как переход от $t_k$ $(k > 0)$ к $k$-ому по счёту $a$-наследнику $t$ (или фиктивному состоянию, если $a$-наследников меньше $k$).

Таким образом получается (не более чем счётная) тотальная детерминированная древовидная модель $M''' = (D''', R''', E''')$. И снова очевидно (может быть доказано индукцией по структуре

формул), что для любой формулы $\psi$, для любого состояния $t \in D''$ имеем $t \in M(\psi) \Leftrightarrow t \in M'''(totdet(\psi))$. В частности, для начального состояния и формулы имеем $s \in M''(\phi) \Leftrightarrow s \in M'''(totdet(\phi))$. Утверждение доказано. □

Утверждение 2 очевидным образом влечет

**Следствие 1.** *Каждая PDL-формула равноистинна/выполнима с TPDL-формулой, которую можно построить за линейное время. Следовательно, проблема разрешимости для EPDL, PDL, SPDL, DEPDL, DPDL, DSPDL сводится к проблеме разрешимости для TPDL, причём для EPDL, PDL, DEPDL и DPDL — за линейное время.*

Приведённое следствие пригодится нам в следующей части статьи. Но первая часть доказательства утверждения 2 имеет более важное

**Следствие 2.** *Всякая формула EPDL, PDL, SPDL и их детерминированных вариантов DEPDL, DPDL и DSPDL выполнима тогда и только тогда, когда она выполнима в некоторой конечной модели. Следовательно, множества выполнимых формул EPDL, PDL, SPDL и их детерминированных вариантов DEPDL, DPDL и DSPDL рекурсивно перечислимы.*

Свойство логики, когда выполнимая формула обязательно имеет конечную модель, называется *финитной аппроксимируемостью* или *свойством конечной модели* (*finite model property*).

Следствие 2 о финитной аппроксимируемости логик EPDL, PDL, SPDL, DPDL и DSPDL доказывает только частичную разрешимость (т.е. рекурсивную перечислимость), а не разрешимость множества выполнимых формул этих логик. Исторически первой работой, установившей разрешимость PDL (и, следовательно, EPDL и SPDL) была статья [10]; метод, который использовался в цитируемой работе, был хорошо известен в неклассических логиках как *фильтрация* [6, 39]. Более тонкое использование метода фильтрации позволило установить разрешимость DPDL в работе [26], а потом повторить доказательство в работе [2].

**Определение 14.** *Эрбранова модель*[13] *— это такая древовидная модель $(D, R, E)$, в которой область $D$ — это множество $Act^*$ всех конечных слов в алфавите $Act$ (включая пустое слово, которое в этой статье мы будем обозначать $\theta$), а интерпретация $R$ — синтаксическая интерпретация $Synt$ определяется следующим образом: $Synt(a)(w) = wa$ для любой программной переменной $a \in Act$ и любого слова $w \in Act^*$.*

Очевидно, что всякая эрбранова модель является тотальной и детерминированной. Теперь, ещё раз проанализировав доказательство утверждения 2, можно доказать следующее

**Утверждение 3.** *Пусть $\phi$ — произвольная фиксированная формула EPDL, PDL, SPDL или их детерминированных вариантов DEPDL, DPDL и DSPDL. Она тождественно истинна (выполнима) тогда и только тогда, когда $totdet(\phi)$ выполнима на пустом слове $\theta$ в любой (соответственно некоторой) эрбрановой модели.*

*Доказательство.* Достаточно в модели $M'''$ переименовать каждую вершину дерева так, что вместо копии состояния (заимствованного из модели $M$) именовать эту вершину по пути в дереве от корня $s$. При таком переименовании дерево превратится в эрбранову модель, корень которого — пустое слово $\theta$. □

## 3. НЕДЕТЕРМИНИРОВАННЫЕ СХЕМЫ ЯНОВА

### 3.1 Определения

Схемы Янова [45, 46, 32] — классическая модель программ, для которой разрешимы многие важные для анализа свойства, как-то (функциональная) эквивалентность, пустота (тотальная зацикливаемость), тотальность (остановка во всех моделях) [33, 35]. Сам термин *схемы Янова* был введён в обращение в работе [32] А.П. Ершовым [40, 14]; в этой же работе [32] А.П. Ершов также предложил графическую нотацию для схем Янова в стиле блок-схем и полную и корректную графическую аксиоматику для функциональной эквивалентности схем Янова [33, 14].

[13] Термин преимущественно используется в англоязычной литературе (*Herbrand model*), в литературе по математической и прикладной логике [13, 18]. В русскоязычной литературе по теории схем программ аналогом является *свободная интерпретация* [38, 35].

Заметим, что графическое представление и аксиоматизация классических схем Янова и проблемы функциональной эквивалентности чрезвычайно наглядна и интуитивно понятна для человека. Однако более мощные результаты для функциональной эквивалентности схем программ, обобщающих классические схемы Янова, были получены на пути алгебраической интерпретации этих обобщенных схем и сведения проблемы эквивалентности к известным теоретико-групповым проблемам [30].

*Недетерминированные схемы Янова*, по-видимому, предлагались разными авторами, но нам известно только о варианте, предложенном в [13, 38] для доказательства разрешимости EPDL, PDL, SPDL и их детерминированных вариантов DEPDL, DPDL, DSPDL; затем в работах [18, 44, 20] недетерминированные схемы Янова были использованы для доказательства разрешимости $\mu$-Исчисления.

**Определение 15.** *Будем использовать в качестве меток натуральные числа (включая 0) в какой-либо фиксированной нотации*[14]. *Присваивание (или помеченный оператор присваивания) — это произвольное выражение вида "$l:\ f$ goto $L$", где $l$ — это метка (данного оператора), $f$ — программная переменная (называемая преобразователем этого оператора), а $L$ — произвольное конечное (возможно пустое) множество меток*[15]. *Выбор (или помеченный условный оператор) — это произвольное выражение вида "$l:\ if\ p\ then\ L^+\ else\ L^-$", где $l$ — метка (данного оператора), $p$ — пропозициональная переменная (называемая условием этого оператора), а $L^+$ и $L^-$ — конечные множества меток (каждое может быть пустым). Недетерминированная схема Янова — это призвольное конечное множество (помеченных) операторов.*

Как видно из этого определения, любая метка может метить сразу несколько операторов в недетерминированной схеме Янова. Классические схемы Янова можно выделить, наложив следующие синтаксические ограничения:

- любая метка может метить не более одного оператора в схеме;
- все множества $L$, $L^+$ и $L^-$ в любом из операторов схемы являются одноэлементными.

В данной статье для краткости мы резервируем термин *схема Янова* (или просто *схема*) за недетерминированными схемами Янова, а в тех случаях, когда будем говорить о классических схемах Янова, будем использовать термин *классическая схема (Янова)*. Недетерминированные схемы Янова (вслед за классическими [32]) удобно мыслить в теоретико-графовых терминах и изображать в виде блок-схем. Такой ориентированный граф получается в результате представления операторов как вершин графа (помеченных метками и преобразователями для операторов присваивания или условиями для операторов выбора), а потока управления (т.е. goto, then и else) — дугами (помеченными '+' для then-дуг и '−' для else-дуг). (Пример дан в следующем разделе 3.2.)

Семантика схем Янова определяется в тотальных детерминированных моделях. Вычисление схемы в такой модели может начинаться из любого состояния, но с оператора, помеченного "начальной" меткой 0 (ноль), затем следует цепочка срабатываний помеченных операторов с преобразованием состояния при присваиваниях и передачей управления от оператора к оператору согласно меткам, и заканчивается передачей управления любой "финальной" метке, которая не метит ни одного оператора схемы.

**Определение 16.** *Выберем произвольно и зафиксируем (в рамках данного определения) недетерминированную схему Янова $S$ и тотальную детерминированную модель $M = (D, R, E)$. Конфигурация — это призвольная пара вида (метка, состояние). Срабатывание оператора (который встречается в $S$) — произвольная (упорядоченная) пара конфигураций $(l, s)(l', s')$, удовлетворяющая следующим свойствам:*

- $s' = R(f)(s)$ *и* $l' \in L$ *при срабатывании присваивания "$l:\ f$ goto $L$";*
- $s' = s$ *и* $l' \in \begin{cases} L^+ \text{ если } s \in E(p), \\ L^- \text{ в противном случае,} \end{cases}$ *при срабатывании помеченного условного оператора "$l:\ if\ p\ then\ L^+\ else\ L^-$".*

[14] В статье, разумеется, используется десятичная запись арабскими цифрами.

[15] В статье используется стандартная математическая нотация: $\varnothing$ обозначает пустое множество, а непустое конечное множество задаётся перечислением его элементов через запятую в скобках '{' и '}'.

*Трасса — это произвольная последовательность конфигураций* $(l_0, s_0)\dots(l_k, s_k)(l_{k+1}, s_{k+1})\dots$ *такая, что каждая пара соседних конфигураций в этой последовательности* $(l_k, s_k)(l_{k+1}, s_{k+1})$ *является срабатыванием какого-либо оператора (из S). Начальная трасса — это произвольная трасса, которая начинается с конфигурации с начальной меткой 0 (ноль); финальная трасса — это произвольная трасса, которая заканчивается конфигурацией с призвольной финальной меткой (которая не метит ни одного оператора в S); полная трасса — произвольная начальная бесконечная или произвольная одновременно начальная и финальная трасса.*

Напомним некоторые известные и используемые для классических схем Янова факты и понятия. Каждая классическая схема Янова в каждой тотальной детерминированной интерпретации в каждом состоянии имеет единственную полную трассу, начинающуюся с этого состояния. Классическая схема Янова называется *тотальной*, если она не имеет бесконечных полных трасс ни в какой тотальной детерминированной модели. *Проблема тотальности* для классических схем Янова — алгоритмическая задача определения по входной схеме, является она тотальной или нет. Известно, что классическая схема Янова является тотальной тогда и только тогда, когда она останавливается (т.е.имеет конечную полную трассу) в любой эрбрановой модели начиная с конфигурации[16] $(0, \theta)$ [35].

Теперь вернёмся к недетерминированным схемам Янова.

**Определение 17.** *Пусть* $Fin \subseteq Var$ *— произвольное конечное множество пропозициональных переменных. Будем говорить, что эрбранова модель* $M = (D, R, E)$ *подходящая для* $Fin$*, если для любой из этих пропозициональных переменных* $p \in Fin$ *множество* $E(p)$ *является конечным. Говорят, что схема S тотальна относительно (или по отношению к)* $Fin$*, если в любой эрбрановой модели, подходящей для* $Fin$*, эта схема S имеет конечную полную трассу, начинающуюся с конфигурации* $(0, \theta)$*. Недетерминированная схема Янова называется тотальной, если она тотальна относительно пустого множества пропозициональных переменных.*

Заметим, что если недетерминированная схема Янова является классической схемой Янова, то только что определённое понятие тотальности совпадает с классическим понятием тотальности.

**Определение 18.** *Проблема относительной тотальности — алгоритмическая задача определения по входной схеме и конечному множеству пропозициональных переменных, является данная схема тотальной или нет относительно этого множества.*

Разрешимость проблемы относительной тотальности впервые была доказана в [13, 38]. Позже в работе [18] понятие относительной тотальности было обобщено (за счёт использования кванторов второго порядка по пропозициональным переменным), была сформулирована проблема *обобщённой тотальности*, и доказана разрешимость этой проблемы (модификацией алгоритма для относительной тотальности). Оценка сложности разрешающих процедур из работ [13, 38, 18] — $\exp(n_A + n_C)$, где $n_A$ и $n_C$ — число операторов присваивания и число операторов выбора в схеме. Обновлённое доказательство этой оценки дано в разделе 3.4. Отметим, что в работе [20] представлен новый *полиномиальный* алгоритм решения проблемы относительной тотальности для специального класса схем — так называемых *беспетельных схем.*

**Определение 19.** *Недетерминированная схема Янова называется беспетельной, если любой путь по графу её блок-схемы, начинающийся с оператора выбора и заканчивающийся в операторе выбора (возможно — том же самом) с одинаковым условием, обязательно содержит хотя бы один оператор присваивания.*

Идея этого нового алгоритма заимствована из работы [19], в которой подобный алгорим используется для проверки специального свойства автоматов на бесконечных словах и по сути является алгоритмом верфикации (model checking) специальной фиксированной формулы $\mu$-Исчисления в конечной модели, заданной

[16] Напоминаем, что $\theta$ — это пустое слово.

блок-схемой схемы Янова. Блок-схему беспетельной схемы можно рассматривать как модель, так как такая схема является *свободной* [35].

**Определение 20.** *Схема называется свободной, если любой путь по её блок-схеме реализуется в некоторой модели, т.е.: для любого пути по блок-схеме* $l_0 : op_0, \ldots l_k : op_k$ *(где* $k \geq 0$ *— произвольное число, а* $l_0 : op_0, \ldots l_k : op_k$ *— помеченные операторы из данной схемы), существует такая модель* $M$ *и последовательность состояний* $s_0, \ldots s_k$*, что* $(l_0, s_0)$, $\ldots$ $(l_k, s_k)$ *является трассой этой схемы в модели* $M$*.*

**Утверждение 4.** *Всякая беспетельная схема является свободной.*

*Доказательство.* Так как в любом пути по блок-схеме беспетельной схемы любые два вхождения одной и той же пропозициональной переменной обязательно разделены хотя бы одним присваиванием, то в любой эрбрановой модели повторная проверка одного и того же условия происходит в разных состояниях. Поэтому любой путь по блок-схеме можно подтвердить некоторой трассой в подходящей эрбрановой модели. □

Кроме семантики трасс, схемы Янова могут быть наделены семантикой, задающей в моделях бинарное отношение *вход-выход* на состояниях.

**Определение 21.** *Пусть* $S$ *— произвольная недетерминированная схема Янова, а* $M = (D, R, I)$ *— произвольная тотальная детерминированная модель. Определим бинарное отношение* $M(S) \subseteq D^2$ *вход-выход* $S$ *как*

$$\{(s,t) \in D^2 \ : \textit{существует полная трасса } S, \textit{ которая начинается в состоянии } s, \textit{ а заканчивается в состоянии } t\}.$$

Таким образом, семантика вход-выход не принимает во внимание бесконечные вычисления, или вычисления, "заканчивающиеся" передачей управления (в результате *goto*-, *then*- или *else*-перехода) на пустое множество меток. А так как для схем определена семантика вход-выход в моделях, то мы можем воспользоваться определением эвивалентности 10.

**Определение 22.** *Схемы называются функционально эквивалентными в некотором классе (тотальных детерминированных) моделей* $\mathbb{M}$*, если в любой модели из этого класса семантики вход-выход этих схем совпадают. Схемы называются функционально эквивалентными, если они функционально эквивалентны в классе всех тотальных детерминированных моделей.*

Название *функциональная эквивалентность* выбрано потому, что для любой классической схемы Янова $S$ в любой (тотальной детерминированной) модели $M$ отношение $M(S)$ — график частичной функции на $D_M$; следовательно, функциональная эквивалентность классических схем Янова в модели — это экстенсиональное равенство функций, "вычисляемых" по схемам в этой модели, а сама функциональная эквивалентность классических схем Янова — равенство соотвествующих функций во всех моделях. Проблема функциональной эквивалентности для классических схем Янова — алгоритмическая задача определения по любой паре таких схем, являются они эквивалентными или нет. Как уже было сказано, эта проблема была решена ещё в работах Ю.И. Янова [45, 46], корректная и полная графическая аксиоматизация для функциональной эквивалентности была построена А.П. Ершовым [32, 33], а в работе [30] получены хорошие верхние оценки сложности для проблемы функциональной эквивалентности для детерминированных моделей программ, обобщающих классические схемы Янова.

## 3.2 Пример

Поясним данные выше определения на примере классической и недетерминированной схем Янова.

Начнём с построения по программе моделирующей схемы Янова такой, чтобы, анализируя свойства схемы, сделать полезные выводы об исходной программе. Рассмотрим следующую вычислительную программу Prod(a,b) (представим её на паскалеподобном псевдокоде, подразумевая стандартную интерпретацию всех операций[17]):

```
const a, b: integer;
var x=a, y=b, z=0: integer;
```

[17] В этой программе odd — проверка на нечётность.

```
while y>0
      do if odd(y) then z:=z+x fi ;
         x:=2*x; y:=y/2;
      od
```

Можно показать истинность следующего утверждения тотальной корректности: `[b>=0] Prod(a,b) [z=a*b]`, (т.е. что для любого значения параметра a и любого неотрицательного значения параметра b эта программа Prod(a,b) останавливается со значением z=a*b).

Применим к этой программе (предварительно представив цикл $while-do$ в терминах $if-then-else$ и $goto$) так называемую *булевскую абстракцию* (boolean abstarction) [1]:

> The idea of Boolean abstraction is to map "concrete" states to "abstract" states according their evaluation under a finite set of predicates ("Boolean expressions"). The predicates induce an "abstract" system with a transition relation over the abstract states.

В результате получим следующую классическую схему Янова $S_{cl}$, представленную на Рис. 2 в синтаксической и графической нотации. Здесь $f_{x:=a}$, $f_{y:=b}$, $f_{z:=0}$, $f_{z:=z+x}$, $f_{x:=2*x}$ и $f_{y:=y/2}$ — различные программные переменные (мнемонически помеченные присваиваниями, которым они соответствуют в программе), а $p_{y>0}$ и $p_{odd(y)}$ — различные пропозициональные переменные (мнемонически помеченные предикатами, которым они соответствуют в программе); единственная финальная метка $S_{cl}$ — метка 8. (В графическом представлении финальные метки соответствуют висящим вершинам, представленным овалами с меткой внутри.)

Очевидно, что $S_{cl}$ является беспетельной и, следовательно, свободной. Но эта схема не является тотальной: достаточно взять, например, модель, в которой пропозициональная переменная $p_{y>0}$ означивается тождественной истиной (т.е. предикатом, истинным на всех состояниях). Однако, если принять в качестве $Fin$ множество $\{p_{y>0}\}$, то $S_{cl}$ тотальна относительно этого множества, так как эта схема является беспетельной, и, следовательно, ни в какой подходящей для $Fin$ эрбрановой модели мы не можем бесконечно часто в операторе, помеченном меткой 3, выбирать then-переход и зацикливаться.

$\{\ 0:\ f_{x:=a}\ goto\ \{1\}$
$1:\ f_{y:=b}\ goto\ \{2\}$
$2:\ f_{z:=0}\ goto\ \{3\}$
$3:\ if\ p_{y>0}\ then\ \{4\}\ else\ \{8\}$
$4:\ if\ p_{odd(y)}\ then\ \{5\}\ else\ \{6\}$
$5:\ f_{z:=z+x}\ goto\ \{6\}$
$6:\ f_{x:=2*x}\ goto\ \{7\}$
$7:\ f_{y:=y/2}\ goto\ \{3\}\}$

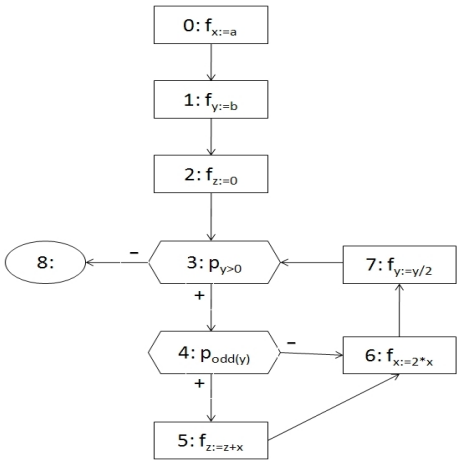


Рис. 2.

На Рис. 3 дан пример недетерминированной схемы Янова $S_{nd}$ (также в синтаксической и графической форме), искусственно построенный на базе схемы $S_{cl}$: в $S_{nd}$ по сравнению с $S_{cl}$ добавлена возможность зацикливаться на операторах выбора с метками 3 и 4. (Поэтому схемы $S_{cl}$ и $S_{nd}$ являются функционально эквивалентными.) Эта схема $S_{nd}$ не является беспетельной, так как её блок-схема имеет циклы, состоящие из операторов выбора. Однако она всё равно является свободной, так как любая передача управления между операторами по then-ветви имеет аналог по else-ветви и наоборот. Как и $S_{cl}$, эта схема не является тотальной, но она тотальна относительно $Fin = \{p_{y>0}\}$, так как в любой подходящей для $Fin$ эрбрановой модели всегда можно выбрать ту из ветвей недетерминированной передачи управления, которая ведёт к оператору присваивания.

$\{\ 0:\ f_{x:=a}\ goto\ \{1\}$
$1:\ f_{y:=b}\ goto\ \{2\}$
$2:\ f_{z:=0}\ goto\ \{3\}$
$3:\ if\ p_{y>0}\ then\ \{3,4\}\ else\ \{3,8\}$
$4:\ if\ p_{odd(y)}\ then\ \{3,4,5\}\ else\ \{3,4,6\}$
$5:\ f_{z:=z+x}\ goto\ \{6\}$
$6:\ f_{x:=2*x}\ goto\ \{7\}$
$7:\ f_{y:=y/2}\ goto\ \{3\}\}$

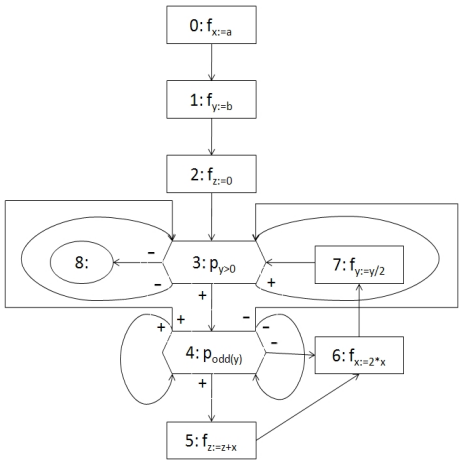


Рис. 3.

### 3.3 Относительная тотальность и разрешимость вариантов PDL

В этом разделе мы покажем, как проблемы разрешимости для DEPDL и EPDL сводятся к проблеме относительной тотальности для недетерминированных схем Янова. Алгоритм сведения проблемы разрешимости для других вариантов PDL — а именно SPDL, DPDL и DSPDL — можно найти в [13, 38]. В этих же работах можно найти вариант разрешающей процедуры для проверки свойства относительной тотальности недетерминированных схем Янова, а в работе [18] — разрешающую процедуру для проблемы обобщённой тотальности. Для удобства и краткости алгоритмы сведения мы будем называть трансляцией логики в схемы.

В алгоритмах трансляции в данной статье будем использовать следующие макросы для стандартных структурированных конструкций и организации потока управления для схем Янова:

- *skip* для фиксированной схемы, которая всегда останавливается, не изменив состояние (например, $\{0:\ if\ p\ then\ \{1\}\ else\ \{1\}\}$);
- *excp* для фиксированной схемы, которая не имеет ни одной трассы[18] (например, $\{0:\ if\ p\ then\ \varnothing\ else\ \varnothing\}$);
- *loop* для фиксированной схемы, которая всегда зацикливается (например, $\{0:\ if\ p\ then\ \{0\}\ else\ \{0\}\}$);
- для любой программной переменной $a \in Act$ схема $(a)$ — это $\{0:\ a\ goto\ \{1\}\}$;
- $S'; S''$ для последовательной композиции двух схем;
- $if\ q\ then\ S'\ else\ S''\ fi$ для детерминированного выбора одной из двух схем;
- $S' \cup S''$ для недетерминированного выбора одной из двух схем;
- $S^*$ для недетерминированной итерации схемы;
- $while\ q\ do\ S\ od$ для детерминированного цикла из схемы;

все перечисленные средства организации потока управления могут быть легко определены в терминах недетерминированных схем Янова. Кроме того, мы будем использовать метапеременную *stop* как макрос финальных меток схемы в контексте места использования: переход (передача управления) на *stop* (через *goto*, *then* или *else*), будет означать завершение работы (передачу управления на финальную метку).

Начнём с трансляции DEPDL. Заметим, что в этом случае нам не нужна вся сила утверждения 2, мы можем обойтись более эффективным преобразованием формул DPDL в формулы TEPDL (т.е. тотальной версии EPDL).

Пусть $\phi$ — произвольная формула DEPDL. Пусть (как и в утверждении 2) для каждой программной переменной $a \in Act$, которая встречается в $\phi$, $g_a$ и $p_a$ — новые программная и пропозициональная переменные (индивидуальные для каждого $a$). Пусть формула $\psi$ получается в результате замены в $\phi$ всех вхождений каждой программной переменной $a \in Act$ на програм-

[18] *excp* — аббривиатура от *exception*.

му $(p_a?;g_a)$. Тогда[19] DEPDL-формула $\phi$ равноистинна/равновыполнима с TPDL-формулой $\psi$, в которой все программы имеют вид $(q?;c)$, где $q$ и $c$ — некоторые пропозициональная и программная переменные. Но для любых PDL-формул $\xi$ и $\chi$ и PDL-программы $\alpha$ имеют место следующие две эквивалентности:

- $[\xi?;\alpha]\chi$ эквивалентна $\xi \to [\alpha]\chi$,
- $\langle\xi?;\alpha\rangle\chi$ эквивалентна $\xi \wedge \langle\alpha\rangle\chi$.

Кроме того, как уже было отмечено в разделе 2.3, каждая TEPDL-формула эквивалентна некоторой нормальной формуле без модальности необходимости. Поэтому мы можем считать, что формула $\psi$ — это формула тотальной детерминированной элементарной пропозициональной динамической логики TEPDL в нормальной форме и без модальности необходимости (эта формула $\psi$ может быть построена за квадратичное время на линейной памяти по исходной формуле $\phi$).

Трансляция нормальных TEPDL-формул без модальности необходимости в недетерминированные схемы Янова осуществляется описанным ниже рекурсивным алгоритмом $TE2S$.

- $TE2S(true) = skip$ и $TE2S(false) = excp$;
- для любой пропозициональной переменной $p \in Var$
  - $TE2S(p) = \{0 :\ if\ p\ then\ stop\ else\ excp\}$;
  - $TE2S(\neg p) = \{0 :\ if\ p\ then\ excp\ else\ stop\}$;
- $TE2S(\phi \wedge \psi) = if\ q\ then\ TE2S(\phi)\ else\ TE2S(\psi)\ fi$, где $q \in Var$ — новая пропозициональная переменная;
- $TE2S(\phi \vee \psi) = TE2S(\phi) \cup TE2S(\psi)$;
- $TE2S(\langle a\rangle\phi) = (a)\ ;\ TE2S(\phi)$ для любой программной переменной $a \in Act$.

Из описания алгоритма $TE2S$ следует, что он за квадратичное время, но на линейной памяти транслирует TEPDL-формулы (без модальности необходимости) в свободные недетерминированные схемы Янова (в которых вообще нет циклов).

**Определение 23.** *Пусть $M = (D, R, E)$ и $M' = (D', R', E')$ — две произвольные модели с одной и той же областью (т.е. $D = D'$) и совпадающей интерпретацией программных переменных (т.е. $R = R'$), а $Fin$ — множество пропозициональных переменных. Будем говорить, что $M'$ является модификацией $M$ на $Fin$ (и писать $M' = M \mod Fin$), если означивания $E$ и $E'$ совпадают на всех пропозициональных переменных кроме (может быть) $Fin$ (т.е. $E(p) = E'(p)$ для каждой $p \notin Fin$).*

**Утверждение 5.** *Пусть $M$ — произвольная эрбранова модель, $\omega \in Act^*$ — произвольное состояние этой модели, $\phi$ — произвольная нормальная TEPDL-формула без модальности необходимости, а $Cnj$ — множество всех новых пропозициональных переменных $q$, которые были использованы при выполнении алгоритма $TE2S(\phi)$ при трансляции конъюнкций. Имеем: $\omega \models_M \phi$ тогда и только тогда, когда недетерминированная схема Янова $TE2S(\phi)$ имеет конечную полную трассу, начинающуюся с состояния $\omega$, в любой модели $M'$, которая является модификацией $M$ на $Cnj$.*

*Доказательство.* Индукция по структуре формул. База индукции — формулы $true$, $false$ или литералы (пропозициональные переменные или их отрицания): утверждение очевидно по явному построению схем по этим формулам. На шаге индукции надо рассмотреть формулы, построенные при помощи конъюнкции $\wedge$, дизъюнкции $\vee$ и модальности возможности $\langle\ \rangle$. Случай дизъюнкции и модальности являются тривиальными. Рассмотрим подробнее случай конъюнкции. Поскольку $TE2S(\phi \wedge \psi) = if\ q\ then\ TE2S(\phi)\ else\ TE2S(\psi)$, где $q$ — новая (не используемая нигде более) программная переменная, вошедшая в $Cnj$, то эта переменная может быть означена на $\omega$ произвольным образом, что позволяет

[19] В случае DEPDL нам не нужна ещё одна новая программная переменная $f_a$ для каждой программной переменной $a \in Act$, которая встречалась в $\phi$: вместо программы $(f_a^*;p_a?;g_a)$ мы теперь использует программу $(p_a?;g_a)$; так как в DEPDL интерпретация всех программных переменных — графики частичных функций, то означивание новых пропозициональных переменных позволяет моделировать область определённости программных переменных из $\psi$, а интерпретация новых программных переменных — интерпретацию прежних программных переменных посредством их тотальных расширений.

"выбрать" любую из схем $TE2S(\phi)$ и $TE2S(\psi)$ для "проверки" на существование конечной полной трассы, начинающейся с $\omega$. □

**Следствие 3.** *По произвольной формуле DEPDL за квадратичное время на линейной памяти можно построить такую беспетельную недетерминированную схему Янова, которая тотальна тогда и только тогда, когда исходная формула тождественно истинна.*

Проиллюстрируем метод трансляции DEPDL в схемы на следующих двух формулах:

- $\langle a\rangle p \wedge [a]\neg p$;
- $\langle a\rangle p \vee [a]\neg p$.

(Первая является тождественно ложной, а вторая — тождественно истинной.)

Для первой формулы имеем следующее. После преобразования её в равноистинную нормальную формулу TEPDL без модальности необходимости получается формула

$$(p_a \wedge \langle g_a\rangle p) \wedge (\neg p_a \vee \langle g_a\rangle \neg p).$$

Если применить к этой формуле алгоритм $TE2S$, получится схема на Рис. 4, в которой $q_1$ и $q_2$ — новые пропозициональные переменные, соответствующие конъюнкциям в этой формуле, а висячие дуги соотвествуют *excp* (т.е. передаче управления по пустому множеству меток). Очевидно, что эта схема не является тотальной: например, в эрбрановой модели $M = (Act^*, R, E)$ такой, что $E(q_1) = E(q_2) = \{\theta\}$ и $E(p_a) = \varnothing$, схема не имеет конечной полной трассы.

Для второй формулы имеем следующее. После преобразования её в равноистинную нормальную формулу TEPDL без модальности необходимости получается формула

$$(p_a \wedge \langle g_a\rangle p) \vee (\neg p_a \vee \langle g_a\rangle \neg p).$$

Если применить к этой формуле алгоритм $TE2S$, получится схема на Рис. 5, в которой $q$ — новая пропозициональная переменная, соответствующая конъюнкции в этой формуле. На этом рисунке (и далее) звезда — помеченный "оператор" *безусловного недетерминированного выбора* (результат трансляции

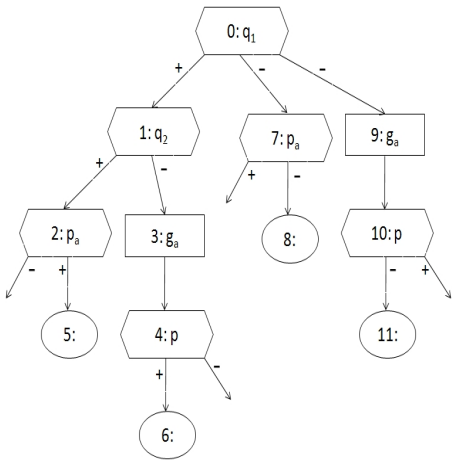


Рис. 4.

конструкции ∪). В данном случае он может быть реализован, например, оператором выбора $0 : \ if\ p\ then\ \{1,7,9\}\ else\ \{1,7,9\}$. В общем случае оператор безусловного недетерминированного выбора из множества меток $L$, помеченный меткой $l$, может быть реализован оператором выбора $l : \ if\ x\ then\ L\ else\ L$, где $x$ — произвольная новая пропозициональная переменная. Легко видеть, что схема на Рис. 5 является тотальной в силу возможности недетерминированно выбрать в любой модели ту ветвь схемы, которая в этой модели порождает конечную полную трассу.

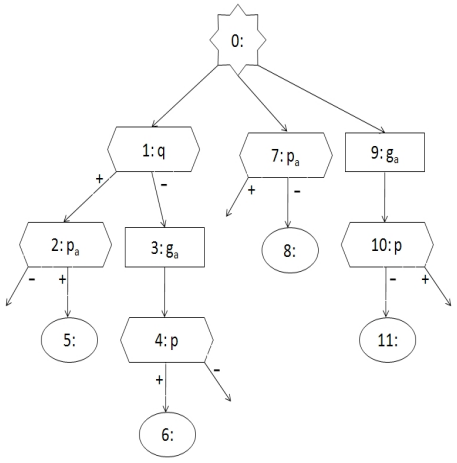


Рис. 5.

Теперь рассмотрим трансляцию EPDL.

В этом случае упростить преобразование[20] формул EPDL к формулам TPDL нельзя, приходится использовать само преобразование *totdet*. Однако, в результате применения этого преобразования получаются формулы TPDL, в которых внутри модальностей все программы имеют один и тот же вид $(f^*;p?;g)$, где $f$, $g$ и $p$ — программные и пропозициональные переменные (индивидуальные для каждой программной переменной из исходной формулы EPDL). Обозначим множество всех таких формул TPDL через *totdet*-EPDL.

Трансляция нормальных формул *totdet*-EPDL в недетерминированные схемы Янова осуществляется описанным ниже рекурсивным алгоритмом $EPDL2S$.

- Для констант $true$, $false$, литералов, конъюнкции и дизъюнкции — аналогично алгоритму $TE2S$;
- $EPDL2S([f^*;p?;g]\phi) =$
  $= \ while\ r\ do\ (f)\ od\ ;$
  $if\ p\ then\ skip\ else\ stop\ fi\ ;$
  $(g)\ ;\ EPDL2S(\phi),$
  где $r \in Var$ — новая пропозициональная переменная;
- $EPDL2S(\langle f^*;p?;g\rangle\phi) =$
  $= \ (f)^*\ ;\ if\ p\ then\ skip\ else\ excp\ ;$
  $(g)\ ;\ EPDL2S(\phi).$

**Утверждение 6.** *Пусть $M$ — произвольная эрбранова модель, $\omega \in Act^*$ — произвольное состояние этой модели, $\phi$ — произвольная нормальная формула totdet-EPDL, $Box$ — множество всех новых пропозициональных переменных $r$, которые были использованы при выполнении алгоритма $EPDL2S(\phi)$ при трансляции модальностей необходимости $[\ldots]$, а $Cnj$ — множество всех новых пропозициональных переменных $q$, которые были использованы при выполнении алгоритма $EPDL2S(\phi)$ при трансляции конъюнкций. Имеем: $\omega \models_M \phi$ тогда и только тогда, когда недетерминированная схема Янова $EPDL2S(\phi)$ имеет конечную полную трассу, начинающуюся с состояния $\omega$ в любой модели $M'$, подходящей для $Box$, которая является модификацией $M$ на $(Box \cup Cnj)$.*

*Доказательство.* Индукция по структуре формул. База индукции и шаг индукции для конъюнкции и дизъюнкции — аналогично доказательству утверждения 5. Остаётся обосновать шаг индукции для модальностей возможности и необходимости.

Для модальности возможности доказательство следует из очевидного равенства
$M(f^*;p?;g) =$
$= \ M\big((f)^*\ ;\ if\ p\ then\ skip\ else\ excp\ fi\ ;\ (g)\big),$
а для модальности необходимости — из равенств
$M(f^*;p?;g) \ = \ \bigcup_{k\geq 0} M((f)^k;p?;g) \ =$
$= \ \bigcup_{k\geq 0} M\Big(for\ i = 0\ to\ k\ do\ (f)\ od\ ;$
$if\ p\ then\ skip\ else\ excp\ fi\ ;\ (g)\Big) \ =$

$$= \bigcup_{\substack{P \overset{fin}{\subset} Act^*,\\ M' = M \mod\{r\},\\ M'(r) = P}} M' \left( \begin{array}{r} while\ r\ do\ (f)\ od\ ; \\ if\ p\ then\ skip\ fi \\ else\ excp\ ; \\ (g) \end{array} \right).$$

Остаётся заметить только, что в последней else-ветви стоит $excp$, который при трансляции модальности [ ] в формулах будет заменяться на $stop$. □

Данное утверждение вместе с утверждениями 1, 2(следствие 1) и 3 влекут

**Следствие 4.** *По произвольной формуле EPDL за квадратичное время на линейной памяти можно построить такую беспетельную недетерминированную схему Янова и конечное множество пропозициональных переменных, которая тотальна относительно этого множества тогда и только тогда, когда исходная формула тождественно истинна.*

Проиллюстрируем метод трансляции EPDL в схемы на следующих двух формулах:

- $\langle a\rangle p \wedge \langle a\rangle \neg p$;
- $\langle a\rangle p \vee [a]\neg p$.

(Первая является выполнимой, но не является тождественно истинной, а вторая является тождественно истинной.)

Для первой формулы имеем следующее. После преобразования её в равноистинную нормальную

[20] Имеется в виду преобразование из утверждения 2 при сведении проблемы разрешимости EPDL к проблеме разрешимости TPDL.

формулу TPDL получается формула

$$\langle (f_a^*; p_a?; g_a\rangle p \wedge \langle (f_a^*; p_a?; g_a\rangle \neg p.$$

Если применить к этой формуле алгоритм *EPDL2S*, то получится схема, представленная на Рис. 6, в которой $q$ — новая пропозициональная переменная, соответствующая конъюнкции в этой формуле. Так как в данном случае множество *Box* пусто, то относительная тотальность превращается в тотальность; схема на Рис. 6 не является тотальной, так как она свободна и имеет циклы по блок-схеме.

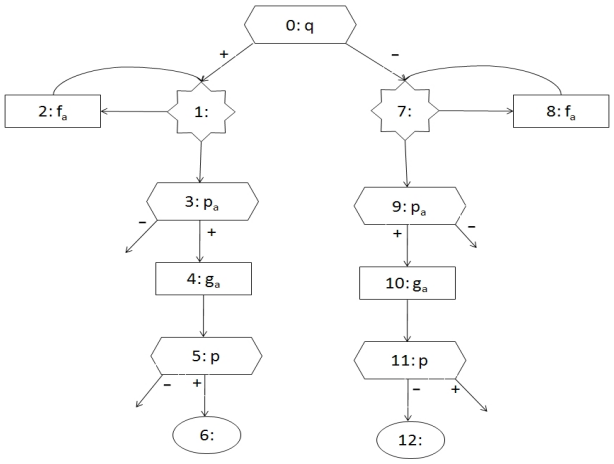


Рис. 6.

Для второй формулы после преобразования её в равноистинную нормальную формулу TPDL получается формула

$$\langle f_a^*; p_a?; g_a\rangle p \vee [f_a^*; p_a?^*; g_a]\neg p.$$

В результате применения к этой формуле алгоритма *EPDL2S* получится схема, представленная на Рис. 7, в которой $r$ — новая пропозициональная переменная, соответствующая модальности необходимости $[\dots]$ в этой формуле. В данном случае множество *Box* состоит из единственного символа $r$, тотальность этой схемы относительно $Box = \{r\}$ является очевидной (так как она является беспетельной и, следовательно, свободной).

Алгоритмы трансляции PDL, SPDL, DPDL и DSPDL в схемы, аналогичные алгоритму *EPDL2S* по сути и по сложности, могут быть найдены в работах [13, 38] (однако в результате их применения могут получиться не беспетельные и не свободные схемы).

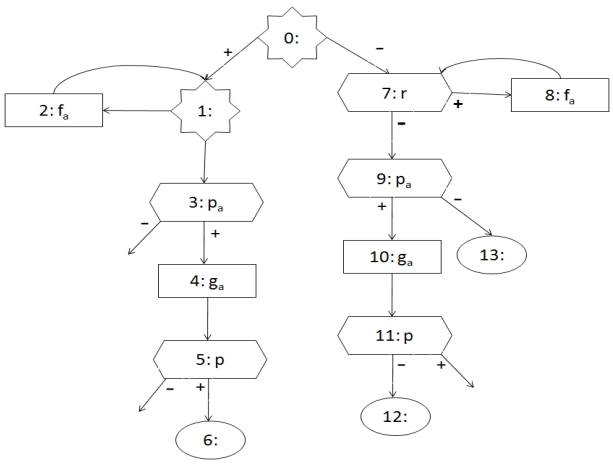


Рис. 7.

### 3.4 Разрешимость проблемы относительной тотальности

Как уже было сказано ранее, экспоненциальный алгоритм проверки относительной тотальности можно найти в работах [13, 38]. В этом разделе мы дадим обновлённую и исправленную версию этого алгоритма.

Для удобства изложения выберем и зафиксируем вплоть до следствия 5 произвольную недетерминированную схему Янова $S$, в которой каждая метка метит не более одного оператора. Также выберем и зафиксируем произвольное конечное множество $\{p_1, \dots p_m\} \subseteq Var$ $(m \geq 0)$ пропозициональных переменных, относительно которых мы хотим проверить тотальность схемы $S$. Кроме того, пусть $\{q_1, \dots q_n\} \subseteq Var$ $(n \geq 0)$ — множество всех пропозициональных переменных, которые встречаются в $S$, но отличны от $\{p_1, \dots p_m\}$. Таким образом, $p_1, \dots p_m, q_1, \dots q_n$ — это множество всех пропозициональных переменных, которые встречаются в $S$.

Определим отношение однотипности для меток операторов присваивания и/или финальных меток схемы $S$: произвольные метки операторов присваивания $l'$ и $l''$ являются однотипными, если они метят операторы присваивания с одинаковой программной переменной[21]; любая финальная метка является однотипной для любой метки оператора присваивания.

[21] т.е. для некоторой программной переменной $a \in Act$ и множеств меток $L'$, $L''$, операторы $(l' :\ a\ goto\ L') \in S$ и $(l'' :\ a\ goto\ L'') \in S$

Для любого булевского вектора $(u_1 \dots u_{m+n}) \in \{0,1\}^{m+n}$ (длиной $(m+n)$) определим на множестве меток, которые метят в $S$ операторы присваивания или являются финальными, бинарное отношение $\overset{u_1 \dots u_{m+n}}{\rightsquigarrow}$ следующим образом: для произвольных меток $l'$ и $l''$ имеет место $l' \overset{u_1 \dots u_{m+n}}{\rightsquigarrow} l''$, когда существует путь от оператора, помеченного меткой $l'$ до метки $l''$, проходящий только через условные операторы и согласованный с оценкой пропозициональных переменных $\{p_1, \dots p_m, q_1, \dots q_n\}$, заданной вектором $(u_1 \dots u_{m+n})$. Формально: метка $l'$ метит в $S$ некоторый оператор присваивания $l' :$ *a goto* $L$ ($a \in Act$ — произвольная программная переменная) и существует $k \geq 0$ и последовательность меток $l_1, \dots l_{k+1}$ такие, что $l_{k+1} = l''$, а метки $l_1, \dots l_k$ метят в $S$ операторы выбора $(l_1 :$ *if* $r_1$ *then* $L_1^+$ *else* $L_1^-)$, $\dots$ $(l_k :$ *if* $r_k$ *then* $L_k^+$ *else* $L_k^-)$ и для любого $1 \leq i \leq k$ имеем

$$l_{i+1} \in \begin{cases} L_i^+, \text{ если } r_i \equiv p_j \text{ и } u_j = 1 \\ \qquad \text{или } r_i \equiv q_j \text{ и } u_{m+j} = 1 \\ L_i^-, \text{ если } r_i \equiv p_j \text{ и } u_j = 0 \\ \qquad \text{или } r_i \equiv q_j \text{ и } u_{m+j} = 0 \end{cases}.$$

Определим множество *LabSpc* как[22] $\{L : L -$ множество однотипных меток из $S\}$, т.е. как множество всех множеств однотипных меток из $S$. Для любого булевского вектора $(u_1 \dots u_{m+n}) \in \{0,1\}^{m+n}$ (длиной $(m+n)$) распространим на *LabSpc* бинарное отношение $\overset{u_1 \dots u_{m+n}}{\rightsquigarrow}$ следующим образом: для произвольных $L', L'' \in$ *LabSpc* имеем $L' \overset{u_1 \dots u_{m+n}}{\rightsquigarrow} L''$, когда для любой метки $l'' \in L''$ существует метка $l' \in L'$ такая, что $l' \overset{u_1 \dots u_{m+n}}{\rightsquigarrow} l''$.

Теперь мы готовы описать базовый алгоритм *Check*, результат работы которого позволяет проверить тотальность недетерминированной схемы Янова относительно множества пропозициональных переменных. Мы опишем применение этого алгоритма к фиксированным выше множеству пропозициональных переменных $\{p_1, \dots p_m\}$ и схеме $S$, (в которой всякая метка метит не более одного оператора, а все пропозициональные используемые переменные — $p_1, \dots p_m, q_1, \dots q_n$). Алгоритм *Check* состоит из двух этапов. В описание этого алгоритма мы сразу введём и будем использовать две вспомогательные целочисленные переменные и операторы инкрементации их значений; эти переменные никак не влияют на работу алгоритма, но используются в его спецификации и верификации.

**Пусть** $X$ и $Y$ — две переменных для подмножеств *LabSpc* (т.е. их значения — конечные множества множеств однотипных меток), а $i$ и $j$ — две вспомогательные целочисленные переменные.

**Первый этап:**

Инициализируем $X$ множеством $\{L \in$ *LabSpc* $: L$ содержит финальные метки$\}$, а переменную $i$ — нулём;
повторять
$Y := X$ ; $i := i + 1$ ;
$X := \{L' \in$ *LabSpc* $:$
для любого булевского вектора $(w_1 \dots w_n) \in \{0,1\}^n$
существует $L'' \in Y$ такое,
что $L' \overset{0 \dots 0 \; w_1 \dots w_n}{\rightsquigarrow} L''$ $\}$
пока $X \neq Y$.

**Второй этап:**

Инициализируем переменную $j$ нулём;
повторять
$Y := X$ ; $j := j + 1$ ;
$X := \{L' \in Y$ :
для любого булевского вектора $(v_1 \dots v_m w_1 \dots w_n) \in \{0,1\}^{m+n}$
существует $L'' \in Y$ такое,
что $L' \overset{v_1 \dots v_m w_1 \dots w_n}{\rightsquigarrow} L''$ $\}$
пока $X \neq Y$.

**Утверждение 7.** *Если в схеме все операторы помечены разными метками, а* $n_A$ *— число операторов присваивания в схеме* $S$*, то алгоритм Check завершается, выполнив не более* $2^{n_A}$ *итераций цикла на каждом из двух этапов, причем по завершении алгоритма для заключительного значения переменной* $X$ *имеет место следующее равенство:*

$X = \{L \in$ *LabSpc* $:$
*для любой эрбрановой модели* $M$
*, подходящей для* $p_1, \dots p_m$
*существует* $l \in L$ *такая, что*
*в* $M$ *существует конечная финальная трасса,*
*начинающаяся с* $(l, \theta)$ $\}$.

[22] *LabSpc* — аббривиатура от *Label Space*

*Доказательство.* Заметим, что на первом этапе значения переменной $X$ после каждой итерации цикла монотонно не убывают (по включению $\subseteq$), а на втором этапе — наоборот, после каждой итерации цикла монотонно не возрастают; кроме того, эти значения все время являются подмножествами $LabSpc$; следовательно, каждый цикл не может быть итерирован более $|LabSpc| \leq 2^{n_A}$ раз. Таким образом, завершаемость алгоритма доказана.

Теперь докажем частичную корректность алгоритма методом индуктивных утверждений Флойда [37, 42]. Для этого представим множество всех эрбрановых моделей $\mathbb{F}$, подходящих для $p_1, \dots p_m$, в виде $\bigcup_{j\geq 0} \mathbb{F}_j$, где $\mathbb{F}_i = \{M = (Act^*, Synt, E) - $ эрбранова модель $: E(p_k) \subseteq \{t \in Act^* : |t| \leq j\}$ для любого $1 \leq k \leq m\}$ (т.е. множество тех эрбрановых моделей, в которых все $p_1, \dots p_m$ ложны на термах, длина которых превосходит $j$).

Тогда инвариантом первого цикла является следующее утверждение[23]:

$$X = \{L \in LabSpc \ : \text{для любой } M \in \mathbb{F}_0 \text{ существует } l \in L \text{ и финальная трасса, которая начинается с } (l, \theta) \text{ и содержит не более } i \text{ присваиваний}^{24}\}.$$

В силу этого инварианта и условия выхода из первого цикла (означающего, что на последней итерации цикла значение переменной $X$ не изменилось) и в силу леммы Кёнига (аналогично алгоритму проверки тотальности стандартных схем программ [35]), после завершения первого этапа имеем:

$$X = \{L \in LabSpc \ : \text{для любой } M \in \mathbb{F}_0 \text{ существует } l \in L \text{ и финальная трасса, которая начинается с } (l, \theta)\}.$$

Тогда инвариантом второго цикла является следующее утверждение[25]:

$$X = \{L \in LabSpc \ : \text{для любой } M \in \mathbb{F}_j \text{ существует } l \in L \text{ и финальная трасса, которая начинается с } (l, \theta)\}.$$

[23] Обратите внимание на использование вспомогательной переменной $i$ в этом инварианте.

[24] точнее: содержащая не более $i$ вхождений меток операторов присваивания из $S$

[25] Обратите внимание на использование вспомогательной переменной $j$ в этом инварианте.

Так как $\mathbb{F} = \bigcup_{j\geq 0} \mathbb{F}_j$, то из этого инварианта следует наше утверждение. □

**Следствие 5.** *Проблема относительной тотальности для недетеримнированных схем Янова разрешима с верхней оценкой сложности* $\exp(n_A + n_C)$*, где* $n_A$ *и* $n_C$ *— число операторов присваивания и условных операторов схемы.*

*Доказательство.* Пусть $S$ — произвольная недетерминированная схема Янова, а $Fin \subseteq Var$ — произвольное конечное множество пропозициональных переменных. Простое переименование меток, метящих операторы, позволяет по схеме $S$ построить функционально эквивалентную схему $S'$, в которой любые два оператора помечены разными метками. Далее, заметим, что для произвольной программной переменной $a \in Act$, схема $S'$ тотальна относительно $Fin$ тогда и только тогда, когда схема $S'' \equiv \big((a); S'\big)$ тотальна относительно $Fin$. Но схема $S''$ удовлетворяет условиям утверждения 7, и, кроме того, начальная метка 0 метит в ней оператор присваивания. Поэтому, если применить к этой схеме $S''$ алгоритм $Check$, то имеем следующее.

Во-первых, в силу утверждения 7 сложность этого алгоритма $\exp(n_A) \times 2^{(}n_C)$, где множитель $2^{(}n_C)$ возникает за счёт перебора всех булевских векторов $(v_1 \dots v_m w_1 \dots w_n) \in \{0,1\}^{m+n}$, а $n + m \leq n_C$.

Во-вторых, если $LastX \subseteq LabSpc$ — значение переменной $X$ по завершении алгоритма, то в силу утверждения 7

$$\{0\} \in LastX \Leftrightarrow$$
$$\Leftrightarrow \text{для любой подходящей для } Fin \text{ эрбрановой модели } M \text{ существует } l \in \{0\} \text{ и финальная трасса } S'', \text{ начинающаяся с } (l, \theta) \Leftrightarrow$$
$$\Leftrightarrow \text{в любой подходящей для } Fin \text{ эрбрановой модели } M \text{ существует конечная полная трасса, начинающаяся с } (0, \theta) \Leftrightarrow$$
$$\Leftrightarrow S'' \text{ тотальна относительно } Fin \Leftrightarrow$$
$$\Leftrightarrow S \text{ тотальна относительно } Fin.$$

□

В конце предыдущего раздела 3.3 уже было сказано, что алгоритмы трансляции PDL, SPDL, DPDL и DSPDL в схемы аналогичны алгоритму $EPDL2S$; поэтому разрешимость с экспоненциалной верхней оценкой сложности PDL, SPDL,

DPDL и DSPDL следует из существования этих алгоритмов трансляции и следствия 5 о разрешимости проблемы относительной тотальности. Но формально в этой статье (на основании следствия 5 и следствий 3 и 4) пока доказано только следующее

**Следствие 6.** *Элементарная Пропозициональная Динамическая Логика EPDL и её детерминированный вариант DEPDL разрешимы с экспоненциальной верхней оценкой сложности.*

## 4. ПРОПОЗИЦИОНАЛЬНОЕ $\mu$-ИСЧИСЛЕНИЕ

### 4.1 Синтаксис и семантика $\mu$C

Синтаксис и семантика пропозиционального $\mu$-Исчисления ($\mu$-Calculus $\mu$C) расширяют синтаксис (см. определение 3) и семантику (см. определение 5) Элементарной Пропозициональной Динамической Логики EPDL следующим образом.

**Определение 24.** *Пусть Con, Var и Act — те же алфавиты булевских констант, пропозициональных и программных переменных, что и для EPDL. Синтаксис $\mu$C состоит из формул, которые определяются структурной индукцией:*

- *Все средства построения формул, разрешённые в EPDL, являются легальными и в $\mu$C: булевские константы, пропозициональные переменные, пропозициональные комбинации формул без коллизий свободных и связанных вхождений переменных*[26]*, модальности необходимости и возможности.*
- *Для любой пропозициональной переменной $x \in Var$ и формулы $\phi$ без негативных*[27] *вхождений $x$, конструкторы наименьшей неподвижной точки ($\mu x.\phi$) и наибольшей неподвижной точки ($\nu x.\phi$) являются формулами.*

Таким образом, синтаксис EPDL — фрагмент синтаксиса $\mu$C, в котором нельзя использовать конструкторы неподвижных точек. Для удобства чтения и записи формул мы будем использовать те же правила опускания скобок, что и для EPDL.

Заметим, что из определения следует, что все связанные пропозициональные переменные в любой $\mu$C-формуле различны[28]. Значит, из двух конструкций

- $(\mu\ p.\langle a\rangle p) \wedge (\nu\ p.\langle a\rangle \neg p)$,
- $(\mu\ p.\langle a\rangle p) \wedge (\nu\ q.\langle a\rangle \neg q)$

первая не является формулой $\mu$-Исчисления, а вторая — является.

Введём несколько удобных и полезных синтаксический операций. Для произвольного синтаксического выражения $r$ и двух произвольных синтаксических выражений одного типа[29] $s$ и $t$ обозначим через $r_{t/s}$ результат подстановки $t$ вместо всех вхождений $s$ в $r$; также обозначим через $r^0_{t/s}$ выражение $t$, а для любого $n \geq 0$ обозначим через $r^{n+1}_{t/s}$ выражение $r_{r^n_{t/s}/s}$; таким образом, $r^m_{t/s}$ — результат $m$-кратной подстановки $t$ вместо $s$ в $\phi$. Например, если $\phi$ — это формула $\psi \vee \langle a\rangle x$ (где $x \in Var$ и $a \in Act$) то[30]

- $\phi^0_{true/x} \equiv true$,
- $\phi^1_{true/x} \equiv (\psi \vee (\langle a\rangle true))$,

[26] Вхождение переменной в выражение называется *связанным*, если это вхождение находится в области действия какого-либо конструктора $\mu$ и/или $\nu$ в этом выражении; в противном случае вхождение переменной называется *свободным*; например, в выражении $\big((\mu\ p.\langle a\rangle p) \wedge (\langle a\rangle \neg p)\big)$ первое вхождение пропозициональной переменной $p$ является связанным, а второе — свободным. Колизия свободных и связанных переменных — это ситуация, в которой какая-либо переменная имеет свободное и связанное вхождения (в разные синтаксически непересекающиеся подформулы); например, в этом же выражении $\big((\mu\ p.\langle a\rangle p) \wedge (\langle a\rangle \neg p)\big)$ есть коллизия переменных.

[27] Вхождение подформулы в формулу называется *негативным*, если это вхождение находится в области действия нечётного числа отрицаний $\neg$ в этой формуле; в противном случае вхождение подформулы называется *позитивным*. Например, в формуле $\neg\langle a\rangle \neg p \wedge \langle a\rangle \neg p$ первое вхождение пропозициональной переменной $p$ является позитивным, а второе — негативным.

[28] Тем самым мы избегаем необходимости вводить понятие $\alpha$-конверсии, обычной для переименовывания разных связанных переменных.

[29] т.е., например, пары формул, пары программ и т.д.

[30] В этой статье мы используем знак '$\equiv$' для синтаксической идентичности выражений, а знак '$=$' — для семантического равенства.

- $\phi^2_{true/x} \equiv (\psi \vee (\langle a \rangle (\psi \vee (\langle a \rangle true))))$,

и т.д.

Введём несколько семантических операций для того, чтобы определить семантику $\mu$C. Для любой функции $f : X \times Y$, произвольных элементов $x \in X$ и $y \in Y$ обозначим через[31] $upd(f, x, y)$ следующую функцию из $X$ в $Y$:

$$\lambda z \in X. \text{ если } z = x \text{ то } y \text{ иначе } f(y).$$

Если $M = (D, R, E)$ — произвольная модель, $S \subseteq D$ — произвольное множество состояний, а $x \in Var$ — пропозициональная переменная, то обозначим через $M_{S/x}$ модель $(D, R, E_{S/x})$, где $E_{S/x} = upd(E, x, S)$ — означивание, которое может отличаться от $E$ только на пропозициональной переменной $x$, причём $E_{S/x}(x) = S$.

Семантика $\mu$C определяется (как и для всех рассмотренных программных логик) в моделях посредством распространения означивания (заданного моделью для пропозициональных переменных) структурной индукцией по построению формул.

**Определение 25.** *Для произвольной модели $M = (D, R, E)$ имеем:*

- *$M(true)$, $M(false)$, $M(x)$ (для любой пропозициональной переменной $x \in Var$), $M(\neg\phi)$, $M(\phi \wedge \psi)$, $M(\phi \vee \psi)$, $M(\langle a \rangle \phi)$ и $M([a]\phi)$ (для любой программной переменной $a \in Var$) — аналогично определению 5;*

- *$M(\mu x.\phi)$ — наименьшее (относительно частичного порядка $\subseteq$) множество состояний $S \subseteq D$ такое, что $M_{S/x}(\phi) = S$;*

- *$M(\mu x.\phi)$ — наибольшее (относительно частичного порядка $\subseteq$) множество состояний $S \subseteq D$ такое, что $M_{S/x}(\phi) = S$.*

Для $\mu$C отношение выполнимости $\models$, выполнимость, истинность в моделях и тождественная истинность определяются аналогично EPDL.

Вариант $\mu$-Исчисления, в котором используются только детерминированные модели, мы будем называть детерминированным $\mu$-Исчислением ($\mu$D-Исчислением, $\mu$D), а вариант, в котором используются только тотальные детерминированные модели — тотальным $\mu$-Исчисленим ($\mu$T-Исчислением, $\mu$T).

Определение семантики $\mu$C (и его вариантов) нуждается в некотором обосновании, так как использует (как бы подразумевает) существование в любой модели $M$ наименьшего и наибольшего в $(2^D, \subseteq)$ решения теоретико-множественного семантического уравнения $M_{S/x}(\phi) = S$. Корректность данного определения следует из *свойства монотонности* формул $\mu$C и теоремы Кнастера-Тарского о неподвижных точках [23, 12, 29]. Монотонность $\mu$C-формул — это следующее свойство:

**Утверждение 8.** *Для любой пропозициональной переменной $x \in Var$, произвольной $\mu$C-формулы $\phi$, для произвольной модели $M = (D, R, E)$ если $\phi$ не имеет негативных вхождений $x$, то функция функция $\lambda S \subseteq D.M_{S/x}(\phi) : 2^D \to 2^D$ монотонно не убывает (от отношению $\subseteq$), и, если $\phi$ не имеет позитивных вхождений $x$, то эта функция монотонно не возрастает.*

Из-за ограниченности места мы не будем здесь воспроизводить ни доказательство этого утверждения, ни обоснование корректности определения семантики $\mu$C, ни доказательство следующего важного для нас следствия из свойства монотонности и теоремы Кнастера-Тарского:

**Следствие 7.** *Для любой пропозициональной переменной $x \in Var$, произвольной $\mu$C-формулы $\phi$ (к которой легально применять связывание $\mu x$ или $\nu x$), для любого $n \geq 0$ и произвольной модели $M$ имеют место следующие включения: $M(\phi^n_{false/x}) \subseteq M(\mu x.\phi)$ and $M(\nu x.\phi) \subseteq M(\phi^n_{true/x})$.*

В статье [12], в которой было определено $\mu$-Исчисление, были определены синтаксис, семантика (в терминах моделей) и предложена корректная аксиоматизация[32]. Первая корректная и полная аксиоматизация $\mu$C была построена только спустя 10 лет [28], а полнота исходной аксиоматизации была доказана только через 17 лет [29] после публикации работы [12].

[31] *upd* — аббривиатура от *update*

[32] Таким образом, изначально $\mu$-Исчисление не было *исчислением* в общелогическом смысле слова (см. раздел 1.2), а было языком с семантически определённым понятием выполнимости и истинности.

Как уже было сказано ранее, разрешимость $\mu$-Исчисления с экспоненциальной верхней оценкой сложности была сначала доказана схемным методом [18] (с использованием обобщённой проблемы тотальности) и (несколько позже) — теоретико-автоматным методом [9] (сведением проблемы выполнимости к проблеме непустоты автоматов Бюхи [5, 7, 34]). В той же работе [9] была установлена полнота проблемы выполнимости для $\mu$C в классе $EXP - Time$.

### 4.2 Примеры использования и выразительная сила $\mu$C

Мы уже видели в разделе 2.3, что сякая программа и формула SPDL эквивалентны некоторой программе и (соответственно) формуле PDL. Так как все программы $\mu$C — это программные переменные, то, разумеется, не все программы PDL могут быть эквивалентны программам $\mu$C. Но зато всякая формула PDL эквивалентна некоторой формуле $\mu$C.

Действительно, во-первых можно заметить, что для любых PDL-программ $\alpha$ и $\beta$, для любых PDL-формул $\phi$ и $\psi$ следующие пары PDL-формул эквивалентны:

- $[\phi?]\psi$ и $\phi \to \psi$ (т.е. $\neg\phi \vee \psi$);
- $\langle\phi?\rangle\psi$ и $\phi \wedge \psi$;
- $[\alpha;\beta]\phi$ и $[\alpha]([\beta]\phi)$;
- $\langle\alpha;\beta\rangle\phi$ и $\langle\alpha\rangle(\langle\beta\rangle\phi)$;
- $[\alpha \cup \beta]\phi$ и $([\alpha]\phi) \wedge ([\beta]\phi)$;
- $\langle\alpha \cup \beta\rangle\phi$ и $(\langle\alpha\rangle\phi) \vee (\langle\beta\rangle\phi)$.

Во-вторых, для любых программной и пропозициональной переменных $a \in Act$ и $p \in Var$ следующие формулы эквивалентны:

- $[a^*]p$ и $\nu\ x.(p \wedge [a]x)$;
- $\langle a^*\rangle p$ и $\mu\ x.(p \vee \langle a\rangle x)$.

Следовательно, в рамках $\mu$C можно ввести макросы $[?]$ и $\langle?\rangle$, $[;]$ и $\langle;\rangle$, $[\cup]$ и $\langle\cup\rangle$, $[*]$ и $\langle*\rangle$, причём, только два последних макроса используют конструкторы неподвижных точек. Более того, в рамках $\mu$C можно разрешить использовать любые программные конеструкции PDL и SPDL, элиминируя их по правилам, зависящим от того, внутри каких модальностей $\langle\ \rangle$ или $[\ ]$ эти конструкции используются.

Однако, выразительная сила $\mu$C сильнее чем выразительная сила PDL, т.е. существуют формулы $\mu$C, которые не имеют эквивалентных в PDL. Например, расмотрим две новых конструктора формул [22]: для любой программы $\alpha$ любого варианта L логики PDL $loop(\alpha)$ и $\Delta\alpha$ — формулы логики $loop$-L и $\Delta$L соотвтественно (например, $loop$-DSPDL и $\Delta$-PDL). Семантика новых формул определяется следующим образом: для любой модели $M = (D, R, E)$ и любого состояния $s \in D$ имеем

- $s \models_M loop(\alpha)$ тогда и только тогда, когда существует бесконечная вычислительная трасса[33] программы $\alpha$, начинающаяся с состояния $s$;
- $s \models_M \Delta\alpha$ тогда и только тогда, когда существует бесконечная последовательность состояний $s_0, \ldots, s_k,\ s_{(k+1)}, \ldots$ такая, что $s_0 = s$ и $s_k\langle\alpha\rangle_M s_{(k+1)}$ для любого $k \geq 0$.

Можно заметить, что для любых формулы $\phi$ и программ $\alpha$ и $\beta$ любого варианета PDL следующие пары формул эквивалентны:

- $loop(\phi?)$ и $false$;
- $loop(\alpha;\beta)$ и $loop(\alpha) \vee \langle\alpha\rangle loop(\beta)$;
- $loop(\alpha \cup \beta)$ и $loop(\alpha) \vee loop(\beta)$;
- $loop(\alpha^*)$ и $\Delta\alpha \vee \langle\alpha^*\rangle loop(\alpha)$;
- $\Delta\alpha$ и $\nu\ x.\langle\alpha\rangle x$, где $x \in Var$ — новая пропозициональная переменная.

Следовательно, для любого варианта L логики PDL, логика $loop$-L выразима в $\Delta$L, которая выразима в $\mu$-Исчислении или (если L — детеримнированный вариант или тотально детерминированный вариант PDL) в детерминированном $\mu$-Исчислении и, соотвтественно, в тотальном $\mu$-Исчислении. Однако известно [22], что $\mu$C имеет

[33] Так же как в доказательстве утверждения 2 (см. сноску 12) мы не определяем это понятие формально, но интуитивно оно совершенно ясно: это любая бесконечная последовательность состояний, которую "прошла" программа при её некотором исполнении от начала до конца.

большую выразительную силу, чем ΔPDL, которая сильнее *loop*-PDL, а *loop*-PDL имеет большую выразительную силу, чем PDL.

Ещё один пример формулы $\mu$C, которая не выразима в PDL [31], — это следующая формула

$$WIN_A \equiv$$
$$\equiv \mu\, x.\Big(\neg win_B \wedge win_A \vee$$
$$\vee\big(\neg win_B \wedge \langle move_A\rangle(win_A \vee$$
$$\vee \neg win_B \wedge [move_B]x)\big)\Big).$$

Эта формула имеет отношение к решению позиционных игр двух игроков с нулевой суммой. Будем говорить, что модель $M = (D, R, E)$ представляет позиционную игру двух игроков $A$ и $B$, если

- $D$ — это множество позиций,
- $R(move_A)$ и $R(move_B)$ — множество ходов игроков $A$ и $B$ соответственно,
- $R(win_A)$ и $R(win_B)$ — непересекающиеся множества выигрышных позиций игроков $A$ и $B$ соответственно.

Если игру начинает игрок $A$, а игроки ходят по очереди ($A$ - $B$ - $A$ - ...) до первого попадания в множество выигрышных позиций $A$ (тогда выигрывает $A$) или до первого попадания в множество выигрышных позиций $B$ (тогда выигрывает $B$), то формула $WIN_A$ выделяет в модели $M$, представляющей позиционную игру двух игроков $A$ и $B$, множество $M(WIN_A)$ всех тех состояний (позиций), в которых у $A$ есть выигрышная стратегия против $B$.

Приведём теперь пример формулы $\mu$C, использующей вложенные конструкторы неподвижных точек. (Этот пример играет особую роль в работах [19, 20].)

**Определение 26.** *Пусть $Fun \subseteq Act$ — произвольное конечное множествор программных переменных, $\phi$ — произвольная $\mu C$-формула, $M = (D, R, E)$ — произвольная модель, а $s \in D$ — произвольное состояние. Бесконечная последовательность состояний $s_0, s_1, \cdots \subseteq D$ порождена $Fun$ из $s$ в $M$, если эта последовательность — бесконечная вычислительная трасса*[34] *PDL-программы $(\bigcup_{a\in Fun} a)^*$, начинающаяся с состояния $s$ (т.е. $s_0 = s$ и для любого $k \geq 0$ найдётся программная переменная $a \in Fun$ такая, что $(s_k, s_{k+1}) \in R(a)$). Будем говорить, что*

- *формула $\phi$ неизбежна для $Fun$ в $M$ из $s$, если любая бесконечная последовательность состояний $s_0, s_1, \cdots \subseteq D$ порождённая $Fun$ из $s$ в $M$ содержит состояние $s_j \models_M \phi$;*
- *множество $Fun$ справедливо для (или по отношению к) $\phi$ в $M$ из $s$, если каждая последовательность состояний $s_0, s_1, \cdots \subseteq D$ порождённая $Fun$ из $s$ в $M$ имеет бесконечно много элементов*[35] *$t$ таких, что $t \models_M \phi$ (т.е. $\phi$ бесконечно часто выполнена в этой последовательности).*

**Определение 27.** *Пусть $Fun \subseteq Act$ — произвольное конечное множество программных переменных, $\phi$ — произвольная $\mu C$-формула. Определим два макроса:*

- *пусть*[36] *$AF(Fun, \phi)$ — это макрос для $\mu y.(\phi \vee \bigwedge_{a\in Fun}[a]y)$,*
- *а $fair(Fun, \phi)$ — макрос для $\nu x.AF(Fun, \phi \wedge x)$.*

**Утверждение 9.** *Для любой модели $M = (D, R, E)$ и любого состояния $s \in D$, произвольного конечного множества программных переменных $Fun$ и произвольной $\mu C$-формулы $\phi$ имеют место следующие эквивалентности:*

- *$s \models_M AF(Fun, \phi) \Leftrightarrow \phi$ неизбежна для $Fun$ в $M$ из $s$;*
- *$s \models_M fair(Fun, \phi) \Leftrightarrow Fun$ справедлива по отношению к $\phi$ в $M$ из $s$.*

*Доказательство.* Первая из этих эквивалентностей достаточно тривиальна и, по крайней мере, хорошо известна из контеста Computational Tree Logic [7, 34]. Для доказательства второй эквивалентности возьмём произвольную бесконечную последовательность состояний $s_0, s_1, \cdots \subseteq D$ порождённую $Fun$ из $s$ в $M$ такую, что $s_0 \models_M fair(Fun, \phi)$; в соотвествии с первой эквивалентностью, в этой последовательности есть состояние $t_0 \models_M \phi \wedge AF(Fun, fair(Fun, \phi))$, т.е. $t_0 \models_M$

[34] См. сноску 33.

[35] Возможно, совпадающих состояний, но с разными индексами в $s_0, s_1, \ldots$

[36] *AF* means *Always in Future* — модальность из темпоральной логики ветвящегося времени Computation Tree Logic CTL [7, 34].

$\phi$ и $t_0 \models_M AF(Fun, fair(Fun, \phi))$; следовательно, в этой последовательности есть состояние $t_1$ (где-то после $t_0$), в котором имеет место $t_1 \models_M \phi \wedge AF(Fun, fair(Fun, \phi))$; для этого состояния мы можем повторить те же рассуждения, что и для $t_0$ и найти состояние $t_2$, в котором имеет место $t_1 \models_M \phi \wedge AF(Fun, fair(Fun, \phi))$ , и так далее. □

В заключении данного раздела сошлёмся на более абстрактную характеристику выразительной силы $\mu$-Исчисления: в работе [25] показано, что $\mu$C на древовидных моделях имеет равную выразительную силу с теорией второго порядка монадических функций следования S(n)S [15].

## 5. РАЗРЕШИМОСТЬ $\mu$-ИСЧИСЛЕНИЯ

### 5.1 Вспомогательные утверждения

Напомним определение 11: говорят, что формула находится в нормальной форме, если все случаи оптрицания в этой формуле находятся на уровне литералов. Как было показано, в утверждении 1, любая формула любой из рассмотренных ранее в этой статье логик может быть преобразована за квадратичное время на линейной памяти к эквивалентной нормальной формуле этой же логики. Оказывается, что каждая формула $\mu$C(и, следовательно, её вариантов $\mu$D и $\mu$T) тоже может быть преобразована за квадратичное время на линейной памяти в эквивалентную нормальную формулу соответствующей логики; этот факт следует из утверждения 1 и

**Утверждение 10.** *Для любой пропозициональной переменной $x \in Var$ и произвольной $\mu C$-формулы $\phi$ (к которой легально применять связывание $\mu x$ или $\nu x$) следующие пары форму эквивалентны:*

- $\neg(\mu x.\phi)$ *и* $\nu x.(\neg\phi_{(\neg x)/x})$,
- $\neg(\nu x.\phi)$ *и* $\mu x.(\neg\phi_{(\neg x)/x})$.

Для $\mu$-исчисления имеет место аналог утверждения 2 (впервые доказан в [18]):

**Утверждение 11.** *Пусть $\phi$ — произвольная фиксированная $\mu C$-формула. Для каждой программной переменной $a \in Act$, которая встречается в $\phi$, пусть $f_a$ и $g_a$ — новые программные переменные, а $p_a$ — новая пропозициональная переменная (индивидуальные для каждого $a$). Пусть формула $totdet(\phi)$ получается в результате замены в $\phi$ всех вхождений каждой программной переменной $a \in Act$ на программу*[37] *$(f_a^*; p_a?; g_a)$; тогда имеем: формула $\phi$ равноистинна с формулой $totdet(\phi)$ тотального детерминированного варианта $\mu C$ (т.е. $\mu T$).*

*Доказательство.* Всё происходит аналогично доказательству утверждения 2 за исключением первого этапа доказательства, когда по произвольной модели $M$ и состоянию $s$ идёт построение конечной модели $M'$, содержащей это же состояние $s$, такой, что $s \models_M \phi \Leftrightarrow s \models_{M'} \phi$; теперь мы не можем построить конечную модель, но можем построить счётную (что достаточно для доказательства утверждения) после некоторых вспомогатнльных манипуляций с формулой.

Во-первых, в силу утверждения 10 не теряя общности можно считать, что формула $\phi$ находится в нормальной форме. Во-вторых, можно считать, что в формуле $\phi$ нет конструкцоров наименьшей неподвижной точки $\mu$ кроме как в одной специальной форме $\langle U \rangle$, где $U$ — это программа $\left(\bigcup_{a \text{ встречается в } \phi} a\right)^*$. Действительно, в силу свойства монотонности (утверждение 8) нормальная формула $\phi$ равноистинна с формулой $\phi_{(x \vee \langle U \rangle(\psi \wedge \neg x))/(\mu x.\psi)}$: формула $x \vee \langle U \rangle(\psi \wedge \neg x)$ эквивалентна формуле $[U](\psi \to x) \to x$, а в модели посылка этой формулы означает, что во всех достижимых состояниях означивание переменной $x$ содержит неподвижную точку (и, следовательно, наименьшую неподвижную точку).

И так, пусть даны произвольные нормальная $\mu$C-формула без конструкций наименьшей неподвижной точки кроме как в одной специальной форме $\langle U \rangle$, модель $M = (D, R, E)$ и состояние $s \in D$. Пусть $OBL$ — специальная переменная

[37] Напомним, что в рамках $\mu$C можно использовать любые программные конструкции PDL и SPDL, понимая и элиминируя их по правилам, зависящим от того, внутри каких модальностей $\langle\ \rangle$ или $[\ ]$ эти конструкции используются.

типа *очередь*[38] для хранения пар вида (состояние, формула), инициализируем её пустой очередью, а затем поставим в очередь пару $(s, \phi)$. Теперь, пока $OBL$ непустая очередь выполняем следующий цикл:

- пусть $(t, \psi)$ — первая пара ("голова") в очереди $OBL$; удалим эту пару из $OBL$;
- если $\psi$ не является формулой BOOL без переменных, связанных в $\phi$, то поступаем в соотвествии с разбором случаев:
  - если $\psi$ — это пропозициональная переменная $x$, связанная в $\phi$ в подформуле $\nu x.\xi$, то ставим в очередь $OBL$ пару $(t, \xi)$;
  - если $\psi$ — это формула вида $\xi \wedge \chi$ или $\xi \vee \chi$, то ставим в очередь $OBL$ две пары $(t, \xi)$ и $(t, \chi)$;
  - если $\psi$ — это формула вида $\langle a \rangle \xi$ или $[a]\xi$ и $t \models_M \psi$, то пусть $v$ — произвольное состояние $M$ такое, что $v \models_M \xi$; поставим в очередь $OBL$ пару $(v, \xi)$;
  - если $\psi$ — это формула вида $\langle a \rangle \xi$ или $[a]\xi$ и $t \not\models_M \psi$, то пусть $v$ — произвольне состояние $M$ такое, что $v \not\models_M \xi$; поставим в очередь $OBL$ пару $(v, \xi)$;
  - если $\psi$ — это формула вида $\langle U \rangle \xi$ и $t \models_M \psi$, то пусть $v$ — произвольное состояние $M$ такое, что $v \models_M \xi$; поставим в очередь $OBL$ пару $(v, \xi)$;
  - если $\psi$ — это формула вида $\langle U \rangle \xi$ и $t \not\models_M \psi$, то пусть $v$ — произвольне состояние $M$ такое, что $v \not\models_M \xi$; поставим в очередь $OBL$ пару $(v, \xi)$.

Представленный алгоритм определяет процесс, который может работать бесконечно (так как каждый раз когда во время цикла доходим до связанных переменных мы добавляем новые пары, но с более сложными формулами). Однако множество состояний, которые появлялись во время работы этого алгоритма, счётно. Обозначим это множество $D'$. Обозначим $R'$ следующую интерпретацию: $R'(a) = \{(t, u) \in (D')^2 \ : \ (t, u) \in R(a)\}$ для любой программной переменной $a \in Act$. Обозначим $E'$ следующее означивание: $E'(p) = \{t \in D' \ : \ t \in E(p)\}$ для любой пропозициональной переменной $p \in Var$. И, наконец, обозначим $M'$ модель $(D', R', E')$. Очевидно (или может быть доказано индукцией по структуре формул), что для любой формулы $\psi$, для любого состояния $t \in D'$, если пара $(t, \psi)$ возникала во время выполнения алгоритма, то $t \in M(\psi) \ \Leftrightarrow \ t \in M'(\psi)$. В частности, для начального состояния и формулы имеем $s \in M(\phi) \ \Leftrightarrow \ s \in M'(\phi)$. Следовательно, все переменные, связанные в формулы посредством $\nu$, в моделе $M'$ означены неподвижными точками соответствующих фодформул; в силу свойства монотонности (утверждение 8) эти означивания являются соотвествующими *наибольшими* неподвижными точками. Таким образом, искомая счётная модель построена. □

Из утверждений 10 и 11 и анализа доказательства утверждения 2, следует аналог утверждения 3:

**Утверждение 12.** *Произвольная $\mu C$-формула $\phi$ тождественно истинна тогда и только тогда, когда $totdet(\phi)$ выполнима на пустом слове $\theta$ в любой эрбрановой модели.*

Следующее утверждение является ствием теоремы Кнастера-Тарского о неподвижных точках.

**Утверждение 13.** *Для любой пропозициональной переменной $x \in Var$ и произвольной $\mu T$-формулы $\phi$ (к которой легально применять связывание $\mu x$ или $\nu x$), для любой тотальной детерминированной модели $M$ имеют место следующие равенства:*

- $M(\mu x.\phi) = \bigcup_{n \geq 0} M(\phi^n_{false/x})$,
- $M(\nu x.\phi) = \bigcap_{n \geq 0} M(\phi^n_{true/x})$.

*Доказательство.* Доказательство обоих равенств проходит аналогично одновременной индукцией по числу конструкций неподвижных точек в формуле; более того, доказательство шага индукции проходит аналогично доказательству базы индукции; поэтому мы здесь приведём только доказательство базы иннддукции для конструктора наименьшей неподвижной точки, т.е. докажем равенство

[38] В доказательстве утверждения 2 эта переменная имела тип множество.

$M(\mu x.\phi) = \bigcup_{n\geq 0} M(\phi^n_{false/x})$ для формулы $\phi$ не содержащей конструкций неподвижных точек.

Во-первых, включение $\bigcup_{n\geq 0} M(\phi^n_{false/x}) \subseteq M(\mu x.\phi)$ верно в любой модели (см. следствие 7).

Во-вторых, предположим противное, что $\bigcup_{n\geq 0} M(\phi^n_{false/x}) \neq M(\mu x.\phi)$. Обозначим $\bigcup_{n\geq 0} M(\phi^n_{false/x})$ через $S$. По нашему предположению $M_{S/x}(\phi) \neq S$. Следовательно, существует состояние $s \in D_M$ такое, что $s \in (M_{S/x}(\phi) \setminus S)$. Так как $\phi$ не содержит конструкций неподвижных точек, а $M$ — тотальная детерминированная модель, то существует конечное множество $S_{fin} \subseteq S$ such that $s \in M_{S_{fin}/x}(\phi)$. Но тогда $s \in \bigcup_{n\geq 0} M(\phi^n_{false/x})$ из-за конечности $S_{fin}$. — Пришли к противоречию со сделанным предположением. Поэтому $M_{S/x}(\phi) = S$. □

## 5.2 Трансляция $\mu$-Исчисления в схемы

**Определение 28.** *Оператор в недетеримнированной схеме Янова называется незащищённым, если в этой схеме существует путь от начала (метки 0) до этого оператора, проходящий только через операторы выбора; в противном случае оператор называется защищённым.*

Определим рекурсивный алгоритм $\mu T2S$ трансляции нормальных формул тотального детерминированного $\mu$-Исчисления в беспетельные недетерминированные схемы Янова:

- Для констант *true*, *false*, литералов, конъюнкции и дизъюнкции, модальнсти $\langle\,\rangle$ (и эквивалентной[39] модальности $[\,]$) — аналогично алгоритму $TE2S$;
- $\mu T2S(\mu x.\phi)$ получается из схемы $\mu T2S(\phi)$ в результате замены каждого оператора выбора $if\ x\ then\ stop\ else\ excp$ с условием $x$ на оператор безусловного перехода $goto\ \{0\}$.
- $\mu T2S(\nu x.\phi)$ получается из схемы $\mu T2S(\phi)$ в результате замены
  - каждого *защищённого* оператора выбора $if\ x\ then\ stop\ else\ excp$ с условием $x$ на оператор выбора $if\ x\ then\ \{0\}\ else\ stop$,
  - и каждого *незащищённого* оператора выбора $if\ x\ then\ stop\ else\ excp$ с условием $x$ на *stop*.

**Утверждение 14.** *Алгоритм $\mu T2S$ за экспоненциальное время на линейной памяти строит по нормальным формулам $\mu T$ беспетельную недетерминированную схему Янова.*

Это утверждение непосредственно следует из описания алгоритма, отметим только, что экспоненциальное время возникает из-за необходимости распознать в схеме защищённые и незащищённые операторы выбора.

**Утверждение 15.** *Пусть $M$ — произвольная эрбранова модель, $\omega \in Act^*$ — произвольное состояние этой модели, $\phi$ — произвольная нормальная формула тотального детерминированного варианта $\mu$-Исчисления (т.е. $\mu T$-формула), $Gfp$ (Greatest fix points) — множество всех пропозициональных переменных этой формулы, связанных в ней конструктором наибольшей неподвижной точки, а $Cnj$ — множество всех новых пропозициональных переменных $q$, которые были использованы при выполнении алгоритма $\mu T2S(\phi)$ при трансляции конъюнкций. Имеем: $\omega \models_M \phi$ тогда и только тогда, когда недетерминированная схема Янова $EPDL2S(\phi)$ имеет конечную полную трассу, начинающуюся с состояния $\omega$ в любой модели $M'$ подходящей для $Gfp$, которая является модификацией $M$ на $(Gfp \cup Cnj)$.*

*Доказательство.* Индукция по структуре формул. База индукции и шаг индукции для конъюнкции, дизъюнкции и модальности $\langle\,\rangle$ — аналогично доказательству утверждения 5. Остаётся обосновать шаг индукции для конструкторов неподвижных точек.

Начнём со случая $\mu p.\phi$. В соотвествии со следствием 13, $M(\mu x.\phi) = \bigcup_{n\geq 0} M(\phi^n_{false/x})$. Так как $\mu T2S(\mu x.\phi)$ получается из $\mu T2S(\phi)$ в результате подстановки $goto\ \{0\}$ вместо $if\ x\ then\ stop\ else\ excp$, то $\mu T2S(\mu x.\phi)$ эквивалентна $\mu T2S(\bigvee_{n\geq 0} \phi^n_{false/x})$.

Рассмотрим случай $\nu p.\phi$. Имеем (в соотвествии со следствием 13): $M(\nu x.\phi) =$

[39] Как это уже было отмечено в разделе 2.3, в тотальных детерминированных моделях формулы $[a]\phi$ и $\langle a\rangle\phi$ эквивалентны.

$\bigcap_{n\geq 0} M(\phi^n_{true/x})$. А так как $\mu T2S(\nu x.\phi)$ получается из $\mu T2S(\phi)$ в результате подстановки $if\ x\ then\ \{0\}\ else\ stop$ вместо защищённых воператоров $if\ x\ then\ stop\ else\ excp$ и $stop$ вместо незащищённых, то $\mu T2S(\nu x.\phi)$ эквивалентна $\mu T2S(\bigwedge_{n\geq 0}\phi^{n+1}_{true/x})$ так как $x \in Gfp$ и, следовательно, его означивание может быть истинно только на произвольных но обязательно конечных множестве термов. □

Основная теорема следует из утверждений 10-15 и следствия 5.

**Теорема 1.** *Пропозициональное $\mu$-Исчисление разрешимо с экспоненциальной верхней оценкой сложности (от линейного размера формулы).*

### 5.3 Примеры трансляции $\mu$T

Проиллюстрируем метод трансляции $\mu$T в схемы на следующих трёх формулах:

- $\nu x.(\langle f\rangle x)$;
- $\nu x.\Big(\mu y.\big((p \wedge \langle g_a\rangle x) \vee \langle f_a\rangle y\big)\Big)$;
- $\mu x.(p \vee [f]x) \vee \nu y.(\neg p \wedge \langle f\rangle y)$.

В классе тотальных детерминированных моделей первая формула тождественно истинна, так как она эквивалентна $\Delta f$, а любая программная переменная в любой тотальной детерминированной модели всегда бесконечно самоприменима. Результат её трансляции алгоритмом $\mu T2S$ представлен на Рис. 8. Так как для этой формулы множество $Gfp$ состоит из единственного символа $x$, то тотальность представленной схемы относительно этого множества очевидна.

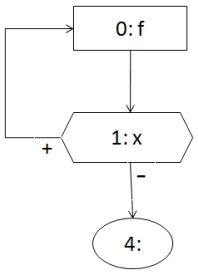


Рис. 8.

Вторая из перечисленных формул эквивалентна $totdet(\Delta a)$. Поэтому она выполнима, но не истинна, так как конструкция, использованная в *totdet*, позволяет моделировать не всюду определённые бинарные отношения (см. доказательство утверждения 2). Результат её трансляции алгоритмом $\mu T2S$ представлен на Рис. 9. И для этой формулы множество $Gfp$ состоит из единственного символа $x$. Но если каждую пропозициональную переменную $q$ и $p_a$ означить пустым множеством термов, то очевидно, что в такой модели эта схема не имеет ниодной конечной полной трассы.

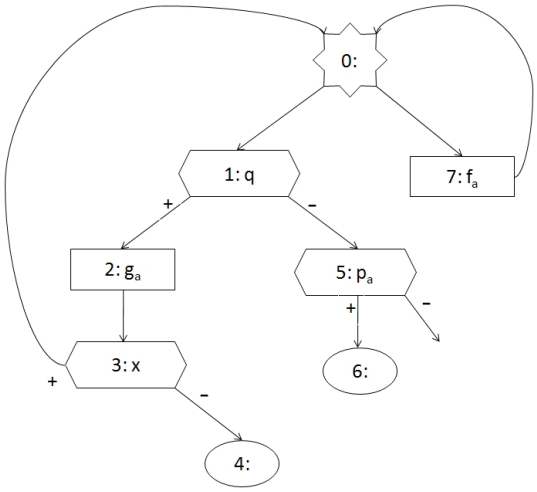


Рис. 9.

Третья из перечисленных формул эквивалентна формуле $(AFp \vee EG\neg p)$ темпоральной логики ветвящегося времени Computation Tree Logic CTL[40] [7, 34]. Она является тождественно истинной, так как формула утверждает, что или на любой бесконечной трассе $f^*$ обязательно встретиться состояние, в котором верно $p$, или найдётся бесконечная трасса $f^*$, в которой во всех состояниях всегда $\neg p$. Результат трансляции этой формулы алгоритмом $\mu T2S$ представлен на Рис. 10. Для этой формулы множество $Gfp$ тоже состоит из единственного символа $x$. Достаточно очевидно, что схема на этом рисунке тотальна относительно $\{x\}$.

### 5.4 Заключительные замечания

И так, в данной статье представлен главный результат развития схемного метода доказательства разрешимости пропозициональных

[40] С модальностью $AF$ — *Always in Future* — мы уже встречались (см. сноску 36), а вторая модальность $EG$ — этот *Exists Globally*.

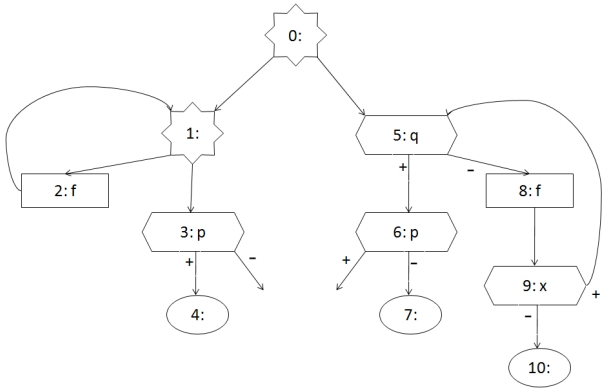


Рис. 10.

программных логик — разрешимость вариантов пропозициональной динамической логики и $\mu$-Исчисления впутём сведения проблемы к проблеме относительной тотальности недетерминированных схемс Янова. Первоначально этот метод был разработан в 1983-1987 гг. только для вариантов пропозициональной динамической логики [13, 38], однако ряд шагов в цитируемых публикациях не был достаточно обоснован[41]. Затем метод был обобщён для $\mu$-Исчисления в [18], при этом, однако, вместо *тносительной* тотальности была использована *обобщённая* тотальность. Как уже было сказано в данной работе, в проблема обобщённой тотальности используются кванторы второго порядка по пропозициональным переменным, которые возникают при элиминации неподвижных точек. Как это происходит, можно понять из следующего утверждения, доказанного в [18].

**Утверждение 16.** *Пусть $\phi$ — произвольная $\mu C$-формула, $x \in Var$ — произвольная пропозициональная переменная (к которой легально применять связывание $\mu x$ или $\nu x$ в $\phi$), $U$ — это программа*[42] $\left(\bigcup_{a \text{ встречается в } \phi} a\right)^*$, *а $M$ и $\omega \in Act^*$ — роизвольная эрбранова интерпретация и терм. Тогда имеют место следующие эквивалентности.*

- $\omega \models_M \mu x.\phi$ *тогда и только тогда, когда для* <u>*любой*</u> *модели $M'$, которая является модификацией $M$ на $x$, верно $\omega \models_{M'} \big([U](\phi \to x) \to x\big)$;*
- $\omega \models_M \nu x.\phi$ *тогда и только тогда, когда для* <u>*некоторой*</u> *модели $M'$, которая является модификацией $M$ на $x$, верно $\omega \models_{M'} \big([U](x \to \phi) \wedge x\big)$;*

Следует, однако, заметить, что разрешающая процедура для проблемы обобщённой тотальности значительно сложнее для понимания и обоснования, чем для относительной тотальности.

Ещё один новый интересный вариант схемного метода для $\mu$-Исчисления был представлен в качестве доклада на 10-ой Ershov Informatics Conference PSI-2015 (25 - 27 August 2015, Innopolis, Kazan, Russia) [20]. В этом варианте транслируются *защищённые* нормальные формулы $\mu$C (точнее, $\mu$T), а для проверки относительной тотальности используется верификация специальной фиксированной $\mu$-формулы в конечных моделях, заданных блок-схемами беспетельных недетерминированных схем Янова. Эта специальная формула — это формула для свойства справедливости[43] $fair(\ldots,\ldots)$ из определения 9, она имеет линейную сложность верификации в конечной модели[44]. Так как в [20] для доказательства разрешимости $\mu$-Исчисления всякую формулу сначала надо преобразовать в эквивалентную защищённую формулу, а проблема разрешимости для $\mu$-Исчисления полна в классе $EXP - Time$ [9], то, в частности, в [20] утверждается, что алгоритмическая проблема преобразования $\mu$C-формул в эквивалентные защищённые формулы имеет экспоненциальную нижнюю оценку. (Ранее в работе [4] было отмечено, что экспоненциальную нижнюю оценку имеет алгоритм элиминации незащищённых связанных переменных, описанный в [29].)

В заключении отметим, что не смотря на значение $\mu$-Исчисления для обоснования чрезвычайно восстребованной на практике верификации конечных моделей [7, 34], нам не извесно публикаций о применении разрешающих процедур для $\mu$-Исчисления в практической верификации.

[41] Например, переход к эрбрановым моделям для логик.

[42] Эту программу мы уже использовали в доказательстве утверждения 11. Фактически, в этом доказательстве используется первая из следующих эквивалентностей.

[43] Эта же $\mu$C-формула и табличный метод верификации $\mu$-Исчисления в конечных моделях из [8] использованы в [19] для аксиоматизации пропозициональной логики линейного времени LTL (propositional Linea Time temporal Logic) [21].

[44] линейную от размера модели

## СПИСОК ЛИТЕРАТУРЫ


1. *Ball T., Podelski A., Rajamani S.K.* Boolean and Cartesian Abstraction for Model Checking C Programs // Microsoft Research, 2000, MSR-TR-2000-115. (Доступны на `http://research.microsoft.com/apps/pubs/default.aspx?id=69821`.)

2. *Ben-Ari M., Halpern J.Y., Pnueli A.* Deterministic propositional dynamic logic: Finite models, complexity, and completeness // Journal of Computer and System Sciences, 1982, v.25, n.3. P.402–417.

3. *Bradfield J., Stirling C.* Modal mu-calculi // In: The Handbook of Modal Logic, P. Blackburn, J. van Benthem and F. Wolter (eds.). Elsevier, 2006. P.721–756.

4. Bruse F., Friedmann O., Lange M. Guarded Transformation for the Modal mu-Calculus. arXiv:1305.0648v2, 2013 (available at `http://arxiv.org/abs/1305.0648`).

5. *Büchi* Weak Second Order Arithmetic and Finite Automata. The Collected Works of J. Richard Buchi, New York: Springer, 1990. P.398–424. (Есть русский перевод: *Бюхи Д.Р.* Слабая арифметика второго порядка и конечные автоматы // Кибернетический сборник, вып.8, М.: Мир, 1964. С.42–77.)

6. *Chagrov A.*[45], *Zakharyaschev M.* Modal Logic. Oxford University Press, 1997.

7. *Clarke E.M., Grumberg O., Peled D.* Moedel Checking, MIT Press, 1999. (Есть русский перевод: *Кларк Э.М., Грамберг О., Пелед Д.* Верификация моделей программ. Model Checking. М.: МЦНМО, 2002.)

8. *Cleaveland R.* Tableau-based model checking in the propositional mu-calculus // Acta Informatica, 1990, vol.27. P.725–747.

9. *Emerson E.A., Jutla C.J.* The Complexity of Tree Automata and Logics of Programs // SIAM J. Comput., 1999, vol.29. P.132-158.

10. *Fischer M., R. Ladner* Propositional dynamic logic of regular programs // Journal of Computer and System Sciences, 1979, vol.18. P.194–211.

11. *Harel D., Kozen D., Tiuryn J.* Dynamic Logic. MIT Press, 2000.

12. *Kozen D.* Results on the Propositional Mu-Calculus // Theoretical Computer Science, 1983, vol.27. P.333-354.

13. *Nepomniaschy V.A., Shilov N.V.* Non-deterministic program schemata and their relation to dynamic logic. Internat. Conf. on Math. Logic and its Applications. Plenum Press, 1987, 137-147.

14. *Podlovchenko R.I.* A.A. Lyapunov and A.P. Ershov in the Theory of Program Schemes and the Development of Its Logic Concepts // Lecture Notes in Computer Science, v.2244, 2001, p.8-23.

15. *Rabin M.O.* Decidability of second order theories and automata on infinite trees // Trans. ASM, 1969, v. 141. P.1–35. (Есть русский перевод: *Рабин М.О* Разрешимость теорий второго порядка и автоматы над бесконечными деревьями // Кибернетический сборник, вып. 8, М.: Мир, 1971. С.72–116.)

16. *Rutledge J.D.* On Ianovs program schemata // Journal of the ACM. 1964, v.11, No 1. P.1–9.

17. *Rutledge J.D.* Program schemata as automata. Part I. // Journal of Computer and System Sciences. 1973, vol. 7, No 6. P. 543–578.

18. *Shilov N.V.* Program Schemata vs. Automata for Decidability of Program Logics // Theor. Comput. Sci. 1997, vol.175. P.15-27.

19. *Shilov N.V.* An approach to design of automata-based axiomatization for propositional program and temporal logics (by example of linear temporal logic) // In: Logic, Computation, Hierarchies. Series: Ontos Mathematical Logic, v.4, Brattka V., Diener H., Spreen D. (eds.). Ontos-Verlag/De Gruyter. 2014. P.297-324.

20. *Shilov N.V.* Program Schemata Technique to Solve Propositional Program Logics Revised // Post-proceedings of 10th Ershov Informatics Conference PSI-2015 (25-27 August 2015, Innopolis, Kazan, Russia). To appear in Springer Lecture Notes in Computer Science, 2015, 15 p.

21. *Stirling C.* Modal and Temporal Logics // In: Handbook of Logic in Computer Science, v.2, Abramsky S., Gabbay D.M, Maibaum S.E. (eds.). Oxford University Press, 1992. P.477-563.

22. *Streett R.* Propositional dynamic logic of looping and converse is elementary decidable // Information and Control. 1982, vol. 54. P. 121–141.


[45] Александр Васильевич Чагров (1957-2016), заведующий кафедрой алгебры и математической логики Тверского государственного университета, скоропостижно скончался 6 февраля 2016 г., когда готовилась данная статья.

23. Tarski A. A lattice-theoretical fixpoint theorem and its applications. Pacific Journal of Mathematics, v.5, 1955, p.285-309.

24. The Description Logic Handbook: Theory, Implementation, Applications / F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider, editors. Cambridge University Press, 2003.

25. *Schlingloff B.-H.* On the Expressive Power of Modal Logics on Trees // Lecture Notes in Computer Science, 1992, v.620. P.441–451.

26. *Valiev M.K.* Decision complexity of variants of propositional dynamic logic // Lecture Notes in Computer Science 1980, v.88. P.656–664.

27. *Vardi M.Y., Wolper P.* Automata-Theoretic techniques for modal logics of programs // Journal of Computer and System Sciences. 1986, vol. 32, No 2. P. 183–221.

28. Walukiewicz I. A Complete Deductive System for the mu-Calculus. Proceedings of IEEE LICS'93, 1993, p.136-147.

29. Walukiewicz I. Completeness of Kozen's axiomatisation of the propositional Mu-calculus. Information and Computation, v.157, 2000, p.142-182.

30. *Zakharov V.A.* An efficient and unified approach to the decidability of equivalence of propositional programs // Lecture Notes in Computer Science, 1998, v.1443. P.247–258.

31. *Бодин Е.В., Шилов Н.В., Ии К.* О программных логиках — просто // Системная информатика: сборник научных трудов, вып. 8. Новосибирск: Наука, 2003. С.206–249.

32. *Ершов А.П.* Об операторных схемах Янова // Проблемы кибернетики: сборник научных трудов, выпуск 20. М: Наука, 1968. С. 181–200.

33. *Ершов А.П.* Введение в теоретическое программирование (беседы о методе). Учебное пособие. М.: Наука, 1977.

34. *Карпов Ю.Г.* Model Checking. Верификация параллельных и распределенных программных систем. СПб.: БХВ-Петербург, 2009.

35. *Котов В.Е., Сабельфельд В.К.* Теория схем программ. М.: Наука, 1991.

36. *Кфури А.Д., Столбоушкин А.П., Ужичин П.* Некоторые открытые вопросы в теории схем программ и динамических логик // УМН. 1989, т. 44, No 1(265). С. 35-–55.

37. *Непомнящий В.А., Рякин О.М.* Прикладные методы верификации программ. М.: Радио и связь, 1988.

38. *Непомнящий В.А., Шилов Н. В.* Схемы недетерминированных программ и их отношение к динамической логике. // Кибернетика, 1988, No 3. С.12–19.

39. *Одинцов С.П., Сперанский С.О., Дробышевич С.А.* Введение в неклассические логики: учебое пособие. Новосиб. гос. ун-т. Новосибирск: РИЦ НГУ, 2014.

40. *Подловченко Р.И.* От схем Янова к теории моделей программ // Математические вопросы кибернетики, вып. 7, под ред. С.В. Яблонского. М.: Наука, Физматлит, 1998. С.281–302.

41. *Шилов Н.В.* Формализмы и средства создания и поддержания онтологий // Системная информатика: монография, вып. 11, под ред. А.Г. Марчука. Новосибирск: Издательство Сибирского Отделения Российской Академии Наук, 2009. С. 10–48.

42. *Шилов Н.В.* Основы синтаксиса, семантики, трансляции и верификации программ. Новосибирск: Новосибирский государственный университет, 2011.

43. *Шилов Н.В., Бернштейн А.Ю., Шилова С.О.* Применение недетерминированных монадических схем программ к исследованию свойств программных логик с неподвижными точками // Международная конференция МАЛЬЦЕВСКИЕ ЧТЕНИЯ (тезисы докладов), Новосибирск, 11-15 ноября 2013 г.. Новосибирск: ИМ СО РАН, 2013. С.58. (Доступны на `http://www.math.nsc.ru/conference/malmeet/13/maltsev13.pdf`.)

44. *Шилов Н.В., Шилова С.О., Бернштейн А.Ю.* Обобщенная тотальность недетерминированных схем Янова и разрешимость программной логики с неподвижными точками // Современные информационные технологии и ИТ-образование. Сборник избранных трудов IX Международной научно-практической конференции. МГУ, 14-16 ноября 2013 г., под ред. В.А. Сухомлина. М.: МГУ, 2014. С.444–455.

45. *Янов Ю.И.* О логических схемах алгоритмов // Проблемы кибернетики: сборник научных трудов, вып. 1. М: Наука, 1958. С. 75–127.

46. *Янов Ю.И.* О локальных преобразованиях схем алгоритмов // Проблемы кибернетики: сборник научных трудов, вып. 20. М: Наука, 1968. С. 201–216.