RU EN ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ ISSN 2079-3316
научная статья ТЕОРЕТИЧЕСКИЕ ОСНОВАНИЯ ПРОГРАММНЫХ СИСТЕМ

УДК 004.4'242
10.25209/2079-3316-2026-17-3-163-190

Проверка бисимуляционной эквивалентности формальных моделей игровых механик с охранными условиями и взвешенными переходами

Влада Владимировна **Кугуракова**
Казанский федеральный университет, Казань, Россия

Аннотация. Настоящая статья посвящена задаче формальной проверки эквивалентности двух реализаций одной игровой механики. На практике эта задача возникает при рефакторинге игровой логики, сравнении независимых реализаций одной спецификации и верификации балансировочных преобразований. Для формального представления таких механик используется модель GD-FST (Game Design Finite State Transducer). Стандартный подход теории верификации – бисимуляция – здесь неприменим напрямую, поскольку в GD-FST рёбра несут не просто метки, а пары (условие охраны, вес), что порождает новую нетривиальную задачу согласования переходов.

Вклад работы:

1) введено определение бисимуляционной эквивалентности для трансдьюсеров GD-FST (определение 6), учитывающее структуру рёбер формализма;
2) предложен алгоритм проверки бисимуляционной эквивалентности (алгоритм 1), построенный на адаптации классического разбиения Канеллакиса–Смолки;
3) доказаны корректность и полнота алгоритма (теорема 1).

Практическая значимость результата состоит в возможности автоматической проверки того, что рефакторинг или балансировочное преобразование игровой механики не изменяет её наблюдаемого поведения. Полученный аппарат применим не только к игровым механикам, но и к другим системам, управляемым правилами с охранными условиями и числовыми весами.

Ключевые слова и фразы: бисимуляционная эквивалентность, GD-FST, игровые механики, верификация, алгоритм Канеллакиса–Смолки, разбиение Хопкрофта, темпоральная логика

Благодарности: Работа выполнена при поддержке Академии наук Республики Татарстан, Соглашение от 22.12.2025 № 12/2025-ПД-КФУ

Для цитирования: Кугуракова В.В. *Проверка бисимуляционной эквивалентности формальных моделей игровых механик с охранными условиями и взвешенными переходами* // Программные системы: теория и приложения. 2026. **Т. 17**. № 3(72). С. 163–190. https://psta.psiras.ru/read/psta2026_3_163-190.pdf

1. Введение

Одной из фундаментальных задач инженерии программного обеспечения является обеспечение поведенческой эквивалентности систем при их модификации, оптимизации или рефакторинге. В области разработки видеоигр эта проблема приобретает особую остроту из-за высокой итеративности геймдизайна и растущей сложности внутренних взаимосвязей механик.

Игровая механика может быть реализована по-разному, сохраняя при этом идентичное наблюдаемое поведение. Классический пример: кулдаун, реализованный через непрерывный таймер, и кулдаун, реализованный через дискретный счётчик тиков (тактов игрового процесса), могут быть интуитивно неотличимы для игрока, однако формальная проверка такой эквивалентности требует явного согласования словарей переменных и переходов фаз `ACTIVE` → `COOLDOWN` → `READY`. Без такого согласования две реализации могут оказаться различимы на уровне охранных условий. Аналогичная ситуация возникает при рефакторинге игровой логики в редакторе Tula [1]: геймдизайнер переструктурировал механику, и требуется гарантия, что новая реализация поведенчески эквивалентна исходной.

В теории верификации программ задача поведенческой эквивалентности решается через *бисимуляцию* [2, 3]: два автомата бисимулярны, если каждый переход в одном может быть «зеркально» воспроизведён в другом, и наоборот. Бисимулярные системы удовлетворяют одним и тем же формулам темпоральных логик CTL (Computation Tree Logic – логика деревьев вычислений) [4] и LTL (Linear Temporal Logic – линейная темпоральная логика) [5, 6], что делает бисимуляцию мощным инструментом верификации.

Переход от игровой механики к формальной модели. В настоящей работе игровая механика рассматривается как конечная графовая модель поведения: вершины задают наблюдаемые фазы или конфигурации механики, а рёбра – допустимые переходы между ними, вызываемые действиями игрока, системными событиями или изменениями параметров среды. Такой взгляд позволяет сравнивать не конкретный код в игровом движке, а абстрактное поведение механики.

Для такого представления используется ориентированный конечный граф состояний и переходов, дополненный разметкой вершин, охранными условиями и числовыми весами переходов, – называемый *трансдьюсером* GD-FST[1]. Формальное определение GD-FST приведено в определении 2.

[1] GD-FST – *Game Design Finite State Transducer*. В прикладных контекстах встречается также трактовка, связанная с анализом разнообразия игрового процесса (*Gameplay Diversity Finite State Transducer*).

Нетривиальность проверки эквивалентности. После перехода к графовой модели задача не сводится к проверке изоморфизма графов. Стандартная бисимуляция определена для размеченных систем переходов с помеченными рёбрами [2], однако в GD-FST рёбра несут составные метки $(\mathrm{cond}, w) \in \mathrm{Cond} \times \mathbb{R}_{\geq 0}$, где cond – булево условие охраны, w – числовой вес перехода. Это порождает три нетривиальных вопроса, не возникающих в стандартной теории:

1) Когда два перехода (v_1, e_1, v_1') и (v_2, e_2, v_2') считаются «согласованными»?
2) Как сравнивать логически эквивалентные, но синтаксически различные условия охраны?
3) Как трактовать вероятностные веса при проверке стохастической эквивалентности?

Настоящая статья отвечает на эти вопросы, вводя *GD-бисимуляцию* – расширение классического понятия, специализированное для моделей игровых механик с охранными условиями и взвешенными переходами.

Существующие инструменты и их ограничения. Дорманс [7] предложил диаграммы Machinations — графический формализм с операционной семантикой для моделирования потоков ресурсов в игровых механиках; днако этот аппарат ориентирован на симуляцию и балансировку и не поддерживает ни формальную верификацию, ни проверку поведенческой эквивалентности реализаций. Обзор методов software model checking применительно к игровым движкам [8] показывает, что существующие подходы ориентированы на верификацию кода реализации, а не игровой логики, и не предоставляют формального аппарата для сравнения двух реализаций одной спецификации. Таким образом, задача GD-бисимуляции возникает в предметной области, где существующие инструменты оставляют формальный пробел.

Множественность семантик. Один и тот же GD-FST допускает несколько устоявшихся семантик в зависимости от вопроса верификации: как *раскрашенная сеть Петри* он поддаётся анализу параллелизма и достижимости, как *цепь Маркова* (DTMC/MDP) – вероятностной проверке в PCTL/CSL, как *сеть временны́х автоматов* – анализу с временными ограничениями в TCTL (Timed Computation Tree Logic), то есть временно́м расширении CTL для систем с часами. Этот список не является исчерпывающим: возможны и другие интерпретации – например, взвешенный автомат над полукольцом для количественного анализа путей или стохастическая игра для синтеза стратегий.

Каждая такая семантическая интерпретация заимствует доказательную базу зрелой теории. Возможность изучать одну и ту же формальную модель через различные поведенческие семантики отражает устоявшийся

методологический принцип формальных методов [9]. Настоящая работа использует интерпретацию GD-FST как *размеченной системы переходов*[2], а её алгоритмика восходит к минимизации конечных автоматов [10]. Именно это позволяет перенести классический аппарат без изменения его сути, адаптировав понятие согласованности переходов к составным меткам (cond, w).

Вклад и позиционирование. Работа относится к теоретическому фундаменту формальных методов анализа игровых механик и решает самостоятельную задачу в рамках класса K5 (поведенческая эквивалентность) классификации задач формального анализа игровых систем [11] – проверку бисимуляционной эквивалентности моделей GD-FST. Она примыкает к линии работ автора по формальным методам в геймдизайне [12, 13], оставаясь при этом законченным независимым исследованием: вводимые определения, алгоритм и доказательства самодостаточны и не опираются на результаты этих работ.

Переносимость. Введённые определения и алгоритм применимы к любым размеченным системам переходов с охранными условиями и числовыми весами: сетевым протоколам, рабочим процессам (workflow), мультиагентным системам. Игровые механики служат основным, но не единственным примером.

2. Предварительные сведения

2.1. Обозначения

В таблице 1 собраны основные обозначения, используемые в статье.

Таблица 1. Основные обозначения

Обозначение	Смысл
$\mathcal{T} = \langle V, E, \lambda_V, \lambda_E \rangle$	GD-FST трансдьюсер
$(\mathcal{T}, v_0)$	отмеченный трансдьюсер с начальной вершиной v_0
$v \xrightarrow{a} v'$	переход по метке a
$\mathrm{cond}_1 \equiv \mathrm{cond}_2$	логическая эквивалентность условий охраны
$e_1 \simeq e_2$	(cond, w)-совместимость рёбер (определение 5)
$v_1 \sim v_2$	GD-бисимулярность вершин (определение 6)
$\mathcal{T}_1 \sim \mathcal{T}_2$	GD-бисимулярность отмеченных трансдьюсеров
$\mathcal{P}_i$	разбиение на i-й итерации алгоритма
$\mathcal{P}^*$	стабилизированное разбиение

[2] LTS – *Labelled Transition System*, размеченная, или маркированная, система переходов: формальная модель поведения, в которой переходы между состояниями снабжены метками действий, событий или условий наблюдения. Формальное определение используется в разделе 2.3.

2.2. Игровая механика и GD-FST трансдьюсер

Определение 1 (Игровая механика). *Игровой механикой* называется правило, регулирующее взаимодействие игрока с игровой системой: оно описывает, при каких условиях действие доступно, каков его эффект и какова его стоимость [7].

Чтобы проверять эквивалентность механик формально, далее игровая механика рассматривается как конечная графовая модель: вершины соответствуют фазам механики, а рёбра – переходам между фазами, помеченным условиями применимости и числовыми параметрами перехода. Такую модель фиксирует следующий формализм.

Определение 2 (GD-FST трансдьюсер [1, 14]). *GD-FST трансдьюсером* (Game Design Finite State Transducer — конечный трансдьюсер состояний игрового дизайна) называется четвёрка $\mathcal{T} = \langle V, E, \lambda_V, \lambda_E \rangle$, где V – непустое конечное множество *вершин* (игровых состояний); $E \subseteq V \times V$ – множество *рёбер* (переходов); $\lambda_V : V \to \Lambda_V$ – функция разметки вершин наблюдаемыми метками состояний, где Λ_V – конечное множество таких меток; $\lambda_E : E \to \mathrm{Cond} \times \mathbb{R}_{\geq 0}$ – функция пометки рёбер условием охраны и весом.

В простейшем ролевом варианте Λ_V может включать метки `source`, `transform` и `terminal`: `source` обозначает порождающую (входную) вершину, `transform` – промежуточную преобразующую фазу, `terminal` – завершающую (поглощающую) вершину. (В прикладных моделях игровых механик Λ_V может быть богаче и включать фазовые метки, например `READY`, `ACTIVE`, `COOLDOWN`.) Терминальные вершины, если они нужны в конкретной модели, задаются разметкой: $V_W = \{v \in V : \lambda_V(v) = \texttt{terminal}\}$.

Вес $w \in \mathbb{R}_{\geq 0}$ допускает две интерпретации: *детерминированную* (стоимость, урон, длительность), при которой согласование весов есть точное равенство $w_1 = w_2$, и *стохастическую* (вероятность перехода) при дополнительном условии нормировки $\sum_{v'} w(v, v') = 1$ в каждой вершине; стохастический случай подробно разобран в Замечании 3.

Для проверки эквивалентности механик далее рассматривается *отмеченный GD-FST трансдьюсер* $(\mathcal{T}, v_0)$, где $v_0 \in V$ – начальная вершина, или стартовая фаза механики. Если начальные вершины ясны из контекста, запись $\mathcal{T}_1 \sim \mathcal{T}_2$ используется как сокращение для GD-бисимулярности отмеченных трансдьюсеров $(\mathcal{T}_1, v_0^1)$ и $(\mathcal{T}_2, v_0^2)$.

В рамках этого формализма конкретная игровая механика задаётся отмеченным фрагментом GD-FST: набором вершин (фаз механики) и рёбер (переходов между фазами) с пометками условий охраны и весами λ_E.

Объект $\mathcal{T}$ называется *трансдьюсером*, а не просто графом, поскольку несёт двойную разметку: λ_V фиксирует типы состояний, λ_E – условия срабатывания и веса переходов. Трансдьюсерная природа проявляется в том, что механика *сопоставляет* входному состоянию среды, проверяемому охранами, наблюдаемый переход с количественной характеристикой w. Собственно эффекты переходов остаются в объемлющем формализме FAST-GM и в данной работе абстрагируются.

Семантика среды. Охранные условия $\mathrm{cond} \in \mathrm{Cond}$ суть предикаты над конечным множеством переменных среды Var (таймеры, счётчики, ресурсы, флаги состояния; например, $\mathrm{Var} = \{\texttt{timer}, \texttt{ticks}, \texttt{mana}, \texttt{hp}\}$), которым сопоставлена глобальная оценка η, приписывающая каждой переменной значение из её домена. Ребро e с меткой $\lambda_E(e) = (\mathrm{cond}, w)$ проходимо в оценке η, если $\eta \models \mathrm{cond}$. Сами переменные, их домены и функции обновления (эффекты переходов) принадлежат объемлющему формализму сущностей FAST-GM[3] [14], по отношению к которому GD-FST является *проекцией управляющей структуры*: трансдьюсер фиксирует фазы механики и условия и веса переходов между ними, абстрагируясь от конкретной динамики переменных. Соответственно GD-бисимуляция (определение 6) определяется на управляющем уровне *относительно фиксированной разделяемой семантики среды*: оба сравниваемых трансдьюсера интерпретируются над общим словарём Var с общей динамикой η.

2.3. Классическая бисимуляция

Понятие бисимуляции введено Парком [3] и развито Милнером [2] в контексте теории параллельных процессов – CCS (Calculus of Communicating Systems, исчисление взаимодействующих систем) [2]. Для размеченных систем переходов оно определяется следующим образом.

Определение 3 (Размеченная система переходов (LTS)). *Размеченной системой переходов* (Labelled Transition System, LTS) в настоящей работе называется четвёрка

$$\mathcal{S} = (S, \mathcal{A}, \to, \ell),$$

где S — множество состояний, $\mathcal{A}$ — алфавит меток переходов, $\to \subseteq S \times \mathcal{A} \times S$ — отношение переходов, а $\ell : S \to \Lambda$ — функция разметки состояний. Запись $s \xrightarrow{a} s'$ означает, что $(s, a, s') \in \to$.

[3] FAST-GM восходит к работе [14] (*Formal Approach to Spatio-Temporal Game Modeling* — формальный подход к пространственно-временному моделированию игровых систем). В рамках FAST-GM сущность задаётся кортежем $\langle I, X, S_E, T, B\rangle$, а механика – тройкой $\langle \mathrm{pre}, \mathrm{eff}, \mathrm{cost}\rangle$; подробное определение см. [14].

Определение 4 (Бисимуляция [2, 3]). Бинарное отношение $R \subseteq S \times S$ называется *бисимуляцией* на размеченной системе переходов $\mathcal{S} = (S, \mathcal{A}, \to, \ell)$, если для любых s_1, s_2 с $(s_1, s_2) \in R$:

(*i*) $\ell(s_1) = \ell(s_2)$ (совпадение меток состояний);

(*ii*) для любого перехода $s_1 \xrightarrow{a} s_1'$ существует $s_2 \xrightarrow{a} s_2'$ такой, что $(s_1', s_2') \in R$;

(*iii*) симметрично: для любого $s_2 \xrightarrow{a} s_2'$ существует $s_1 \xrightarrow{a} s_1'$ с $(s_1', s_2') \in R$.

Состояния s_1 и s_2 называются *бисимулярными* ($s_1 \sim s_2$), если существует бисимуляция, содержащая пару (s_1, s_2). Наибольшее такое отношение — *бисимуляционная эквивалентность*.

Замечание 1 (Коиндуктивная природа бисимуляции). определение 4 коиндуктивно[4]. Бисимуляции суть постнеподвижные точки монотонного функционала уточнения $\mathcal{F}$, где

$$\mathcal{F}(R) = \{(s_1, s_2) \mid \Phi(s_1, s_2, R)\},$$

а предикат $\Phi(s_1, s_2, R)$ означает, что для пары (s_1, s_2) выполнены условия (*i*)–(*iii*) Определения 4 относительно R. Бисимуляционная эквивалентность $\sim$ — наибольшая неподвижная точка $\nu R.\, \mathcal{F}(R)$, существующая по теореме Кнастера–Тарского ввиду замкнутости множества бисимуляций относительно объединения [6]. Поэтому утверждение $s_1 \sim s_2$ выражает согласованность не одиночного перехода, а всего (вообще говоря, бесконечного) ветвящегося поведения: для каждого конечного или бесконечного пути, исходящего из s_1, существует зеркальный путь из s_2 с попарно бисимулярными состояниями, и наоборот.

Простыми словами: выбор *наибольшей* неподвижной точки означает презумпцию неотличимости — две реализации считаются эквивалентными до тех пор, пока не предъявлена конкретная последовательность переходов, на которой их поведение расходится. «Различимость» требует свидетеля (такой переход и строит алгоритм 1 при ответе $\bot$), тогда как эквивалентность не требует пошаговой проверки всех путей. Именно в этом сила коиндукции: она превращает *локальное* условие («один переход согласован») в *глобальную* гарантию («всё бесконечное поведение

[4] Коиндукция — двойственный к индукции принцип. Индуктивное определение строит множество «снизу вверх» от базовых случаев как *наименьшую* неподвижную точку; коиндуктивное задаёт *наибольшую* неподвижную точку — наибольшее отношение, замкнутое относительно порождающих правил. Применительно к бисимуляции это означает, что эквивалентными признаются все состояния, которые *не удаётся различить* никаким (в общем случае бесконечным) наблюдением за переходами.

эквивалентно») — и на этом принципе построены доказательства корректности и полноты настоящей работы (Леммы 1–2), где локальная согласованность переходов через инвариант разбиения распространяется на всё поведение трансдьюсера.

Ключевое свойство бисимуляции: бисимулярные системы удовлетворяют одним и тем же формулам CTL и LTL [6, теорема 7.14]. Это делает бисимуляцию мощным инструментом верификации: однократная проверка бисимулярности переносит на обе реализации *все* свойства, выразимые в CTL и LTL, включая заранее не сформулированные, без их поштучной проверки. (Это спецификационно-независимая гарантия эквивалентности поведения, а не обязательно более дешёвая альтернатива проверке одного свойства: выигрыш в стоимости проявляется при переносе многих свойств или при замене системы её бисимулярным, но меньшим представлением.)

2.4. Алгоритм Канеллакиса–Смолки

Классический алгоритм проверки бисимуляционной эквивалентности [15] основан на методе *разбиения (partition refinement)*, восходящем к процедуре минимизации детерминированных конечных автоматов (ДКА) Хопкрофта [10]. ДКА — автомат, в котором для каждого состояния и каждого входного символа переход определён однозначно [16]. Алгоритм Хопкрофта строит наименьший (по числу состояний) ДКА, принимающий тот же язык, за время $O(n \log n)$, последовательно разбивая состояния на классы эквивалентности. В алгоритме Канеллакиса–Смолки та же идея применяется к двум автоматам одновременно.

Идея. Начинаем с грубого разбиения состояний: все состояния с одинаковой меткой $\lambda_V(v)$ в одном блоке. Затем итеративно «дробим» блоки: если два состояния v_1 и v_2 из одного блока имеют переходы по одной метке в *разные* блоки, они не могут быть бисимулярны и должны быть разделены. Процедура завершается, когда разбиение стабилизируется; итоговые блоки — классы бисимуляционной эквивалентности.

В базовой формулировке алгоритм Канеллакиса–Смолки выполняет $O(|V|)$ проходов уточнения, каждый из которых просматривает все рёбра, и работает за время $O(|E| \cdot |V|)$. Логарифмическая оценка $O(|E| \cdot \log |V|)$, совпадающая с оценкой алгоритма Хопкрофта для ДКА, достигается оптимизацией Пейджа–Тарьяна [17]: при разбиении блока на две части в качестве разделителя[5] обрабатывается *меньшая* из них, благодаря

[5] Термин «разделитель» соответствует англоязычному *splitter* в литературе по алгоритмам разбиения (partition refinement); этим же английским термином названа переменная *Splitter* в алгоритме 1.

чему рёбра каждой вершины участвуют в уточнении не более $O(\log |V|)$ раз. Именно эта оптимизированная схема используется в реализации (замечание 7).

3. GD-бисимуляция: авторское расширение

3.1. Определение GD-бисимуляции

Стандартная бисимуляция (определение 4) не применима к GD-FST напрямую: метка ребра в GD-FST – это пара (cond, w), а не одиночный символ. Требуется определить, когда два ребра считаются «согласованными».

Авторское нововведение состоит в введении отношения совместимости рёбер, учитывающего как логическую эквивалентность условий охраны, так и совпадение весов. Это расширение нетривиально: проверка логической эквивалентности условий сводится к задаче выполнимости булевых формул или к SMT-проверке для формул с арифметическими ограничениями[6], что принципиально сложнее проверки равенства меток, а совпадение весов требует отдельного обсуждения для стохастических систем. Причина в том, что интерпретация веса различна: для *детерминированных* атрибутов (урон, стоимость, длительность) точное равенство $w_1 = w_2$ не вызывает вопросов, тогда как для *вероятностного* веса на исходящие из вершины веса наложено ограничение нормировки $\sum_{v'} w(v, v') = 1$, и сама эквивалентность распадается на *точную* (равенство вероятностей) и *приближённую* (ε-близость), задающие разные отношения; этот случай разобран в Замечании 3.

Определение 5 (Совместимость рёбер). Пусть сравниваются два трансдьюсера $\mathcal{T}_1$ и $\mathcal{T}_2$, и пусть e и e' – рёбра из дизъюнктного объединения $E_1 \sqcup E_2$. Для ребра $e \in E_i$ используется соответствующая функция пометки λ_E^i; то есть $\lambda_E(e)$ ниже означает $\lambda_E^i(e)$.

Отношение совместимости задаётся именно на всём объединении $E_1 \sqcup E_2$, а не только между ребром первого и ребром второго трансдьюсера. В самом определении бисимуляции далее сравниваются переходы из разных состояний, однако алгоритму нужен единый алфавит классов меток для обоих трансдьюсеров. Поэтому на этапе предобработки необходимо уметь помещать в один класс любые два ребра рассматриваемого объединения, в том числе два ребра одного трансдьюсера. Иначе две логически эквивалентные, но синтаксически различные охраны внутри одной модели могли бы попасть в разные классы и сделать проверку излишне строгой.

[6] SAT — задача проверки выполнимости пропозициональной формулы; классическая NP-полнота SAT установлена Куком [18]. SMT обобщает SAT на формулы относительно фоновых теорий, например линейной арифметики; пример промышленного SMT-решателя – Z3 [19].

Рёбра e и e' называются $(cond, w)$-*совместимыми*, если:

(*i*) $\lambda_E(e) = (\mathrm{cond}, w)$, $\lambda_E(e') = (\mathrm{cond}', w')$;
(*ii*) $\mathrm{cond} \equiv \mathrm{cond}'$ (логическая эквивалентность условий охраны);
(*iii*) $w = w'$ (равенство весов).

Обозначение: $e \simeq e'$.

Замечание 2 (Логическая эквивалентность условий). Условия охраны из Cond рассматриваются как формулы над конечным множеством атомарных предикатов. Если атомарные предикаты трактуются как неделимые булевы переменные, проверка $\mathrm{cond}_1 \equiv \mathrm{cond}_2$ сводится к проверке невыполнимости формулы $\neg(\mathrm{cond}_1 \leftrightarrow \mathrm{cond}_2)$, то есть к задаче SAT (Boolean SATisfiability, выполнимость булевых формул) [18]. Если же атомарные предикаты содержат арифметические ограничения над числовыми переменными (например, $\texttt{timer} \geq 333$ или $\texttt{mana} \geq 5$), используется SMT-проверка (Satisfiability Modulo Theories) в соответствующей теории [19]. В общем случае SAT-задача NP-полна [18, 20]. Однако для типичных игровых условий охраны с небольшим числом атомарных предикатов такая проверка обычно не является узким местом, поскольку выполняется однократно на этапе предобработки.

Замечание 3 (Веса и стохастическая эквивалентность). При $w \in [0, 1]$ и $\sum_{v'} w(v, v') = 1$ (стохастический случай) определение 5(*iii*) задаёт строгое согласование вероятностных весов на уровне отдельных рёбер. Оно близко по духу к вероятностной бисимуляции Ларсена–Скоу [21], однако не совпадает с её классическим определением: в вероятностной бисимуляции сравниваются суммарные вероятности перехода в классы эквивалентности, а не обязательно веса отдельных рёбер. Поэтому настоящая статья рассматривает точную структурную версию согласования весов. Для ε-приближённой версии (когда допустимо $|w_1 - w_2| \leq \varepsilon$) алгоритм обобщается заменой равенства на ε-близость; полное исследование вероятностного обобщения составляет перспективу будущих работ.

Теперь можно дать центральное определение статьи.

Определение 6 (GD-бисимуляция). Пусть $\mathcal{T}_1 = \langle V_1, E_1, \lambda_V^1, \lambda_E^1 \rangle$ и $\mathcal{T}_2 = \langle V_2, E_2, \lambda_V^2, \lambda_E^2 \rangle$ – два GD-FST трансдьюсера, и пусть в них выделены начальные вершины $v_0^1 \in V_1$ и $v_0^2 \in V_2$. Бинарное отношение $R \subseteq V_1 \times V_2$ называется *GD-бисимуляцией*, если для любых $(v_1, v_2) \in R$:

(*i*) $\lambda_V^1(v_1) = \lambda_V^2(v_2)$ (совпадение наблюдаемых меток вершин);

(*ii*) для любого ребра $e_1 = (v_1, v_1') \in E_1$ существует ребро $e_2 = (v_2, v_2') \in E_2$ такое, что $e_1 \simeq e_2$ и $(v_1', v_2') \in R$;

(*iii*) для любого ребра $e_2 = (v_2, v_2') \in E_2$ существует ребро $e_1 = (v_1, v_1') \in E_1$ такое, что $e_1 \simeq e_2$ и $(v_1', v_2') \in R$.

Отмеченные трансдьюсеры $(\mathcal{T}_1, v_0^1)$ и $(\mathcal{T}_2, v_0^2)$ называются *GD-бисимулярными* ($\mathcal{T}_1 \sim \mathcal{T}_2$), если существует GD-бисимуляция R такая, что $(v_0^1, v_0^2) \in R$.

Иными словами, два отмеченных трансдьюсера GD-бисимулярны, если любой игровой сценарий, реализуемый в $\mathcal{T}_1$, может быть воспроизведён в $\mathcal{T}_2$ с теми же наблюдаемыми фазами, совместимыми условиями охраны и совпадающими весами переходов, и наоборот. При этом GD-бисимуляция не требует структурной тождественности трансдьюсеров: допускаются различия в именах вершин, внутренней детализации состояний и способе разбиения поведения на промежуточные шаги, если эти различия не проявляются на уровне выбранной разметки состояний и совместимости переходов. Поэтому речь идёт не об изоморфизме графов, а о поведенческой эквивалентности моделей в заданной наблюдаемой абстракции.

3.2. Свойства GD-бисимуляции

Замечание 4 (Область применимости относительно среды). Поскольку оба трансдьюсера интерпретируются над общим словарём переменных, логическая эквивалентность охран $\mathrm{cond}_1 \equiv \mathrm{cond}_2$ означает их срабатывание при тождественных состояниях среды. Поэтому GD-бисимуляция *достаточна* для эквивалентности управляющего поведения при фиксированной общей интерпретации среды и одинаковой семантике эффектов переходов. Обратное верно лишь относительно выбранного словаря: определение *не* обнаруживает эквивалентности, для установления которых требуется нетривиальный морфизм между *разными* словарями переменных.

На языке геймдизайна: если одна реализация измеряет кулдаун в миллисекундах, а другая — в дискретных тиках, их охраны записаны над несопоставимыми атомами и будут признаны несовместимыми, даже когда соответствуют одному физическому интервалу (раздел 6); установление такой эквивалентности требует явного отображения сред и составляет открытую задачу.

Детерминизм, недетерминизм и выбор бисимуляции. определение 2 не требует детерминизма: из одной вершины может исходить несколько

рёбер, и если их охранные условия одновременно выполнимы, переход недетерминирован. GD-бисимуляция (определение 6) и алгоритм 1 обрабатывают этот случай штатно – условия (*ii*)–(*iii*) формулируются через «для каждого ребра существует совместимое», что является стандартной трактовкой ветвящегося недетерминизма [6] и не опирается на единственность перехода. Назовём GD-FST *детерминированным* в вершине v, если охранные условия исходящих из v рёбер попарно несовместны (их конъюнкция невыполнима), то есть в любом состоянии среды доступен не более чем один переход; корректно построенные механики, как правило, детерминированы в этом смысле.

Выбор *бисимуляции*, а не более слабой следовой эквивалентности, оправдан с двух сторон: на полностью детерминированных LTS-проекциях следовая эквивалентность и бисимуляция совпадают [6]; для GD-FST с недетерминированными или перекрывающимися охранами бисимуляция является более строгим отношением и лучше отражает важное для геймдизайнера свойство – *какие* действия становятся доступны в каждой фазе, а не только в каком порядке они наблюдаются в отдельной партии.

Граничные случаи охран укладываются в ту же схему: $\mathrm{cond} \equiv \top$ – безусловный (всегда доступный) переход; $\mathrm{cond} \equiv \bot$ – «мёртвое» ребро, которое никогда не срабатывает и удаляется при нормализации входа, что необходимо для совпадения структурного сравнения с семантическим.

Замечание 5 (Наибольшая GD-бисимуляция). Объединение произвольного числа GD-бисимуляций является GD-бисимуляцией. В самом деле, пусть $R = \bigcup_i R_i$, где каждое R_i – GD-бисимуляция, и $(v_1, v_2) \in R$. Тогда $(v_1, v_2) \in R_j$ для некоторого j, и условия (*i*)–(*iii*) Определения 6 выполнены для этой пары относительно R_j; поскольку они формулируются через *существование* совместимого ребра с образом в том же отношении, а $R_j \subseteq R$, эти образы принадлежат и R – значит, условия выполнены и относительно R. Следовательно, существует *наибольшая* GD-бисимуляция $\sim$ — отношение GD-бисимуляционной эквивалентности. Это стандартный результат теории бисимуляций [2], немедленно переносящийся на GD-FST в силу сохранения структуры определения.

Следствие 1 (Сохранение формул темпоральных логик). Пусть $\mathcal{T}_1 \sim \mathcal{T}_2$. Тогда для любой формулы φ темпоральной логики CTL или LTL, атомарные высказывания которой интерпретируются над метками вершин λ_V:

$$\mathcal{T}_1 \models \varphi \iff \mathcal{T}_2 \models \varphi.$$

Доказательство. Для пары трансдьюсеров $\mathcal{T}_1, \mathcal{T}_2$ сначала соберём множество всех меток рёбер обоих трансдьюсеров:

$$M = \{\lambda_E^1(e) \mid e \in E_1\} \cup \{\lambda_E^2(e) \mid e \in E_2\}, \qquad K = M/{\approx_K}.$$

Здесь K – общий алфавит классов совместимых меток, а отношение $\approx_K$ задаётся так:

$$(\mathrm{cond}, w) \approx_K (\mathrm{cond}', w') \quad \Longleftrightarrow \quad \mathrm{cond} \equiv \mathrm{cond}' \quad \wedge \quad w = w'.$$

Из каждого трансдьюсера построим размеченную систему переходов над этим общим алфавитом:

$$\mathrm{LTS}_K(\mathcal{T}_i) = (V_i, K, \to_i, \lambda_V^i), \qquad i \in \{1, 2\},$$

где $(v, \kappa, v') \in \to_i$ тогда и только тогда, когда $(v, v') \in E_i$ и класс метки $\lambda_E^i(v, v')$ равен κ.

Факторизация по $\approx_K$ существенна: синтаксически различные метки (cond_1, w_1) и (cond_2, w_2), связанные отношением $\approx_K$ (логически эквивалентные охраны при равных весах), отождествляются в один символ алфавита K. В полученных таким образом LTS условие совместимости $e_1 \simeq e_2$ (определение 5) превращается в точное совпадение меток-классов. Следовательно, GD-бисимуляция R (определение 6) есть стандартная бисимуляция между $\mathrm{LTS}_K(\mathcal{T}_1)$ и $\mathrm{LTS}_K(\mathcal{T}_2)$. Применяя теорему 7.14 из [6] о сохранении формул CTL и LTL при стандартной бисимуляции, получаем $\mathcal{T}_1 \models \varphi \iff \mathcal{T}_2 \models \varphi$. □ □

Следствие 1 является ключевым обоснованием практической ценности GD-бисимуляции: если две реализации механики GD-бисимулярны, они *не могут быть различены* никаким верификационным свойством, выраженным в CTL или LTL. Это означает, что их можно взаимозаменять в любом верификационном пайплайне без изменения результатов проверки.

Замечание 6 (Композициональность – перспектива). Естественным продолжением является вопрос о *композициональности*: сохраняется ли GD-бисимуляция при параллельной композиции механик, то есть влечёт ли $\mathcal{T}_1 \sim \mathcal{T}_1'$ и $\mathcal{T}_2 \sim \mathcal{T}_2'$ эквивалентность композиций $\mathcal{T}_1 \otimes \mathcal{T}_2 \sim \mathcal{T}_1' \otimes \mathcal{T}_2'$. Положительный ответ означал бы возможность *модульного* рефакторинга: замена отдельной механики на GD-бисимулярную сохраняла бы поведение всей игровой системы. Для классических исчислений процессов конгруэнтность относительно параллельной композиции – известный результат [2], однако его перенос на GD-FST требует предварительной формализации самого оператора композиции для рёбер с составными метками (cond, w),

и в первую очередь – фиксации семантики комбинирования весов при синхронизации переходов (произведение вероятностей, сумма стоимостей или иной закон в зависимости от интерпретации веса). Эта формализация и доказательство конгруэнтности составляют предмет отдельного исследования и в настоящей работе не приводятся.

4. Алгоритм проверки GD-бисимуляции

4.1. Описание алгоритма

Алгоритм 1 строится по схеме Канеллакиса–Смолки [15], адаптированной к структуре GD-FST. Адаптация заключается в замене проверки равенства меток рёбер проверкой (cond, w)-совместимости (определение 5). Это единственное, но принципиальное отличие от классического алгоритма.

Поскольку условия охраны сравниваются по *логической* эквивалентности (замечание 2), а не синтаксически, метки рёбер предварительно группируются в *классы совместимости*. Обозначим отношение эквивалентности на метках рёбер через $\approx_K$:

$$(\mathrm{cond}, w) \approx_K (\mathrm{cond}', w') \quad \Longleftrightarrow \quad \mathrm{cond} \equiv \mathrm{cond}' \quad \wedge \quad w = w'.$$

Дальнейшее уточнение разбиения ведётся по классам K, а не по синтаксическим парам меток. Это устраняет риск развести логически эквивалентные, но синтаксически различные переходы.

Вход: трансдьюсеры $\mathcal{T}_1$ и $\mathcal{T}_2$. **Выход**: $\top$ (бисимулярны) или $\bot$ (не бисимулярны), при необходимости – свидетель различия.

В алгоритме 1 через λ_V и λ_E обозначены объединённые функции разметки на дизъюнктных объединениях $V_1 \sqcup V_2$ и $E_1 \sqcup E_2$: на элементах первого трансдьюсера они совпадают с λ_V^1, λ_E^1, а на элементах второго – с λ_V^2, λ_E^2.

Перед запуском алгоритма 1 предполагается нормализация входных трансдьюсеров: рёбра с невыполнимой охраной $\mathrm{cond} \equiv \bot$ удаляются, поскольку они не могут быть пройдены ни при каком состоянии среды и не должны влиять на поведенческую эквивалентность.

Алгоритм 1. Проверка GD-бисимуляционной эквивалентности

$V \leftarrow V_1 \sqcup V_2$ ▷ объединение вершин
$E \leftarrow E_1 \sqcup E_2$ ▷ объединение рёбер
$\mathcal{P}_0 \leftarrow \{\{v \in V : \lambda_V(v) = \ell\} : \ell \in \Lambda_V\}$ ▷ по меткам вершин
$\mathcal{P}_0 \leftarrow \mathcal{P}_0 \setminus \{\emptyset\}$ ▷ исключить пустые блоки
$M \leftarrow \{\lambda_E^1(e) : e \in E_1\} \cup \{\lambda_E^2(e) : e \in E_2\}$ ▷ метки рёбер
$K \leftarrow M/{\approx_K}$ ▷ классы меток
for all $e \in E$ **do**
    $\mathrm{cls}(e) \leftarrow [\lambda_E(e)]_{\approx_K}$ ▷ класс ребра
end for
$i \leftarrow 0$
repeat
    $\mathcal{P}_{i+1} \leftarrow \mathcal{P}_i$
    for all $B \in \mathcal{P}_i$ и $\kappa \in K$ **do**
        $\mathit{Splitter} \leftarrow \{v \in V : \exists\, e = (v, v') \in E, \quad \mathrm{cls}(e) = \kappa, \quad v' \in B\}$
        for all $C \in \mathcal{P}_{i+1}$ такое, что $C \cap \mathit{Splitter} \neq \emptyset$ и $C \not\subseteq \mathit{Splitter}$ **do**
            заменить C на $C \cap \mathit{Splitter}$ и $C \setminus \mathit{Splitter}$
        end for
    end for
    $i \leftarrow i + 1$
until $\mathcal{P}_i = \mathcal{P}_{i-1}$ ▷ стабилизация
if $\exists B \in \mathcal{P}_i : \{v_0^1, v_0^2\} \subseteq B$ **then**
    return $\top$
else
    return $(\bot, \mathit{Witness})$ ▷ свидетель различия
end if

В случае возврата $\bot$ алгоритм 1 может дополнительно возвращать свидетель различия

$$\mathit{Witness} = (C, B, \kappa),$$

где C – блок текущего разбиения, содержащий сравниваемые вершины, B – целевой блок, а $\kappa \in K$ – класс метки перехода, относительно которого блок C был разделён. Формально свидетель задаётся условием

$$C \cap \mathit{Splitter}_{B,\kappa} \neq \emptyset \quad \text{и} \quad C \not\subseteq \mathit{Splitter}_{B,\kappa},$$

где

$$\mathit{Splitter}_{B,\kappa} = \{v \in V : \exists\, e = (v, v') \in E, \quad \mathrm{cls}(e) = \kappa, \quad v' \in B\}.$$

Иными словами, в блоке C находятся вершины, одна часть которых имеет переход класса κ в блок B, а другая часть такого перехода не имеет. Поэтому эти вершины не могут оставаться в одном классе бисимуляционной эквивалентности.

Если в результате такого разбиения v_0^1 и v_0^2 оказываются в разных блоках, то тройка (C, B, κ) фиксирует локальную причину неэквивалентности начальных состояний.

Содержательный смысл алгоритма 1. Начинаем с грубого разбиения: все вершины с одинаковой наблюдаемой меткой λ_V помещаются в один блок. Затем итеративно уточняем разбиение: если два состояния из одного блока различаются набором достижимых блоков по некоторому классу меток $\kappa \in K$, они не могут быть GD-бисимулярны – разделяем их. Процедура завершается, когда никакое разделение невозможно. Начальные вершины v_0^1 и v_0^2 GD-бисимулярны тогда и только тогда, когда они остались в одном блоке.

Недетерминизм обрабатывается корректно и не требует специальной обработки. Если из вершины исходит несколько рёбер одного класса κ в *разные* целевые блоки, это не мешает алгоритму 1: при обработке каждого целевого блока B множество $\mathit{Splitter}(B, \kappa)$ содержит ровно те вершины, у которых есть κ-переход именно в B, поэтому две вершины с различающимися наборами достижимых по κ блоков будут разделены на соответствующей итерации. Это отражает определение GD-бисимуляции, требующее для *каждого* ребра существования совместимого, а не биекции рёбер.

4.2. Вычислительная сложность

Замечание 7 (Сложность Алгоритма 1). Пусть $n = |V_1| + |V_2|$, $m = |E_1| + |E_2|$, $k = |K|$ – число классов совместимости меток рёбер (Алгоритм 1, предобработка), $k \le m$. В буквальной реализации псевдокода один проход уточнения рассматривает до $|\mathcal{P}_i| \cdot k \le nk$ разделителей, а вычисление каждого множества *Splitter* прямым просмотром всех рёбер стоит $O(m)$. Так как число проходов ограничено n (каждый результативный проход увеличивает число блоков, а блоков не более n), наивная верхняя оценка составляет $O(k \cdot m \cdot n^2)$.

Эта оценка описывает именно прямую реализацию псевдокода, удобную для доказательства корректности. В практической версии используется оптимизация Пейджа–Тарьяна [17]: для каждого класса меток поддерживаются обратные списки предшественников, а при разбиении блока в качестве разделителя обрабатывается меньшая часть. Это снижает стоимость цикла уточнения до $O(k \cdot m \cdot \log n)$ в используемой оценке. При фиксированном или малом числе классов меток k такая реализация работает близко к линейно-логарифмической по числу рёбер;

в общем случае зависимость от k остаётся существенной. На практике число различных типов переходов в механике обычно намного меньше числа самих переходов, поэтому проверка остаётся интерактивной для библиотек игровых механик.

Проверки логической эквивалентности охранных условий выполняются однократно на этапе предобработки: метки рёбер кластеризуются по отношению $\approx_K$ (замечание 2), после чего каждому ребру сопоставляется идентификатор класса совместимости, а проверка $e_1 \simeq e_2$ сводится к сравнению идентификаторов классов за $O(1)$. Тем самым стоимость SAT/SMT-проверок вынесена из основного цикла уточнения и не зависит от числа его проходов.

Само построение K требует попарного сравнения меток по отношению $\approx_K - O(m^2)$ проверок в худшем случае (или $O(m \cdot k)$ при сравнении лишь с представителями уже образованных классов). Эта стоимость *не* входит в оценку цикла уточнения: она однократна и на практике мала для охран с небольшим числом атомарных предикатов.

Замечание 8 (Практическая реализация весов и условий охраны). **Вещественные веса.** Точное равенство $w_1 = w_2$ проблематично для вещественных чисел из-за ошибок округления. В реализации платформы Tula веса хранятся как рациональные числа (тип `fractions.Fraction` языка Python), что обеспечивает точное сравнение. Для ε-приближённой версии ($|w_1 - w_2| \leq \varepsilon$) алгоритм обобщается заменой равенства на ε-близость; полная формализация составляет перспективу будущих работ.

Проверка условий охраны. Эквивалентность двух охран проверяется SMT-решателем Z3 [19]: строится формула

$$\neg(\mathrm{cond}_1 \leftrightarrow \mathrm{cond}_2),$$

и проверяется её невыполнимость. Если такая формула невыполнима, то не существует состояния среды, в котором две охраны дают разные истинностные значения. Для типичных игровых условий охраны с небольшим числом атомарных предикатов такая проверка практически не влияет на основной цикл уточнения разбиения.

5. Корректность и полнота алгоритма

В доказательстве рассматривается дизъюнктное объединение двух трансдьюсеров, как и в алгоритме 1. Поэтому запись $v_1 \sim v_2$ в леммах ниже означает принадлежность пары вершин к наибольшей GD-бисимуляции на этом объединении; её ограничение на $V_1 \times V_2$ совпадает с отношением из Определения 6.

Теорема 1 (Корректность и полнота алгоритма 1). Алгоритм 1 возвращает $\top$ тогда и только тогда, когда $\mathcal{T}_1 \sim \mathcal{T}_2$.

Доказательство проводится по классической схеме теории алгоритмов разбиения [6, 10, 15]. Ниже приводится полное доказательство, адаптированное к структуре GD-FST. Авторский вклад – в установлении того, что адаптация с одиночных меток на классы совместимости рёбер сохраняет все инварианты классического доказательства.

Лемма 1 (Инвариант разбиения). В каждый момент работы Алгоритма 1 выполняется: если $v_1 \sim v_2$, то v_1 и v_2 находятся в одном блоке разбиения $\mathcal{P}_i$.

Доказательство по индукции по номеру итерации i.

База. $\mathcal{P}_0$ группирует вершины по λ_V-меткам. По условию (*i*) определения 6 $v_1 \sim v_2$ влечёт $\lambda_V(v_1) = \lambda_V(v_2)$, т. е. v_1 и v_2 в одном начальном блоке.

Шаг. Предположим, что инвариант выполнен для $\mathcal{P}_i$, и пусть на следующей итерации некоторый блок $C \ni v_1, v_2$ дробится разделителем, построенным по блоку $B \in \mathcal{P}_i$ и классу $\kappa \in K$. Разделение означает, что одна из вершин имеет переход класса κ в B, а другая нет. Покажем, что для $v_1 \sim v_2$ это невозможно. Пусть v_1 имеет ребро $e_1 = (v_1, v_1')$ с $\mathrm{cls}(e_1) = \kappa$ и $v_1' \in B$. По условию (*ii*) определения 6 существует ребро $e_2 = (v_2, v_2')$ с $e_1 \simeq e_2$ (то есть $\mathrm{cls}(e_2) = \kappa$) и $v_1' \sim v_2'$. По индукционному предположению из $v_1' \sim v_2'$ следует, что v_1' и v_2' лежат в одном блоке $\mathcal{P}_i$; а поскольку $v_1' \in B$ и блоки $\mathcal{P}_i$ попарно не пересекаются, то $v_2' \in B$. Значит, v_2 также имеет переход класса κ в B. Случай перехода из v_2 рассматривается симметрично. Поэтому обе вершины одновременно имеют либо не имеют переход класса κ в B и не могут быть разделены данным разделителем. □ □

Лемма 1 означает содержательно: алгоритм никогда не разделяет вершины, которые обязаны быть в одном классе эквивалентности. Это гарантирует *корректность* (отсутствие ложных отрицаний).

Лемма 2 (Полнота разбиения). Стабилизированное разбиение $\mathcal{P}^*$ является наибольшей GD-бисимуляцией, т. е. v_1 и v_2 находятся в одном блоке $\mathcal{P}^*$ тогда и только тогда, когда $v_1 \sim v_2$.

Доказательство. Обозначим через $\approx$ отношение «быть в одном блоке» $\mathcal{P}^*$. Достаточно показать, что $\approx$ является GD-бисимуляцией.

Пусть $v_1 \approx v_2$. Тогда:

1. $\lambda_V(v_1) = \lambda_V(v_2)$: по построению $\mathcal{P}_0$.

2. Пусть $e_1 = (v_1, v_1')$ – произвольное ребро с $\mathrm{cls}(e_1) = \kappa$, и пусть $B \in \mathcal{P}^*$ – блок, содержащий v_1'. Поскольку $\mathcal{P}^*$ стабильно, разделитель пары (B, κ) уже был обработан и *не* разбил блок, содержащий v_1 и v_2; следовательно, по построению множества *Splitter* для него обе вершины одновременно либо принадлежат ему, либо нет. Так как $v_1 \in$ *Splitter* (у неё есть переход класса κ в B), то и $v_2 \in$ *Splitter*, то есть существует ребро $e_2 = (v_2, v_2')$ с $\mathrm{cls}(e_2) = \kappa$ (а значит $e_1 \simeq e_2$) и $v_2' \in B$, откуда $v_1' \approx v_2'$.

3. Симметрично.

Таким образом, $\approx$ – GD-бисимуляция. Поскольку по лемме 1 все GD-бисимулярные пары находятся в одном блоке, $\approx$ является наибольшей GD-бисимуляцией. □ □

Доказательство теоремы 1. Объединим обе леммы. По определению GD-бисимулярности трансдьюсеров (определение 6) условие $\mathcal{T}_1 \sim \mathcal{T}_2$ равносильно $v_0^1 \sim v_0^2$. Далее, $v_0^1 \sim v_0^2$ выполнено тогда и только тогда, когда v_0^1 и v_0^2 лежат в одном блоке стабилизированного разбиения $\mathcal{P}^*$: импликация «⇒» обеспечена леммой 1 (бисимулярные вершины никогда не разносятся по разным блокам), а «⇐» – леммой 2 ($\approx$ есть GD-бисимуляция, поэтому совпадение блоков влечёт бисимулярность). Наконец, заключительная проверка псевдокода возвращает $\top$ в точности тогда, когда v_0^1 и v_0^2 оказались в одном блоке. Связывая полученные равносильности, имеем

$$\begin{aligned} \mathcal{T}_1 \sim \mathcal{T}_2 \quad &\iff \quad v_0^1 \sim v_0^2, \\ &\iff \quad \exists B \in \mathcal{P}^* : \{v_0^1, v_0^2\} \subseteq B, \\ &\iff \quad \text{алгоритм возвращает } \top. \end{aligned}$$

Значит, алгоритм корректен: он возвращает $\top$ ровно для GD-бисимулярных отмеченных трансдьюсеров. □

□

6. Пример: две реализации механики восстановления способности

Рассмотрим две реализации механики восстановления способности (или кулдауна, от англ. cooldown) – одной из самых распространённых механик в играх с активными способностями, где перезарядка напрямую влияет на баланс. Обе реализации показаны на рисунке 1.

Реализация $\mathcal{T}_1$ (таймер). Три вершины $\{v_R, v_A, v_C\}$ с фазовыми метками

$$\lambda_V(v_R) = \texttt{READY}, \quad \lambda_V(v_A) = \texttt{ACTIVE}, \quad \lambda_V(v_C) = \texttt{COOLDOWN}$$

и переходами

$$\lambda_E(v_R, v_A) = (\texttt{ready},\ 1.0),$$
$$\lambda_E(v_A, v_C) = (\texttt{active},\ 1.0),$$
$$\lambda_E(v_C, v_R) = (\texttt{timer} \geq 333,\ 1.0).$$

Реализация $\mathcal{T}_2$ (счётчик). Три вершины $\{u_R, u_A, u_C\}$ с фазовыми метками

$$\lambda_V(u_R) = \texttt{READY}, \quad \lambda_V(u_A) = \texttt{ACTIVE}, \quad \lambda_V(u_C) = \texttt{COOLDOWN}$$

и переходами

$$\lambda_E(u_R, u_A) = (\texttt{ready},\ 1.0),$$
$$\lambda_E(u_A, u_C) = (\texttt{active},\ 1.0),$$
$$\lambda_E(u_C, u_R) = (\texttt{ticks} \geq 20,\ 1.0).$$

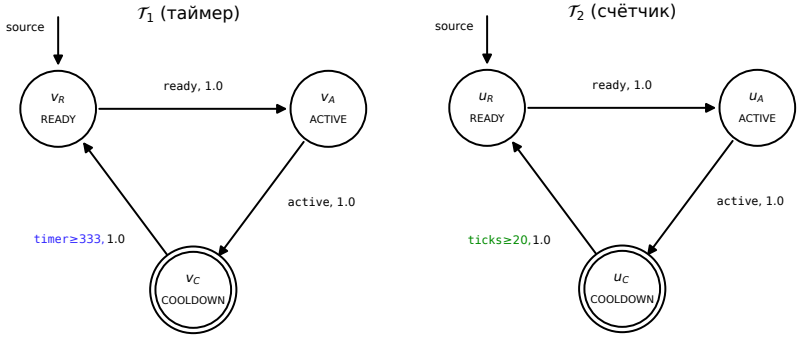

Рисунок 1. Две реализации механики восстановления способности

Пусть начальные вершины – $v_0^1 = v_R$ и $v_0^2 = u_R$. Структурно идентичные трансдьюсеры $\mathcal{T}_1$ (таймер) и $\mathcal{T}_2$ (счётчик) совпадают по фазам (READY, ACTIVE, COOLDOWN) и по двум переходам, различаясь лишь охранным условием перехода возврата – $\texttt{timer} \geq 333$ против $\texttt{ticks} \geq 20$.

Вопрос: верно ли, что $\mathcal{T}_1 \sim \mathcal{T}_2$?

Применениеалгоритма 1. Начальное разбиение строится по фазовым меткам:

$$\mathcal{P}_0 = \{\{v_R, u_R\}, \{v_A, u_A\}, \{v_C, u_C\}\}.$$

На первой итерации рассматриваем класс совместимости κ_{timer}, содержащий метку ($\texttt{timer} \geq 333$, 1.0), и блок $B_R = \{v_R, u_R\}$. У v_C есть переход класса κ_{timer} в B_R, а у u_C такого перехода нет: условие $\texttt{ticks} \geq 20$ логически не эквивалентно условию $\texttt{timer} \geq 333$ над несогласованными словарями переменных. Поэтому блок $\{v_C, u_C\}$ разбивается.

После этого различие распространяется назад. На следующем уточнении разделяются вершины v_A и u_A, поскольку их переходы по классу (`active`, 1.0) ведут в уже блоки фазы `COOLDOWN`. Затем аналогично разделяются начальные вершины v_R и u_R, так как их переходы по классу (`ready`, 1.0) ведут в уже разные блоки фазы `ACTIVE`. Следовательно, алгоритм возвращает $\bot$.

Вывод: $\mathcal{T}_1 \not\sim \mathcal{T}_2$. Несмотря на топологическую идентичность, две реализации *не* эквивалентны по условиям охраны в предположении несогласованных словарей переменных (замечание 4). Геймдизайнер получает свидетель различия: сначала расходится фаза `COOLDOWN`, а затем это различие распространяется до начальных вершин фазы `READY`.

Когда они были бы эквивалентны? Если оба условия определены над согласованными предикатами и логически эквивалентны: $\texttt{timer} \geq 333 \equiv \texttt{ticks} \geq 20$ (например, если задан морфизм среды, согласно которому `ticks` подсчитывают 16,65 мс-тики, то 20 тиков соответствуют 333 мс). В этом случае алгоритм 1 вернул бы $\top$, и геймдизайнер получил бы формальное подтверждение поведенческой эквивалентности.

Положительный случай. Рассмотрим успешный рефакторинг над *общим* словарём предикатов: механика с охраной $\texttt{mana} \geq 5 \wedge \texttt{hp} > 0$ переписана как $\texttt{hp} > 0 \wedge \texttt{mana} \geq 5$ (изменён порядок конъюнктов). Синтаксически метки различны, однако $\texttt{mana} \geq 5 \wedge \texttt{hp} > 0 \equiv \texttt{hp} > 0 \wedge \texttt{mana} \geq 5$, поэтому на этапе предобработки (SMT-проверка) оба ребра попадают в один класс совместимости, разбиение не дробится, и алгоритм 1 возвращает $\top$. Этот пример показывает роль предобработки: без факторизации по $\equiv$ наивное посимвольное сравнение меток ошибочно сочло бы две реализации различными.

Неигровой пример: рефакторинг workflow согласования. Ранее упоминалось, что предлагаемый формализм не привязан к играм: те же определения применимы к любой системе, управляемой правилами с охранными условиями и весами. Рассмотрим процесс согласования

заявки с фазами `DRAFT`, `REVIEW`, `APPROVED`. В реализации $\mathcal{T}_1$ переход из `REVIEW` в `APPROVED` разрешён охраной $\texttt{amount} \leqslant 1000 \wedge \texttt{role} = \texttt{manager}$, в переработанной реализации $\mathcal{T}_2$ — эквивалентной охраной $\texttt{role} = \texttt{manager} \wedge \texttt{amount} \leqslant 1000$ (порядок конъюнктов изменён при рефакторинге). При равных весах переходов метки логически эквивалентны, предобработка помещает их в один класс совместимости, и алгоритм 1 возвращает $\top$: рефакторинг workflow не изменил наблюдаемого поведения. Тем самым проверка бисимуляционной эквивалентности служит инструментом верификации рефакторинга и вне игровой области — всюду, где поведение задаётся размеченными переходами с условиями и весами.

7. Заключение

В настоящей работе введена *GD-бисимуляция* — авторское расширение классической бисимуляции Парка–Милнера на GD-FST трансдьюсеры с охранными условиями и числовыми весами (определение 6). Предложен алгоритм проверки GD-бисимуляционной эквивалентности (алгоритм 1), построенный на адаптации разбиения Канеллакиса–Смолки [15]. Доказаны корректность и полнота алгоритма (теорема 1) методом индуктивных инвариантов разбиения — стандартной техникой теории алгоритмов эквивалентности [10, 15]. Вычислительная сложность алгоритма в наивной реализации псевдокода составляет $O(k \cdot m \cdot n^2)$; практическая реализация с оптимизацией Пейджа–Тарьяна [17] снижает стоимость цикла уточнения до $O(k \cdot m \cdot \log n)$. Обе оценки полиномиальны по размеру входа.

Работа относится к классу K5 (поведенческая эквивалентность) классификации задач формального анализа игровых механик [11], решая в его рамках задачу *бисимуляционной* эквивалентности для моделей GD-FST; другие отношения эквивалентности, минимизация и композициональность в пределах этого класса остаются открытыми (см. ниже).

Полученные результаты опираются на два допущения — общий словарь переменных сравниваемых механик и точное равенство весов; их ослабление составляет содержание перечисленных ниже открытых задач.

Открытыми остаются:

(*а*) *ε-приближение*: GD-бисимуляция для стохастических механик с близкими (но не равными) вероятностными весами;

(*б*) *временны́е ограничения*: GD-бисимуляция для механик с явным течением времени — переход к временны́м автоматам и timed-бисимуляции, сохраняющей TCTL [22];

(*в*) *минимизация трансдьюсера*: факторизация по отношению $\sim$ даёт лишь *кандидата* на минимальную реализацию и не является непосредственным следствием (см. замечание 9);

(*г*) *межсловарная эквивалентность*: обнаружение эквивалентности механик, заданных над *разными* словарями переменных, требующее явного морфизма сред (замечание 4);

(*д*) *композициональность*: конгруэнтность GD-бисимуляции относительно параллельной композиции механик, требующая предварительной формализации оператора композиции для рёбер с составными метками (замечание 6).

Замечание 9 (Минимизация как открытая задача). Стабилизированное разбиение $\mathcal{P}^*$ (лемма 2) задаёт естественного кандидата на минимальную реализацию – склейку классов GD-бисимулярных вершин. Однако сама факторизация *не* является непосредственным следствием: при склейке у образа вершины могут возникнуть параллельные рёбра с различными метками (cond, w), тогда как определение 2 допускает не более одной метки на пару вершин. Корректное определение фактор-трансдьюсера поэтому требует закона агрегации параллельных переходов – того же закона комбинирования весов, что и в задаче композициональности (пункт (*д*)), – а также отдельного доказательства минимальности результата.

Направление (*а*) сводится к замене точного равенства весов на ε-близость в отношении совместимости рёбер; направления (*б*)–(*д*) требуют расширения формализма – временны́ми ограничениями (*б*), законом агрегации весов при склейке и композиции (3, 5) и отображением сред (*г*).

Алгоритм даёт два непосредственных практических применения:

(А) *верификация рефакторинга* – доказательство того, что переработанная механика поведенчески тождественна исходной;

(Б) *классификация и сравнение библиотек механик* – систематическое сравнение механик из разных игр на эквивалентность в рамках общего словаря переменных, открывающее возможность формального переноса баланса между проектами.

Практическая значимость: алгоритм реализован в платформе Tula [1] и позволяет геймдизайнеру выполнять в интерактивном режиме проверку того, что рефакторинг механики не изменил наблюдаемого поведения. Тот же аппарат применим к любым системам, управляемым правилами с охранными условиями и весами, – сетевым протоколам, рабочим процессам (workflow), мультиагентным системам.

Список использованных источников

[1] Рахманкулова В. Р., Кугуракова В. В. *Онлайн-инструмент Tula для балансировки видеоигр* // Электронные библиотеки.– 2025.– Т. **28**.– № 4.– С. 903–930. ↑164, 167, 185

[2] Milner R. *Communication and Concurrency*.– N.Y.: Prentice Hall.– 1989.– ISBN 978-0-13-114984-7.– 260 pp. ↑164, 165, 168, 169, 174, 175

[3] Park D. *Concurrency and automata on infinite sequences* // *Theoretical Computer Science*, Lecture Notes in Computer Science.– vol. **104**, Berlin–Heidelberg: Springer.– 1981.– ISBN 978-3-540-10576-3.– Pp. 167–183. doi ↑164, 168, 169

[4] Clarke E. M., Emerson E. A., Sistla A. P. *Automatic verification of finite-state concurrent systems using temporal logic specifications* // ACM Transactions on Programming Languages and Systems.– 1986.– Vol. **8**.– No. 2.– Pp. 244–263. doi ↑164

[5] Pnueli A. *The temporal logic of programs* // *18th Annual Symposium on Foundations of Computer Science*, SFCS 1977 (Providence, RI, USA, 31 October 1977–02 November 1977).– IEEE.– 1977.– ISBN 978-3540416951.– Pp. 46–57. doi ↑164

[6] Baier C., Katoen J.-P. *Principles of Model Checking*.– Cambridge: MIT Press.– 2008.– ISBN 978-0262026499.– 975 pp. ↑164, 169, 170, 174, 175, 180

[7] Dormans J. *Engineering Emergence: Applied Theory for Game Design*, Ph.D. thesis.– Amsterdam: Universiteit van Amsterdam.– 2012. ↑165, 167

[8] Jhala R., Majumdar R. *Software model checking* // ACM Computing Surveys.– 2009.– Vol. **41**.– No. 4.– Pp. 21:1–21:54. doi ↑165

[9] van Glabbeek R. J. *The linear time – branching time spectrum I. The semantics of concrete, sequential processes* // *Handbook of Process Algebra*.– Elsevier.– 2001.– ISBN 978-0-444-82830-9.– Pp. 3–99. doi ↑166

[10] Hopcroft J. E. *An $n \log n$ algorithm for minimizing states in a finite automaton* // *Theory of Machines and Computations*, eds. Z. Kohavi, A. Paz, New York: Academic Press.– 1971.– ISBN 978-0-12-417750-5.– Pp. 189–196. doi ↑166, 170, 180, 184

[11] Кугуракова В. В. *Видеоигра как информационная система: формальные основания и классификация задач* // Информационное общество.– 2026.– Т. **4**.– С. 105–122. ↑166, 184

[12] Kugurakova V. V. *Correctness of translation of formal game mechanic models into temporal logics* // Lobachevskii Journal of Mathematics.– 2026.– Vol. **11**.– Pp. 1-18 (to appear). ↑166

[13] Кугуракова В. В. *Граф зависимостей игровых механик как аналитический объект: обнаружение дедлоков и перспективы формального анализа* // Геймификация, игры, креативные индустрии.– 2026.– Т. **1**.– № 1.– С. 46–54. ↑166

[14] Кугуракова В. В. *Формальный подход к пространственно-временному моделированию игровых систем* // Учен. зап. Казан. ун-та. Сер. Физ.-матем. науки.– 2024.– Т. **166**.– № 4.– С. 532–554. doi ↑167, 168

[15] Kanellakis P. C., Smolka S. A. *CCS expressions, finite state processes, and three problems of equivalence* // Information and Computation.– 1990.– Vol. **86**.– No. 1.– Pp. 43–68. doi ↑170, 176, 180, 184

[16] Hopcroft J. E., Motwani R., Ullman J. D. *Introduction to Automata Theory, Languages, and Computation*.– Reading: Addison-Wesley.– 1979.– ISBN 978-0201441246.– 535 pp. ↑170

[17] Paige R., Tarjan R. E. *Three partition refinement algorithms* // SIAM Journal on Computing.– 1987.– Vol. **16**.– No. 6.– Pp. 973–989. ↑170, 178, 184

[18] Cook S. A. *The complexity of theorem-proving procedures* // *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing* (Shaker Heights, Ohio, USA, May 3–5, 1971), New York: ACM.– 1971.– ISBN 978-1-4503-7464-4.– Pp. 151–158. ↑171, 172

[19] De Moura L., Bjørner N. *Z3: An efficient SMT solver* // *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Lecture Notes in Computer Science.– vol. **4963**, Berlin–Heidelberg: Springer.– 2008.– ISBN 978-3-540-78799-0.– Pp. 337–340. ↑171, 172, 179

[20] Papadimitriou C. H. *Computational Complexity*.– Reading: Addison-Wesley.– 1994.– ISBN 9780201530827.– 523 pp. ↑172

[21] Larsen K. G., Skou A. *Bisimulation through probabilistic testing* // Information and Computation.– 1991.– Vol. **94**.– No. 1.– Pp. 1–28. ↑172

[22] Alur R., Dill D. L. *A theory of timed automata* // Theoretical Computer Science.– 1994.– Vol. **126**.– No. 2.– Pp. 183–235. ↑184

Поступила в редакцию 13.07.2026;
одобрена после рецензирования 04.08.2026;
принята к публикации 17.08.2026;
опубликована онлайн 15.09.2026.

Рекомендовал к публикации *д.ф.-м.н. А. М. Елизаров*

Информация об авторе:

Влада Владимировна Кугуракова

Кугуракова Влада Владимировна — кандидат технических наук, доцент, заведующий кафедрой индустрии разработки видеоигр Казанского федерального университета. Область научных интересов: формальная верификация, теория автоматов, игровые системы. Автор более 100 научных публикаций.

0000-0002-1552-4910
e-mail: vlada.kugurakova@gmail.com

Декларация об отсутствии личной заинтересованности: *благополучие автора не зависит от результатов исследования.*

EN RU PROGRAM SYSTEMS: THEORY AND APPLICATIONS ISSN 2079-3316
Research Article THEORETICAL FOUDATIONS OF SOFTWARE SYSTEMS

UDC 004.4'242
10.25209/2079-3316-2026-17-3-163-190

Checking Bisimulation Equivalence of Formal Game-Mechanic Models with Guard Conditions and Weighted Transitions

Vlada Vladimirovna **Kugurakova**
Kazan Federal University, Kazan, Russia

Abstract. This paper addresses the formal equivalence checking of two implementations of the same game mechanic. Such a task arises in game logic refactoring, comparison of independent implementations of one specification, and verification of balancing transformations. The formal representation of such mechanics is based on the GD-FST (Game Design Finite State Transducer) model. The standard verification-theoretic approach of bisimulation is not directly applicable here, since GD-FST edges carry not mere labels but pairs (guard condition, weight), which generates a novel non-trivial transition-matching problem.

The contribution is as follows:

(1) a definition of bisimulation equivalence for GD-FST transducers is introduced, accounting for the edge structure of the formalism;
(2) an algorithm for checking bisimulation equivalence is proposed, constructed by adapting the classical Kanellakis–Smolka partition refinement;
(3) soundness and completeness of the algorithm are proved.

The practical significance of the result lies in enabling automatic verification that refactoring or balancing transformation of a game mechanic does not alter its observable behaviour. The proposed apparatus is applicable not only to game mechanics, but also to other rule-governed systems with guard conditions and numerical weights. (*In Russian*).

Key words and phrases: bisimulation equivalence, GD-FST, game mechanics, formal verification, Kanellakis–Smolka algorithm, Hopcroft partition, temporal logic

2020 *Mathematics Subject Classification*: 68Q85; 68N30, 03B44

For citation: Vlada V. Kugurakova. *Checking Bisimulation Equivalence of Formal Game-Mechanic Models with Guard Conditions and Weighted Transitions*. Program Systems: Theory and Applications, 2026, **17**:3(72), pp. 163–190. (*In Russ.*). https://psta.psiras.ru/read/psta2026_3_163-190.pdf

References

[1] V. R. Raxmankulova, V. V. Kugurakova. "Onlajn-instrument Tula dlya balansirovki videoigr", *Elektronnye biblioteki*, **28**:4 (2025), pp. 903–930.

[2] R. Milner. *Communication and Concurrency*, Prentice Hall, N.Y., 1989, ISBN 978-0-13-114984-7, 260 pp.

[3] D. Park. "Concurrency and automata on infinite sequences", *Theoretical Computer Science*, Lecture Notes in Computer Science, vol. **104**, Springer, Berlin–Heidelberg, 1981, ISBN 978-3-540-10576-3, pp. 167–183.

[4] E. M. Clarke, E. A. Emerson, A. P. Sistla. "Automatic verification of finite-state concurrent systems using temporal logic specifications", *ACM Transactions on Programming Languages and Systems*, **8**:2 (1986), pp. 244–263.

[5] A. Pnueli. "The temporal logic of programs", *18th Annual Symposium on Foundations of Computer Science*, SFCS 1977 (Providence, RI, USA, 31 October 1977–02 November 1977), IEEE, 1977, ISBN 978-3540416951, pp. 46–57.

[6] C. Baier, J.-P. Katoen. *Principles of Model Checking*, MIT Press, Cambridge, 2008, ISBN 978-0262026499, 975 pp.

[7] J. Dormans. *Engineering Emergence: Applied Theory for Game Design*, Ph.D. thesis, Universiteit van Amsterdam, Amsterdam, 2012.

[8] R. Jhala, R. Majumdar. "Software model checking", *ACM Computing Surveys*, **41**:4 (2009), pp. 21:1–21:54.

[9] R. J. van Glabbeek. "The linear time – branching time spectrum I. The semantics of concrete, sequential processes", *Handbook of Process Algebra*, Elsevier, 2001, ISBN 978-0-444-82830-9, pp. 3–99.

[10] J. E. Hopcroft. "An $n \log n$ algorithm for minimizing states in a finite automaton", *Theory of Machines and Computations*, eds. Z. Kohavi, A. Paz, Academic Press, New York, 1971, ISBN 978-0-12-417750-5, pp. 189–196.

[11] V. V. Kugurakova. "Video game as an information system: formal foundations and problem classification", *Informacionnoe obshhestvo*, **4** (2026), pp. 105–122 (in Russian).

[12] V. V. Kugurakova. "Correctness of translation of formal game mechanic models into temporal logics", *Lobachevskii Journal of Mathematics*, **11** (2026), pp. 1-18 (to appear).

[13] V. V. Kugurakova. "Graf zavisimostej igrovyx mexanik kak analiticheskij ob"ekt: obnaruzhenie dedlokov i perspektivy formal'nogo analiza", *Gejmifikaciya, igry, kreativnye industrii*, **1**:1 (2026), pp. 46–54.

[14] V. V. Kugurakova. "A formal approach to spatio-temporal modelling of game systems", *Uchen. zap. Kazan. un-ta. Ser. Fiz.-matem. nauki*, **166**:4 (2024), pp. 532–554 (in Russian).

[15] P. C. Kanellakis, S. A. Smolka. "CCS expressions, finite state processes, and three problems of equivalence", *Information and Computation*, **86**:1 (1990), pp. 43–68.

[16] J. E. Hopcroft, R. Motwani, J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, 1979, ISBN 978-0201441246, 535 pp.

[17] R. Paige, R. E. Tarjan. "Three partition refinement algorithms", *SIAM Journal on Computing*, **16**:6 (1987), pp. 973–989.

[18] S. A. Cook. "The complexity of theorem-proving procedures", *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing* (Shaker Heights, Ohio, USA, May 3–5, 1971), ACM, New York, 1971, ISBN 978-1-4503-7464-4, pp. 151–158.

[19] Moura L. De, N. Bjørner. "Z3: An efficient SMT solver", *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Lecture Notes in Computer Science, vol. **4963**, Springer, Berlin–Heidelberg, 2008, ISBN 978-3-540-78799-0, pp. 337–340.

[20] C. H. Papadimitriou. *Computational Complexity*, Addison-Wesley, Reading, 1994, ISBN 9780201530827, 523 pp.

[21] K. G. Larsen, A. Skou. "Bisimulation through probabilistic testing", *Information and Computation*, **94**:1 (1991), pp. 1–28.

[22] R. Alur, D. L. Dill. "A theory of timed automata", *Theoretical Computer Science*, **126**:2 (1994), pp. 183–235.