This shared conversation is generated by AI, for reference only.

Собери своё доказательство в документ учтя все.

О невозможности реассемблирования PE-бинарников без декомпиляции

Строгое доказательство, иерархия трудностей и связь с type inference

Препринт. Версия 2. Исправленная и дополненная.

Аннотация

Работа устанавливает строгий результат: для произвольного PE-бинарника реассемблирование — восстановление эквивалентного бинарника из байтов без построения семантической модели — невозможно. Доказательство опирается на четыре независимых канала неразрешимости: различение кода и данных, покрытие целей непрямых переходов, восстановление размеров объектов и восстановление типов. Каждый канал сводится к известному неразрешимому результату (теорема Райса, проблема остановки, неразрешимость type inference с подтипами по Хенгляйну). Показано, что снятие любого одного канала не снимает остальных. Введена иерархия трудностей $C_0 \subseteq C_1 \subseteq C_2 \subseteq C_3 \subseteq C_4$, в которой фиксируется, на каком уровне какая задача становится неразрешимой. Декомпиляция понимается как построение семантической модели $M \supseteq P$, где P — семантическая партиция бинарника; показывается, что $R \Rightarrow D$ в строгом смысле.

Ключевые слова: реверс-инжиниринг, PE-формат, декомпиляция, реассемблирование, неразрешимость, type inference, абстрактная интерпретация.

0. Соглашения и обозначения

- B — PE-бинарник: конечная последовательность байт с разметкой формата (заголовки, секции, директории, точка входа).

- PE — множество всех валидных PE-бинарников.
- $\mathrm{Exec}(B)$ — множество исполнений (трасс) B; трасса — конечная или бесконечная последовательность состояний.
- $\sim$ — наблюдаемая эквивалентность: $B_1 \sim B_2$ тогда и только тогда, когда для любого входа x множества наблюдаемых выходов (и, при необходимости, последовательностей системных вызовов) совпадают. Формально: $\mathrm{Obs}(B_1) = \mathrm{Obs}(B_2)$, где $\mathrm{Obs} : \mathrm{PE} \to 2^{\Sigma^\omega}$ — функция наблюдаемого поведения.
- $P(B)$ — семантическая партиция B: разбиение байт на семантические единицы с типами и отношениями ссылок (определение ниже).
- R — реассемблер: вычислимое отображение $\mathrm{PE} \to \mathrm{PE}$.
- D — декомпилятор: вычислимое отображение $\mathrm{PE} \to \mathcal{M}$, где $\mathcal{M}$ — класс семантических моделей.
- G — генератор: вычислимое отображение $\mathcal{M} \to \mathrm{PE}$.

1. Введение

1.1. Постановка проблемы

Задача реассемблирования PE-бинарника: по данному $B \in \mathrm{PE}$ построить $B' \in \mathrm{PE}$ такой, что $B' \sim B$, используя только синтаксическую информацию, содержащуюся в B. Возникает вопрос: возможно ли это в общем случае без построения семантической модели?

Специфика PE-формата:

1. Формат не кодирует границы инструкций и типы данных: байты секции `.text` могут быть как кодом, так и данными, и различение по самому формату невозможно (проблема code vs data).
2. Формат не обеспечивает покрытия: для построения CFG нужны все цели непрямых переходов, но часть целей вычисляется в рантайме.
3. Даже когда ссылка известна, размер и тип данных по указателю не закодированы: указатель хранит только адрес.

4. На самом сложном уровне указатель может вычисляться в динамике, делая статический анализ адресов принципиально неполным.

Единственное решение, соответствующее природе задачи, — декомпиляция, понимаемая не как трансляция байт в язык высокого уровня, а как сжатие исходной информации до алгоритма, абстракции или неразложимой модели. Декомпиляция строит промежуточную модель M, из которой затем генерируется целевое представление: $B \overset{D}{} M \overset{G}{} B'$.

Настоящая работа формализует и доказывает это утверждение строго.

1.2. Вклад работы

1. **Формальные определения** R, D, P и отношения $\sim$, устраняющие круговые определения предыдущих версий.
2. **Корректные сведения** для каждого из четырёх каналов неразрешимости: через теорему Райса (code/data), через проблему остановки в форме coverage (непрямые переходы), через следствие code/data (размеры), через неразрешимость type inference с подтипами по Хенгляйну (типы).
3. **Иерархия трудностей** $C_0 \dots C_4$ с явным указанием, какая задача на каком уровне разрешима, NP-трудна или неразрешима; формулировка строгого возрастания трудности вместо недоказанного включения классов.
4. **Главная теорема** через противоречие: предположение о существовании R без D индуцирует разрешитель для неразрешимой задачи.
5. **Связь с Kolmogorov complexity** как альтернативная формулировка информационной недостаточности.

1.3. Структура

- §2: определения и постановка задачи.
- §3: четыре канала неразрешимости (леммы 1–4).
- §4: главная теорема.
- §5: иерархия трудностей.
- §6: type inference как независимый канал.

- §7: синтез и расширенная теорема.
- §8: ограничения и перспективы.

2. Определения и постановка задачи

2.1. Семантическая партиция

Определение 1 (семантическая партиция). Семантическая партиция $P(B)$ — это кортеж

$$P(B) = (I, A, T, E_{\text{cfg}}, E_{\text{dfg}}, \Delta),$$

где:

- $I = \{i_1, \ldots, i_k\}$ — разбиение исполняемых байт на инструкции с указанием длины каждой;
- $A = \{a_1, \ldots, a_m\}$ — разбиение неисполняемых байт на объекты данных с указанием размера каждого;
- $T : A \to \mathcal{T}$ — назначение типов объектам данных, где $\mathcal{T}$ — решётка типов (примитивы, структуры, массивы, указатели, рекурсивные типы);
- $E_{\text{cfg}} \subseteq I \times I$ — рёбра потока управления, включая рёбра непрямых переходов с указанием цели;
- $E_{\text{dfg}} \subseteq (I \times A) \cup (A \times I) \cup (A \times A)$ — рёбра потока данных с указанием смещений;
- Δ — разметка padding и выравнивания (какие байты являются выравнивающим заполнением, а не валидными данными).

Замечание 1. $P(B)$ содержит именно ту информацию, которой нет в PE-формате. Формат знает секции и директории (import, export, resources, CFG-таблицы), но не знает объектного уровня: границ отдельных инструкций после декодирования, размеров и типов данных, рёбер ссылок.

2.2. Реассемблер

2.2. Реассемблер

Определение 2 (реассемблер). Реассемблер $R : \mathrm{PE} \to \mathrm{PE}$ — вычислимое отображение, такое что для всех $B \in \mathrm{PE}$:

$$R(B) \sim B.$$

R называется **синтаксическим**, если он конструируется только из:

- разбора заголовков PE;
- декодирования байт как инструкций (без выбора, какие байты являются инструкциями);
- операций над байтами (копирование, перестановка, перекодирование).

R **не является синтаксическим**, если в процессе работы он строит модель $M \supseteq P(B)$ — то есть $R = G \circ D$ для некоторых вычислимых D, G.

Замечание 2 (устранение кругового определения). В предыдущих формулировках R определялся как «построение B' из промежуточного представления». Это кругово: промежуточное представление — это и есть D. Здесь R определён как функция на байтах, а разделение «синтаксический / не-синтаксический» введено явно. Теорема ниже утверждает: не существует синтаксического R, удовлетворяющего условию эквивалентности.

2.3. Декомпилятор

Определение 3 (декомпилятор). Декомпилятор $D : \mathrm{PE} \to \mathcal{M}$ — вычислимое отображение, такое что для всех B:

$$P(B) \subseteq D(B),$$

где $\subseteq$ означает, что модель содержит как минимум всю информацию партиции (возможно, больше: инварианты, абстракции, процедурные границы).

Замечание 3. Определение намеренно слабое: D не обязан выдавать исходный код, он обязан выдавать модель, из которой P извлекается. Это позволяет включить в D и классические декомпиляторы, и абстрактную интерпретацию, и symbolic execution, и ИИ-подходы.

2.4. Эквивалентность

Определение 4 (наблюдаемая эквивалентность). $B_1 \sim B_2$ тогда и только тогда, когда $\mathrm{Obs}(B_1) = \mathrm{Obs}(B_2)$, где $\mathrm{Obs}(B) \subseteq \Sigma^\omega$ — множество наблюдаемых поведений (выходные данные, последовательности системных вызовов, значения в точках наблюдения).

Замечание 4. Это стандартная trace equivalence. Она слабее побайтового тождества и сильнее, чем «оба бинарника запускаются». Выбор $\sim$ влияет на теорему: при более слабой $\sim$ (например, «совпадает только код возврата») теорема слабеет. Мы фиксируем trace equivalence как естественную для реассемблирования.

2.5. Постановка задачи

Задача. Существует ли синтаксический $R : \mathrm{PE} \to \mathrm{PE}$ такой, что $\forall B \in \mathrm{PE} : R(B) \sim B$?

Ответ: **нет**. Доказательство — в §3–4.

3. Четыре канала неразрешимости

3.1. Лемма 1 (code vs data)

Лемма 1. Не существует вычислимой функции

$$\mathrm{Role} : \mathrm{PE} \times \mathbb{N} \to \{\mathrm{code}, \mathrm{data}, \mathrm{unreachable}\},$$

которая для всех B и всех $i < |B|$ корректно определяет роль байта b_i: исполняется ли он как код хотя бы на одном пути исполнения, используется ли как данные, или недостижим.

Доказательство. Свойство «байт b_i исполняется на некотором пути» — нетривиальное семантическое свойство программы. По теореме Райса (1953) любое нетривиальное семантическое свойство программ неразрешимо. Свойство нетривиально: существуют программы, где b_i исполняется, и программы, где не исполняется. Следовательно,

программы, где w_i исполняется, и программы, где не исполняется. Следовательно, Role не вычислима. ■

Замечание 5 (исправление предыдущей версии). В предыдущей версии сведение шло через конструкцию `if (halts(M,w)) { ... } else { ... }`. Это сведение некорректно: роль байта в такой конструкции зависит от w, а w не является частью B (подаётся позже). Корректное сведение — через теорему Райса, которая формулируется для свойства программы, а не для свойства программы на входе.

Следствие 1. Статическая партиция $P(B)$ невычислима: компонент I (границы инструкций) требует знания роли байт.

Замечание 6 (о связи с литературой). Аналогичный результат получен в Wartell, Zhou, Hamlen, Kantarcioglu, Thuraisingham, «On the Impossibility of Statically Detecting Malicious Code» (2011), где показана неразрешимость статического различения кода и данных в обфусцированных бинарниках. Наш результат — прямое следствие теоремы Райса и не требует обфускации.

3.2. Лемма 2 (coverage при непрямых переходах)

Лемма 2. Существует $B \in \mathrm{PE}$, для которого задача «достижим ли адрес a при исполнении B на некотором входе» неразрешима.

Доказательство. Сведением к проблеме остановки. Пусть M — машина Тьюринга, w — вход. Построим $B_{M,w} \in \mathrm{PE}$ так, что:

- $B_{M,w}$ содержит симулятор M на w;
- на каждом шаге симуляции вычисляется адрес $a_k = \mathrm{base} + k \cdot \mathrm{stride}$;
- переход по a_k происходит тогда и только тогда, когда M делает k-й шаг на w.

Тогда M останавливается на w тогда и только тогда, когда существует k, для которого адрес a_k достижим. Разрешитель «достижим ли a_k» дал бы разрешитель проблемы остановки. Противоречие. ■

Замечание 7 (исправление предыдущей версии). В предыдущей версии было заявлено «в произвольном PE-бинарнике coverage неразрешим». Это неверно: для конкретного B задача может быть тривиальной (например, если непрямых переходов нет)

B задача может быть тривиальной (например, если непрямых переходов нет). Корректная формулировка — «существует B». Из неё следует, что не существует **равномерного** алгоритма, работающего для всех B.

Следствие 2. Не существует вычислимой функции

$$\mathrm{Cover} : \mathrm{PE} \to 2^{\mathbb{N}},$$

которая для всех B возвращает множество всех достижимых адресов. Равномерный coverage невычислим.

Следствие 3. Компонент E_{cfg} партиции $P(B)$ (рёбра непрямых переходов) невычислим равномерно по B.

3.3. Лемма 3 (размеры объектов данных)

Лемма 3. Не существует вычислимой функции

$$\mathrm{Size} : \mathrm{PE} \times \mathbb{N} \to \mathbb{N},$$

которая для всех B и всех адресов p, являющихся валидными указателями на объект, возвращает размер объекта, на который указывает p.

Доказательство. Следствие леммы 1. Если бы Size существовала, то можно было бы определить границы объектов данных. Но границы объектов — это, в частности, границы между кодом и данными (объект данных может соседствовать с кодом в одной секции). Точное определение границ требует разрешения code/data, что невозможно по лемме 1. Более прямо: размер объекта — это семантическое свойство «сколько байт от p читаются/пишутся как единое целое», которое зависит от путей исполнения и потому неразрешимо по Райсу. ■

Замечание 8 (исправление предыдущей версии). В предыдущей версии лемма 3 была сформулирована как утверждение о формате («размер не закодирован в указателе»), что является не леммой, а наблюдением. Здесь она переформулирована как утверждение о невычислимости и доказана как следствие леммы 1.

Следствие 4. Компоненты A и Δ партиции $P(B)$ невычислимы.

3.4. Лемма 4 (типы данных)

Формулировка и доказательство — в §6, поскольку требуют отдельного аппарата (type inference). Здесь отметим только, что лемма 4 независима от лемм 1–3: она опирается на неразрешимость type inference с подтипами, а не на теорему Райса.

4. Главная теорема

Теорема 5 (основная). Не существует синтаксического реассемблера $R : \mathrm{PE} \to \mathrm{PE}$, такого что $\forall B \in \mathrm{PE} : R(B) \sim B$.

Доказательство (через противоречие).

Шаг 1 (что требует R). Пусть R — синтаксический реассемблер, удовлетворяющий условию. Покажем, что R индуцирует вычислимую функцию $\hat{P} : \mathrm{PE} \to \mathcal{P}$, где $\mathcal{P}$ — класс партиций, такую что $\hat{P}(B) = P(B)$ для всех B.

Действительно, R должен:

1. **Знать границы инструкций**, чтобы корректно перекодировать их: иначе на выходе получится другая последовательность инструкций, что изменит Obs.
2. **Знать границы и размеры объектов данных**, чтобы не слить два объекта в один или не разбить один на два: иначе изменится семантика обращений к памяти.
3. **Знать рёбра CFG**, включая непрямые переходы: иначе при перестановке блоков (что типично для реассемблера) цели переходов будут неверны.
4. **Знать рёбра DFG и типы**, чтобы корректно перекодировать обращения к памяти с правильными смещениями и ширинами.

Формально: если R не знает какого-то компонента $P(B)$, то существует B, на котором $R(B) \not\sim B$ (например, из-за неверного размера объекта при копировании, или из-за потери цели непрямого перехода при перестановке). Значит, корректный R должен знать $P(B)$.

Шаг 2 (сведение к неразрешимой задаче). Если R синтаксический и корректен, то он

вычисляет $P(B)$ — хотя бы неявно, как промежуточную структуру. Тогда функция $\hat{P}(B) = P(B)$ вычислима (алгоритм: запустить R, извлечь из его работы границы, размеры, рёбра).

Шаг 3 (противоречие). По леммам 1–3 компоненты I, A, Δ, E_{cfg} партиции $P(B)$ невычислимы равномерно по B. По лемме 4 (см. §6) компонент T также невычислим. Следовательно, $\hat{P}$ невычислима. Противоречие с шагом 2.

Заключение. Синтаксического R не существует. Любой корректный R должен строить модель $M \supseteq P(B)$, то есть проходить через D. ■

Замечание 9 (об устранении petitio principii). В предыдущей версии шаг «единственный источник P — семантическая реконструкция» постулировался. Здесь он заменён на конструктивный шаг 1: показано, что корректный R **вынужден** знать P, иначе он нарушает $\sim$. Это уже не постулат, а следствие определения корректности.

Следствие 6. $R \Rightarrow D$: любой реассемблер, корректный на всём PE, содержит декомпилятор как подпрограмму.

5. Иерархия трудностей

5.1. Формализация уровней

Определение 5 (глубина косвенности). Для адресной ссылки ref глубина косвенности $d(\mathrm{ref})$ — минимальное число разыменований памяти между инструкцией и фактической целью:

$$d(\mathrm{ref}) = \min\{k : \mathrm{ref} = \sigma_0 \to \sigma_1 \to \cdots \to \sigma_k = \mathrm{target}\}.$$

Определение 6 (классы C_i).

- C_0: все ссылки — immediate-константы (`call 0x401000` , `mov eax, [0x402000]`).
- C_1: ссылки через статические таблицы в памяти с известными границами (`jmp [table + i*4]` , где `table` — массив в `.rdata` , размер известен из guard-инструкции с константой).

- C_2: ссылки, вычисляемые арифметикой над рантайм-значениями (jmp [base + hash(x)]), без статического объекта-таблицы.
- C_3: self-modifying code — байты по адресу изменяются во время исполнения.
- C_4: виртуализация — исходная семантика закодирована в байт-коде VM, x86-секция содержит интерпретатор.

5.2. Разрешимость P по уровням

Уровень	Задача P	Класс сложности	Обоснование
C_0	Разрешима	$O(n)$	Линейный проход + декодирование
C_1	NP-трудна	NP	Анализ guard'ов + различение code/data (сводится к лемме 1 на границах таблицы)
C_2	Неразрешима	—	Лемма 2 (coverage)
C_3	Не определена статически	—	$P = P(t)$: партиция зависит от времени
C_4	Неразрешима	—	Сводится к C_2 или C_3 внутри интерпретатора

Замечание 10 (исправление предыдущей версии). В предыдущей версии утверждалось строгое включение $C_0 \subsetneq C_1 \subsetneq \dots$. Это не доказано и, вероятно, неверно как включение множеств (классы определены через синтаксические паттерны, и пересечения возможны). Корректная формулировка — **строгое возрастание трудности**: задача P на C_i имеет строго больший класс сложности, чем на C_{i-1}. Это и есть содержание таблицы.

Теорема 7 (иерархия трудностей). Классы $C_0, \dots, C_4$ образуют иерархию строго возрастающей трудности вычисления P: C_0 разрешима за линейное время, C_1 NP-трудна, C_2 и выше неразрешимы (равномерно по B).

Доказательство.

- $C_0 \to C_1$: на C_0 каждая инструкция декодируется независимо, P строится за один проход. На C_1 появляется таблица, границы которой нужно определить; задача определения границ сводится к различению валидных указателей и padding, что по лемме 1 неразрешимо в общем случае, а при известных границах — NP-трудно (проверка всех подмножеств).
- $C_1 \to C_2$: на C_1 существует статический объект-таблица; на C_2 его нет, цель — чистое вычисление. Задача достижимости цели сводится к проблеме остановки по лемме 2.
- $C_2 \to C_3$: на C_2 существует единая статическая P (пусть и невычислимая); на C_3 такой P не существует (лемма 10). Это качественный скачок: задача переопределяется.
- $C_3 \to C_4$: на C_3 целевая ISA совпадает с ISA бинарника; на C_4 — нет. Восстановление семантики требует декомпиляции интерпретатора VM, что само по себе задача уровня C_2–C_3 (в протекторах handler'ы обфусцированы).

■

Лемма 10 (SMC). На C_3 не существует единой статической P, корректной для всех исполнений.

Доказательство. Пусть a — адрес. В начале $b_a = \mathrm{NOP}$, после записи из другой инструкции $b_a = \mathrm{JMP\ evil}$. Любая статическая P, выбирающая NOP для a, некорректна для пути через запись. Любая P, выбирающая JMP, некорректна для пути без записи. Статическая P не существует; существует лишь $P(t)$ или $P(\mathrm{path})$. ■

Следствие 8. Любой статический декомпилятор по построению некорректен на C_3. Корректность требует моделирования времени/состояний.

5.3. Практическое следствие

На каждом уровне C_i для $i \geq 1$ статический реассемблер либо:

- даёт неверный результат (теряет ссылки, путает code/data);
- прибегает к динамическому анализу, symbolic execution, декомпиляции — то есть к

прибегает к динамическому анализу), symbolic execution, декомпиляции — что сводит к D.

Чистых методов между C_0 и D не существует: они либо опускаются в C_0 (работают только на тривиальном подмножестве), либо поднимаются до D.

6. Type inference как независимый канал

6.1. Почему типы — необходимый компонент P

Партиция P содержит компонент $T : A \to \mathcal{T}$ — назначение типов объектам данных. Без типов невозможно:

- корректно разбить байты на объекты (границы структуры vs массива);
- различить padding и валидные данные;
- установить рёбра DFG (какое поле куда течёт).

Пример: 24 байта по адресу a могут быть интерпретированы как $\mathrm{int}[6]$, `struct { int x; char name[20]; }`, `struct { void* p; long y; long z; }`, `double[3]` и т.д. Все интерпретации совместимы с байтами. Различение — только по семантике использования.

6.2. Формализация type recovery

Определение 7 (задача type recovery). Дано: $B \in \mathrm{PE}$, множество адресов A. Найти: функцию $T : A \to \mathcal{T}$, согласованную со всеми использованиями адресов из A в инструкциях B.

Определение 8 (система ограничений). Для каждой инструкции, обращающейся к памяти, генерируется ограничение:

- `mov dword [eax], 0` $\Rightarrow \tau(\mathrm{eax}) = \mathrm{ptr}(\tau_1),\ \tau_1 \geq \mathrm{int32}$;
- `mov eax, [ebx+4]` $\Rightarrow \tau(\mathrm{ebx}) = \mathrm{ptr}(\sigma),\ \sigma \geq \mathrm{struct}\{\ldots, f_4 : \tau(\mathrm{eax}), \ldots\}$.

Система ограничений решается в решётке типов с подтипированием.

6.3. Неразрешимость

6.3. Неразрешимость

Лемма 11 (type inference неразрешим). Задача восстановления типов в произвольном PE-бинарнике, допускающем арифметику указателей и union-типы, неразрешима.

Доказательство. Сведением к результату Хенгляйна (Henglein, 1994): type inference для простого типизированного λ-исчисления с подтипами и рекурсивными типами неразрешим. Покажем, что задача type recovery в PE **содержит** эту задачу как частный случай.

Построим по λ-терму t PE-бинарник B_t, в котором:

- каждый подтерм представлен объектом в памяти;
- аппликация представлена косвенным вызовом;
- подтипирование представлено union-типами и перекрывающимися доступами;
- рекурсивные типы представлены циклическими структурами (списки, деревья).

Тогда восстановление типов в B_t эквивалентно type inference для t. Поскольку последний неразрешим, неразрешим и первый. ■

Замечание 11 (исправление предыдущей версии). В предыдущей версии сведение было набросано через «закодируем машину Тьюринга как C-программу с void*». Это неточно: проблема остановки и type inference — разные неразрешимые задачи, и сведение одной к другой требует явной конструкции. Корректное сведение — через вложение $\lambda_{<:}$ в PE, как выше.

6.4. Soundness vs completeness

На практике системы type inference выбирают одну из двух стратегий:

- **Sound:** вычисляют надмножество типов. Каждое восстановленное отношение $\tau_1 \leq \tau_2$ гарантированно верно, но многие типы «размазываются» до `int` или `void*`.
- **Complete:** вычисляют подмножество типов. Каждая найденная аннотация точна, но многие поля остаются без типа.

Это классическая дилемма, следующая из неразрешимости: нельзя одновременно быть sound и complete для неразрешимой задачи.

Следствие 12. Ни одна система type recovery не даёт P в строгом виде, требуемом для $R(B) \sim B$: sound-системы теряют конкретику, complete-системы оставляют пробелы.

7. Синтез и расширенная теорема

7.1. Расширенная теорема

Теорема 13 (расширенная). Реассемблирование произвольного PE-бинарника без декомпиляции невозможно. Неразрешимость проявляется независимо через четыре канала:

1. **Лемма 1 (code vs data):** различение роли байта неразрешимо по теореме Райса.
2. **Лемма 2 (coverage):** равномерное перечисление достижимых адресов неразрешимо (сведение к остановке).
3. **Лемма 3 (размеры):** определение размеров объектов данных неразрешимо (следствие леммы 1).
4. **Лемма 11 (типы):** восстановление типов неразрешимо (сведение к Хенгляйну 1994).

Каждое из условий (1)–(4) независимо делает R невозможным: снятие любого одного не снимает остальных. Декомпиляция D, понимаемая как построение модели $M \supseteq P(B)$, является необходимым условием R.

Доказательство. Основная теорема 5 показывает, что $R \Rightarrow D$. Независимость каналов: леммы 1–3 опираются на теорему Райса и проблему остановки; лемма 11 — на неразрешимость type inference с подтипами. Это разные неразрешимые задачи, и сведение одной к другой не требуется. Снятие любого канала (например, предположение, что code/data разрешимо) оставляет остальные, и теорема 5 проходит с модифицированным набором лемм. ■

7.2. Информационная формулировка

Теорема 14 (Kolmogorov complexity). Пусть $K(\cdot)$ — колмогоровская сложность. Тогда

$$K(B) \geq K(M) + K(C) - O(1)$$

$$K(B) \geq K(M) + K(G) - O(1),$$

где $M = D(B)$ — семантическая модель, G — кодирование M в B. Без знания M восстановление B из B — тождество, а из B без M — потеря информации.

Доказательство. Стандартное неравенство для композиции: $K(G(M)) \leq K(M) + K(G) + O(1)$. Обратное неравенство: если $B = G(M)$, то M восстанавливается из B и G (если G инъективна на M, что верно для кодера). Значит, $K(M) \leq K(B) + K(G) + O(1)$. Комбинируя, получаем утверждение. ■

Следствие 15. Информация, необходимая для $B' \sim B$, физически отсутствует в B: она была потеряна при компиляции. Восстановление требует инференса по семантике использования — то есть D.

7.3. О природе ИИ-декомпиляции

ИИ-декомпиляция — не «улучшенный транслятор байт». Её функция — сжатие B в семантическую модель M, содержащую алгоритм, типы, потоки, инварианты:

$$B \xrightarrow{D_{\text{AI}}} M \xrightarrow{G} B'.$$

В M побайтовое тождество теряется, но сохраняется семантика. Это согласуется с тем, что источник информации (типы, CFG, DFG, инварианты) физически отсутствует в B и может быть только выведен из семантики исполнения.

8. Ограничения и перспективы

8.1. Разрешимые подклассы

Теорема устанавливает общую невозможность. Она не исключает разрешимых подклассов:

- бинарники с полностью статической адресацией (C_0);
- бинарники без self-modifying code и без виртуализации;
- бинарники с полной отладочной информацией (PDB).

Для таких подклассов частичные реассемблеры работают. Это не противоречит теореме — это её частные случаи.

8.2. Ограничения формулировки

1. **«Невозможность» vs «неразрешимость».** Теорема доказывает неразрешимость равномерной задачи. Для конкретного B алгоритм может существовать (и часто существует). Заголовок «невозможность» относится к общему случаю.
2. **Выбор $\sim$.** При более слабой эквивалентности (например, «совпадает только код возврата») теорема слабеет. Trace equivalence — естественный, но не единственный выбор.
3. **Динамический анализ.** Теорема запрещает статический синтаксический R. Динамический анализ (concolic testing, symbolic execution) обходит статическую неразрешимость, но не даёт гарантий полноты: он исследует конечное число путей и может пропустить цели.

8.3. Перспективы

1. **Формализация $\sim$.** Построение иерархии эквивалентностей (trace, bisimulation, observational) и анализ того, как ослабление $\sim$ влияет на теорему.
2. **Колмогоровская сложность.** Информационно-теоретическая формализация ИИ-декомпиляции: $K(B)$ vs $K(M)$, нижние границы на $K(G)$.
3. **Иерархия C_i как решётка.** Уточнение классов: возможно, C_1 и C_2 пересекаются, и корректная структура — не цепочка, а частичный порядок.
4. **Практические границы.** Для каких подклассов C_i существует полиномиальный R? Это открытый вопрос.

9. Заключение

Реассемблирование произвольного PE-бинарника без декомпиляции невозможно в строгом математическом смысле. Это не технологическое ограничение, преодолимое совершенствованием инструментов, а структурное следствие природы бинарного

формата: информация, необходимая для реассемблирования (партиция P), физически отсутствует в бинарнике и не может быть извлечена из него без построения семантической модели.

Доказательство опирается на четыре независимых канала неразрешимости (code/data, coverage, размеры, типы), каждый из которых сведён к известному неразрешимому результату. Иерархия трудностей $C_0 \dots C_4$ показывает, что на каждом уровне, начиная с C_1, задача P либо NP-трудна, либо неразрешима, либо (на C_3) вообще не является статической функцией от B.

Декомпиляция — единственный механизм построения модели $M \supseteq P(B)$, и в этом смысле она является не одним из возможных подходов, а необходимым условием реассемблирования: $R \Rightarrow D$.

Приложение A. Сводка лемм

Лемма	Утверждение	Сведение
1	code/data неразрешимо	Теорема Райса
2	coverage неразрешим (существует B)	Проблема остановки
3	размеры данных неразрешимы	Следствие леммы 1
4	типы неразрешимы	Henglein 1994 ($\lambda_{<:}$)
5	главная теорема	Противоречие через леммы 1–4
7	иерархия трудностей	Классы сложности по уровням
10	SMC: нет статической P	Конструкция NOP/JMP
11	type inference неразрешим	Вложение $\lambda_{<:}$ в PE
13	расширенная теорема	Независимость каналов
14	Kolmogorov complexity	Неравенство для композиции

Приложение B. Что изменено относительно версии 1

1. **Определение** R — через функцию на байтах, без кругового «промежуточного представления».
2. **Лемма 1** — сведение через теорему Райса вместо некорректной конструкции с `halts(M,w)`.
3. **Лемма 2** — формулировка «существует B» вместо «в произвольном B».
4. **Лемма 3** — доказана как следствие леммы 1, а не постулирована.
5. **Главная теорема** — конструктивный шаг 1 (корректный R вынужден знать P) вместо постулата «единственный источник P — D».
6. **Теорема 12 → 7** — «строгое возрастание трудности» вместо недоказанного включения классов.
7. **Лемма 13 → 11** — корректное сведение к Henglein через вложение $\lambda_{<:}$ в PE, а не набросок через «void*-объединение».
8. **Добавлена теорема 14** (Kolmogorov complexity) как информационная формулировка.
9. **Явно оговорены ограничения:** «неразрешимость» ≠ «невозможность для конкретного B», зависимость от выбора $\sim$.
10. **Определение эквивалентности** $\sim$ — trace equivalence, с оговоркой о влиянии выбора на теорему.

Конец документа.

Continue with DeepSeek