

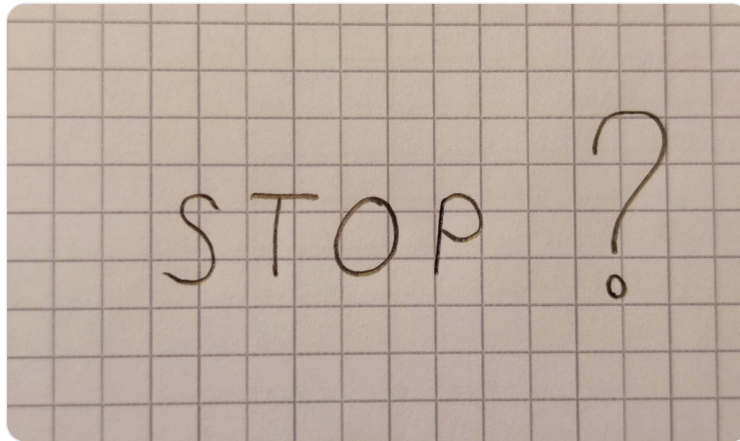


Валерий Жила @ValeriiZhyla

Nov 17, 2021 · 78 tweets · [ValeriiZhyla/status/1460897009842495491](https://twitter.com/ValeriiZhyla/status/1460897009842495491)

Сегодня я расскажу про одну очень интересную и невероятно полезную штуку из Computer Science. Почему наши программы зависают? Почему многие баги нельзя обнаружить заранее?

Встречайте – Проблема Остановки!



Проблема Остановки, она же Задача Остановки, она же просто Halting Problem.

Я расскажу её формулировку, немножко повосхищаюсь Куртом Гёделем и скажу пару слов его теореме о неполноте и свяжу её с Проблемой Остановки.

Далее я шаг за шагом разберу упрощенную версию доказательства неразрешимости Проблемы Остановки, и порассуждаю о том, что это знание нам дает.

К слову, доподлинно известно, что ретвиты моих тредов благоприятно влияют на карму, исцеляют раны бессмертной души и способствуют развитию чувства прекрасного у общества.

Итак, Проблема Остановки. Её суть довольно проста:

У нас в наличии программа и её входные данные (если они вообще нужны).

Задача состоит в том, чтоб определить, завершится ли её работа за конечное время, или же программа будет работать вечно, так никогда и не остановившись.

Немного перефразирую: Проблему Остановки можно решить, если написать алгоритм, который получает исходный код программы и какие-то данные для нее, и выносит вердикт, придет ли выполнение этого кода к какому-то завершению.

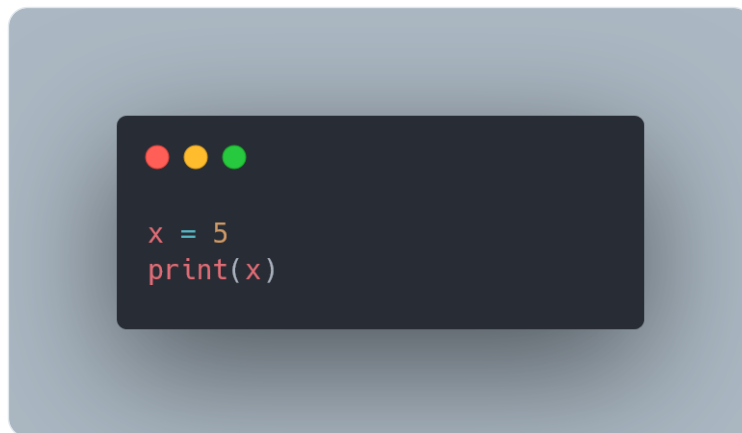
Звучит не то чтобы и очень сложно, правда?

На самом деле, эта проблема довольно прозаична, и никоим образом не надругается над психикой зрителя.

Никаких тебе недетерминизмов, классов сложности, автоматов - никаких классических "приколов" из Computer Science в ней нет.

Рассмотрим несколько примеров, чтоб получше понять эту проблему.  
И понять, почему это не просто проблема, а Проблема!

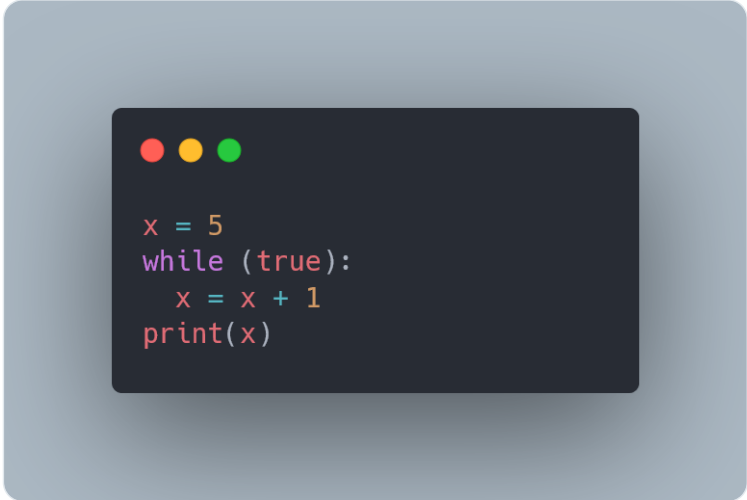
Для полноты картины продемонстрирую сначала элементарный код, который гарантированно "остановится".



Число записывается в переменную, содержимое переменной печатается в консоль.

Два действия, всё просто, прямым с первого дня курсов по программированию для альтернативно одаренных!

Теперь взглянем на очень простой кусочек кода, который гарантированно будет работать в бесконечном цикле.



```
x = 5
while (true):
    x = x + 1
    print(x)
```

Число записывается в переменную, потом бесконечно долго увеличивается на единицу, а потом содержимое этой переменной печатается в консоль.

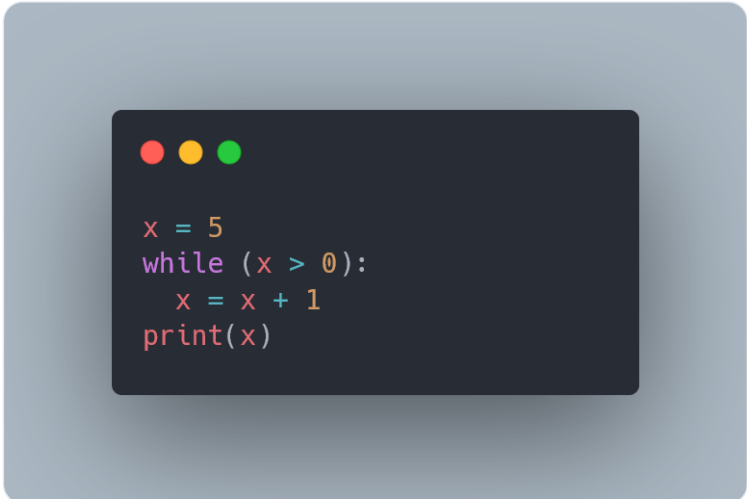
Настолько банальный случай наши IDE распознают, согласен. Среда разработки нам подсветит `print(x)` красным, желтым, зеленым, каким угодно цветом, и подскажет, что этот код "недостижим".

Так что, наши IDE, анализаторы, компиляторы и все остальные волшебные животные стали настолько умными, что решают Проблему Остановки? Вот это новости! Дед, а нафига ты вообще об этом пишешь тогда?!

К сожалению, нет.

В данном случае, это реакция на совсем банальный, однозначный и клишированный `while(true)`. Костыль, хардкод, мелочь (приятная, не спору, иногда помогает)

Давайте чуть-чуть, буквально на крошку усложним наш последний пример.



```
x = 5
while (x > 0):
    x = x + 1
    print(x)
```

`while(true)` мы заменили на `while(x > 0)`. Начинаем мы с пятерки (а она, если мой мозг ещё не совсем вытек, всё-таки больше нуля), и увеличиваем наш гордый `x` раз за разом.  
Цикл всё еще выполняется "бесконечно" - `x` никогда не упадет до нуля и ниже.

(мы забудем про переполнения переменных и скажем, что `x` будет увеличиваться до бесконечности, для простоты примера)

Ууаля - предупреждение о недостижимости `print(x)` пропало. Среда разработки даже не пытается проверить такой цикл на наличие конца.

Какая глупая среда, видно же с первого взгляда!

Она не глупа, о нет, среда эта очень-очень умная. Просто её создатели знали, что это гиблое дело.

Нужно заметить, что Проблема Остановки связана не только с бесконечными циклами.

Любая рекурсия, некоторые вычисления, и очень много других проблем могут заставить нашу программу работать бесконечно.

Даже обнаружение дедлоков в широком смысле связано с Проблемой Остановки.

То, что проблема остановки является "semi-decidable", или же "полу-разрешима", доказал отец всея Computer Science ещё в 1936 году.

Звали этого героя Алан Тьюринг. Я так что его упоминаю, пожалуй, имеет смысл приложить его фото. Ловите:



Что значит эта полуразрешимость? Всё предельно просто. Дед только выпьет свои таблетки и обо всем расскажет!

Сделаем простой алгоритм проверки - он будет запускать программу и просто ждать ее остановки.

Если задача имеет положительное решение (программа из первого примера, которая гарантированно останавливается), то алгоритм проверки говорит: "Друзья, все отлично, она останавливается".

А если задача положительного решения не имеет (программа из третьего примера, который гарантированно не останавливается), то алгоритм проверки ждет окончания её работы вечно - и никакого вердикта мы от него не получим.

Если я правильно перевел, в русском языке полуразрешимость правильно называть "рекурсивной перечислимостью".

В данном контексте мне совсем не нравится это название, если честно. Сойдемся на полуразрешимости, ладно? Одно и то же, нечего голову лишний раз ломать.

Предлагаю немножко отвлечься от Проблемы Остановки. Мы к ней очень скоро вернемся с новым оружием, и займемся основательно!

Я сказал, что Алан Тьюринг считается отцом номер один всей Computer Science.

Отцов у CS было много, но рука об руку с Аланом идет ещё один красавец, родившийся за 6 лет до него и пережившем аж на 24 года.

Звали этого красавца Курт Гёдель. Узнаем своих героев в лицо!



Почему я вдруг о нём вспомнил?

Курт вошел в историю, доказав, что любая формальная система с набором правил, которые не создают противоречий (например, арифметика из младших классов), не позволяет из этого набора правил вывести утверждение о непротиворечивости этой системы.

Это называется Теоремой Гёделя о Неполноте. Точнее, это вторая из двух его теорем о неполноте.

Предполагаю, что на последних абзацах кто-то занервничал. Всё хорошо. Всё будет хорошо. По крайней мере у нас, а герои нашего рассказа уже давно умерли, им точно уже не до нервов.

Что значит вся эта дребедень? Если короче - если (сколько-нибудь осмысленная) система описана правилами, она может себе самой не противоречить, но будет содержать недоказуемые высказывания.

Практически все системы взламываются универсальной отмычкой, именуемая автореференцией или самореференцией.

Что значит это страшное слово? Это "ссылка чего-то на самого себя". Самореферентными называют логические высказывания, содержащие информацию о самом себе.

Самым популярным примером самореференции является Парадокс Лжеца, в разных его формах.

Вот самая простая его форма: высказывание "Данное утверждение ложно".

Если предположить, что оно правдиво, то оно ложно, так как прямым текстом сообщает нам об этом.

И наоборот, если предположить, что оно ложно, то оно сообщает нам правду - ведь действительно, оно же само говорит нам о том, что ложно. Парадокс чистой воды.

Зачем я свернул с Проблемы Остановки в эти дебри?

Дело в том, что я хочу показать доказательство неразрешимости Проблемы Остановки, а в нём как раз используется автореференция, похожий парадокс и вообще Гёдель приложил к нему руку.

Последний случайный факт о нём с Википедии, просто чтоб сложить представление об этом человеке (я им восхищаюсь и вообще это мой краш):

"В 1948 году Гёдель получил американское гражданство. На собеседовании он попытался доказать, что Конституция США формально-логически неполна и не гарантирует от установления диктатуры, но был вежливо остановлен."

Итак, доказательство неразрешимости Проблемы Остановки.

Доказательство работает от противного - мы принимаем за истину то, что хотим опровергнуть, и сводим всё к логическому противоречию.

Доказательство само по себе немного противное, особенно когда видишь подобное в первый раз.

Самореференция заставляет мозги трещать, иногда. Но я постараюсь сгладить острые углы, правда-правда!

Доказательство будет состоять из начального допущения и трех шагов.

Шаг номер 0. Принимаем то, что проблема остановки разрешима.

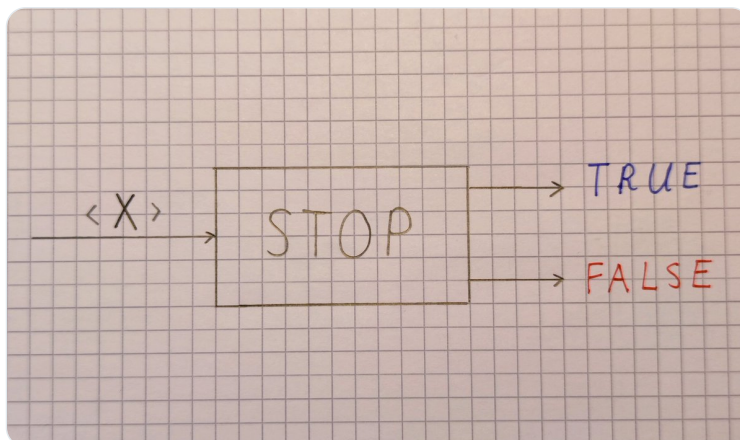
Шаг номер 1. Из шага 0 следует, что существует некая программа, которую мы могли бы назвать `willThisProgramStop`, но назовем её коротко `Stop`, потому что это слово нам потребуется много раз.

`Stop` которая принимает любую программу в качестве вводных данных. Например, для абстрактной программы под названием `X`, `Stop(X)` выдает `True`, если `X` останавливается (как в первом примере без цикла), и `False`, если программа не останавливается (как в примере с бесконечным циклом).

В каком виде передается `X`? Можно это полноценно представить в виде исходного кода программы `X`, и никто на нас не обидится.

Благодаря нашему допущению из пункта 0. мы знаем, что `Stop(X)` выдаст решение за конечное время для любой программы `X`.

Наверное, настало время рисунков. Вот так можно изобразить программу `STOP`. `<X>` значит "исходный код программы `X`".





Держитесь пока? Не сложно? Может, вам капелек каких, от нервов и для храбрости?

Нет? Я пожалуй соблазнюсь, как знаете, как знаете.

Рветесь в бой? До жути интересно, чем закончится эта сага?

Что ж, тогда к делу!

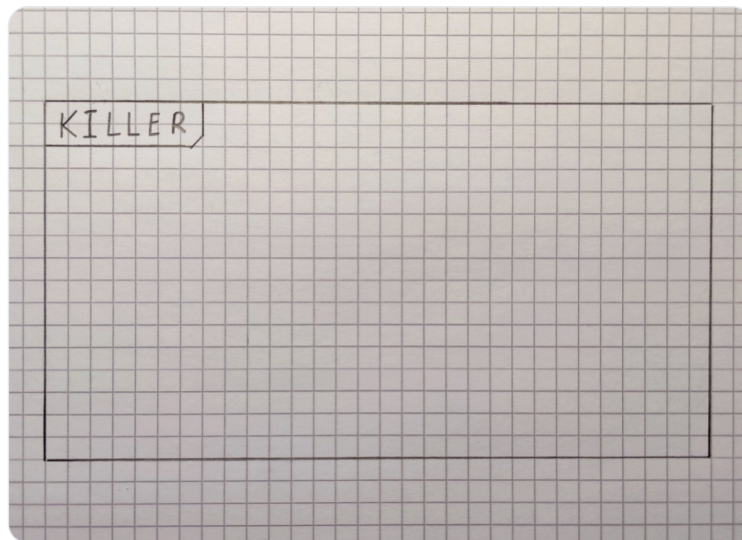
Повторю еще раз, для протокола: Stop принимает исходный код любой программы и за конечное время гарантированно выдает True или False. Важно заметить, что Stop является конкретной программой с конкретным алгоритмом, а значит тоже имеет собственный исходный код.

Шаг номер 2. Сконструируем весельчака прямиком из мира загадок и парадоксов, всё как Гёдель завещал.

Дадим ему броское имя. Имя...назовем его Killer.

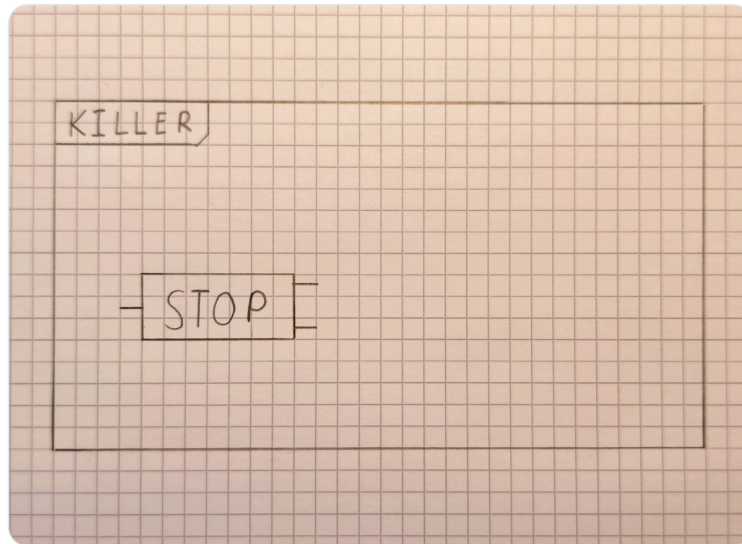
Почему бы и нет? Я не хочу алфавитного онанизма больше со всякими абстрактными X, Y, Z и так далее. Тошно. Только всех путает!

Во-первых, нарисует киллеру каркас. Он пригодится, поверьте!



Что нам нужно знать про программу Killer? Во-первых, она содержит в себе программу Stop, и может её за счет этого "симулировать", с любыми данными, с которыми захочет.

Добавим нашего старого друга Stop в рисунок!

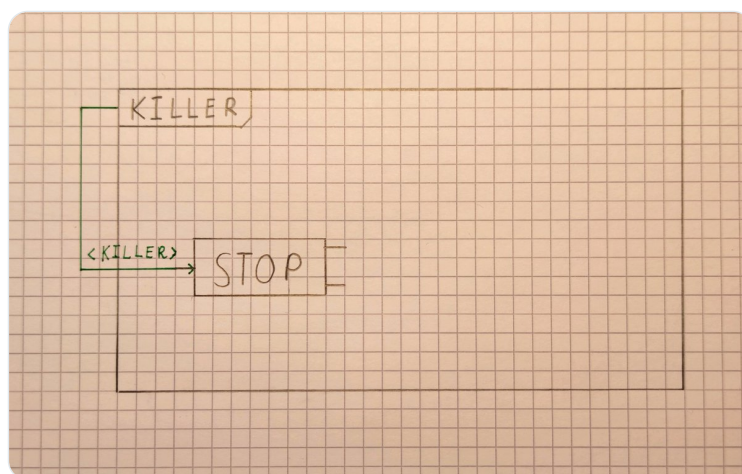


Как и Stop, Killer является конкретной программой с конкретным исходным кодом.

Что делает Killer?

Для начала он берет свой исходный код (который содержит всю логику Killier, включая программу Stop) и запускает Stop с этим исходным кодом.

Тобишь он выполняет Stop(Killer). Рисуем это и попытаемся осознать, это самое сложное место!



В каком-то смысле, мы получаем нехитрую рекурсию.

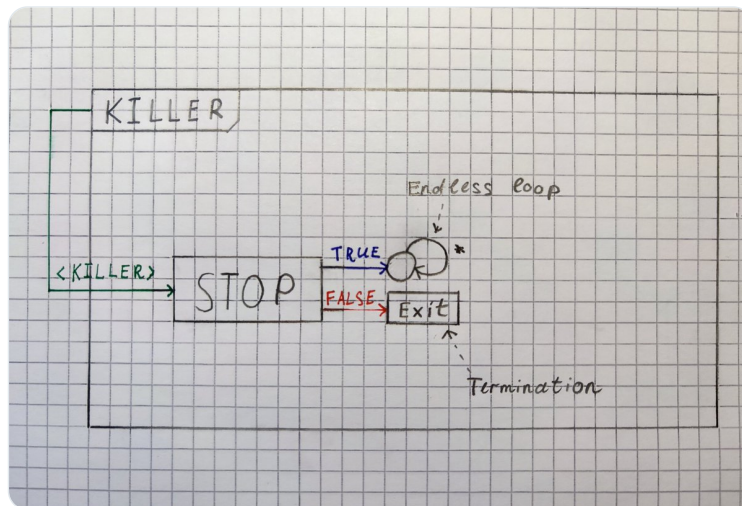
Может быть, даже в самом прямом смысле, я не уверен в этой тонкости русского языка. Ведь Stop(Killer) будет запускать Killer, а он следующий Stop(Killer) внутри себя и так далее. Но это еще не все!

Если Stop(Killer) выдаст False (в случае если "внутренний" Killer не останавливается, как мы условились в шаге 1), то "внешний" Killer прекращает работу, трубит в рог, летят фейверки и бегут чирлидерши.

Если же Stop(Killer) выдаст True ("внутренний" Killer успешно отработал и остановился), то "внешний" Killer грустно вешает нос, достает револьвер, приставляет его к виску и искусственно запускает бесконечный цикл. Какой-то while(true) do nothing, из которого не выбраться.

Нанесем все это на рисунок. Хотел сделать крупнее, оставил место, в итоге вышло как обычно. Что и сказать, художник из меня не ахти.

Надеюсь, что суть последних двух пунктов предельно ясна и видна на рисунке!



Вдох, выдох, по желанию перечитать и вникнуть в это хитрое построение. Если желания нет, можно поверить мне на слово, что оно непротеворечиво (помним про шаги 0 и 1, где мы сказали, что Stop существует и работает для любой программы). Основная сложность позади!

Шаг номер 3. Если Stop существует, и допущение в шаге 0 нам не врет, то написать Killer не очень сложно.

Стартуем процесс, читаем как угодно собственный исходный код включающий в себя и код Stop, передаем его в Stop, проверяем результат.

До этого момента мы дружно обсуждали Killer, восхищались им и плевались от его уродства.

В своем уродстве он прекрасен. Это можно понять, стоит только его запустить!

А теперь внимание. Если "внутренний" Killer останавливается, то Stop(Killer) выдаст True, а значит "внешний" Killer словит Макконахи (зачеркнуто), а значит внешний Killer сам запустит бесконечный цикл.

Если "внутренний" Killer не останавливается, то Stop(Killer) выдаст False, а значит "внешний" Killer с радостью остановится.

Проблема в том, что "внутренний" и "внешний" Killer это есть суть один и тот же Killer!

Одна и та же задача выдает противоречие...самой же себе? Без всяких внешних условий, условностей, внешних эффектов и зависимостей?

Что ж, у нас тут противоречие, возможно парадокс, по коням!

Из этого следует, что наше допущение из шага 0 было ошибочным, ведь все остальное - вполне тривиальные действия.

Получаем Парадокс Лжеца, работающий на Тьюринг Машине, и ломающий её логическую полноту.

Сметите программу Stop в совочек - мы доказали, что её не существует, разрушив допущение, являющееся необходимым для её существования.

Проблема остановки неразрешима. Скупая мужская слеза.

Надеюсь, доказательство обошлось без жертв среди мирного населения и нервных клеток!

Тут я хочу заметить, что на оригинальное доказательство выглядит несколько иначе, но эксплуатирует тот же трюк с самореференцией и парадоксом.

Я решил его немножечко видоизменить, потому что там и Машины Тьюринга, как обычные, так и "универсальные", и Гёделева Нумерация, и Диагональный Язык.

Мы бы застряли там навечно!

Итак, Проблема Остановки не разрешима (а полурешима). По крайней мере, в классических вычислительных моделях, представляющих из себя ограниченную реализацию Машины Тьюринга.

Кстати, тред про Машины Тьюринга у меня тоже есть!



Что это значит для всех нас? Какой нам прок от знания того, что что-то не имеет решения?

Разве это не грустно?

Нет, совсем нет! Это знание всего лишь экономит нам (особенно инженерам железа) время, силы и деньги.

Знание о неразрешимости Проблемы Остановки позволяет нам избежать участия в войне, в которой нам не победить, и вложить свой талант и ресурсы в более перспективные вещи.

Например, заранее определять таймауты для работы программы, избегать циклов в главных потоках без необходимости, и далее в том же духе.

Проблема Остановки очень полезна в Computer Science.

Если мы хотим доказать, что какая-то проблема полу-разрешима (если программа находит ответ тогда отлично, а если не находит то мы об этом не узнаем), то больше нет нужды танцевать со всеми этими парадоксами.

Достаточно доказать, что решение этой проблемы решило бы Проблему Остановки (а мы знаем, что она неразрешима), а значит и наша исходная проблема неразрешима!

А ещё она очень важна для философии, но в эти дебри я уже лезть, с вашего позволения, не стану.

Я дальше античных философов пока не залезал, и вообще очень слаб в этой теме.

Небольшой анонс: следующая на очереди у меня серия тредов про теорию алгоритмов и сами алгоритмы, с самого нуля, от мала до велика.

Постараюсь сделать их такими, чтоб и новичок смог разобраться в новой теме, и опытный человек нашел бы что-нибудь интересное!

Вот вам и вся Проблема Остановки!

Надеюсь, было не очень сложно и местами даже интересно. Как обычно, если возникли вопросы, или я где-то явно напортачил - милости прошу, со всем разберемся.

[@threadreaderapp](#) unroll

[@threadreaderapp](#) unroll

...