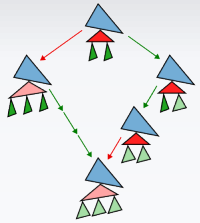
Рекурсивные языки и неразрешимость

Теория формальных языков

2022 г.

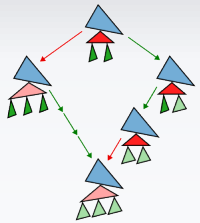


Неэквивалентность представлений

Известно, что:

- детерминированные конечные автоматы
- недетерминированные конечные автоматы
- регулярные выражения

описывают один и тот же класс языков. Насколько отличаются способы записи этих языков с точки зрения сложности статического анализа?



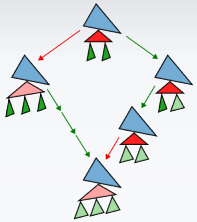
Неэквивалентность представлений

Известно, что:

- детерминированные конечные автоматы
- недетерминированные конечные автоматы
- регулярные выражения

описывают один и тот же класс языков. Насколько отличаются способы записи этих языков с точки зрения сложности статического анализа?

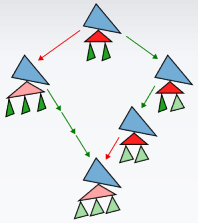
- минимизация детерминированных конечных автоматов — наивный алгоритм $\mathcal{O}(n^2)$;
- минимизация недетерминированных конечных автоматов — PSPACE-полная проблема;
- поиск минимальной звёздной высоты — пока что лучший алгоритм в 2EXP-SPACE.



Модели рекурсивных машин

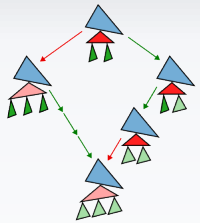
- Машины Тьюринга («автоматы»);
- Рекурсивные функции (μ-оператор);
- Алгорифмы Маркова (грамматики);
- λ-исчисление (HOF)...

Что в них общего и почему они порождают один и тот же класс языков?



λ-исчисление

- Две операции: абстракция по переменной и применение функции к аргументу.
- Бестиповая версия эквивалентна ТМ. Типизированные (ограниченные) версии в простейшем случае обладают свойством всюду завершаемости (т.е. можно по типу функции понять, что в любых сочетаниях с другими типизированными функциями она будет завершаться). В более сложных случаях — завершаемость с дополнительными условиями (по модулю более сложных упрощающих процедур, чем простое применение).



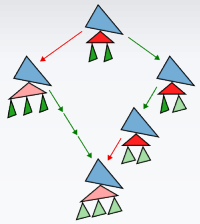
Формальный CTT (Derschowitz et al, 2008)

Аксиомы эффективности

Пусть модель вычислений $\mathcal{C}$ удовлетворяет критериям:

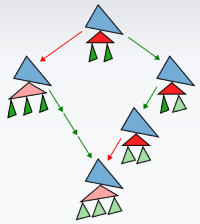
- Последовательное время: задано начальное множество состояний + функция перехода.
- Абстрактность состояний: состояние описывается конечным термом.
- Конечность пространства перебора: на каждом шаге вычислений существует лишь конечное множество термов, которые видны функции перехода.
- Инициальность. Множество начальных состояний — свободная алгебра, элементы которой можно записать с помощью конечной кодировки.

Тогда $\mathcal{C}$ вычисляет только функции, вычислимые по Тьюрингу.



Автоматизация работы с кодом

- Анализ программ (теорема Райса): если предикат не является тривиальным (т.е. выполняется хотя бы для одной МТ и не выполняется хотя бы для одной МТ), то он неразрешим для МТ.
- Генерация программ: дано описание $\mathtt{f}(\mathtt{x})$ в языке $\mathscr{L}$. Построить МТ, вычисляющую $\mathtt{f}(\mathtt{x})$. Насколько сильно эта задача зависит от $\mathscr{L}$?

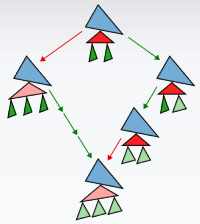


Формальная арифметика и монстры

В арифметике существуют доказуемые утверждения, которые невозможно доказать внутри неё самой.

Гидра Гудстейна

Рассмотрим $g(N, k)$ — экспоненциальное представление числа N в алфавите $\{1, \ldots, k\}$.
Пример: $g(11, 2) = 2^{(2+1)} + 2 + 1$.
Рассмотрим $G(N, m)$, где N записана в виде $g(N, m + 1)$. Формально заменим все $m + 1$ на $m + 2$ в этом представлении, а затем вычтем из результата 1. Будем шаг за шагом применять к результату преобразование G, увеличивая m. Доказуемо, что $\forall N \exists m(G(G(...G(N, 2)...), m) = 0)$. Однако это нельзя доказать в стандартной арифметике Пеано.



А если взять индукцию посильнее?

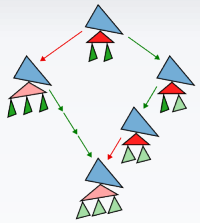
Теоремы о неполноте

Любая непротиворечивая теория, в которой выразима натуральная арифметика со сложением и умножением (а также принципом математической индукции), содержит утверждения, которые нельзя ни доказать, ни опровергнуть. Прежде всего, утверждение о её непротиворечивости.

Недоказушка из мира диофантовых уравнений

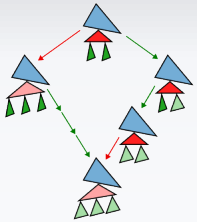
Для всех m, e существует N такое, что для всех a, b есть c, d, A, X такие что для всех x, y есть B, C, F, f, g, h, i, j, k, l, n, s, p, q

$$(x(y+f-x)(A+m+f-y)((A+f-d)^2+(dg+g-c+A)^2+(B+h-dx)^2+(dxi+i-c+B)^2+(C+j-dy)^2+(dyk+k-c+C)^2+(B+l+1-C)^2+(C+n-N)^2+(F+s-b(B+C^2))^2+(bp(B+C^2)+p-a+F)^2+((F-X)^2-qe)^2)=0).$$



Теорема о неполноте и Колмогоровская сложность

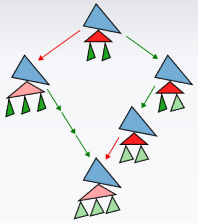
- Каков бы ни был язык $\mathscr{L}$, существует константа M такая, что для всех x утверждение $K_{\mathscr{L}}(x) > M$ неразрешимо.
- Идея доказательства: обозначим утверждение $K_{\mathscr{L}}(x) > C$ как $A_C(x)$. Если оно разрешимо, то напишем тупой алгоритм P_C, перебирающий все программы длиной до C и проверяющий, возвращают ли они формальный вывод $A_C(x)$ для какого-нибудь x. В случае, если возвращают, он печатает x, иначе продолжает работу. Среди достаточно больших C' найдётся какой-нибудь C' такой, что длина $P_{C'}$ меньше C'. Но тогда, если $P_{C'}$ остановится на некотором x, получится, что $K(x) < C'$.



Три кита анализа программ

- Теорема о существовании универсальной функции (самоинтерпретация);
- S–m–n теорема;
- Вторая теорема Клини о рекурсии.

Неразрешимые задачи в анализе программ различаются для разных моделей вычислений.



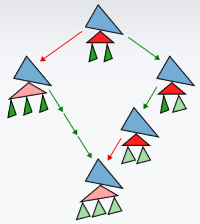
S–m–n теорема Клини

Пусть e — обозначение (геделевского)[1] кода e, а $[\![e]\!]$ — функция, которую он вычисляет.

Теорема

Существует вычислимая функция S_n^m такая, что
$\forall p, x_i, y_i([\![p]\!](x_1, \ldots, x_m, y_1 \ldots, y_n) =$
$[\![[\![S_n^m]\!](p, x_1, \ldots, x_m)]\!](y_1, \ldots, y_n))$.

[1] Вообще любого кода, допускающего интерпретацию универсальной функцией



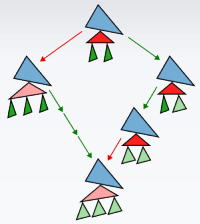
S–m–n теорема Клини

Пусть e — обозначение исходного кода e, а $[\![e]\!]$ — функция, которую он вычисляет.

Теорема

Существует вычислимая функция S_n^m (специализатор, частичный вычислитель) такая, что
$\forall p, x_i, y_i([\![p]\!](x_1, \ldots, x_m, y_1 \ldots, y_n) =$
$[\![[\![S_n^m]\!](p, x_1, \ldots, x_m)]\!](y_1, \ldots, y_n))$.

- Доказательство — Рефал-стиль! Захардкодим x_i в p.
- Существуют S_1^1–функции, превращающие алгоритм наивного поиска подстроки в строке в KMP после специализации по подстроке.



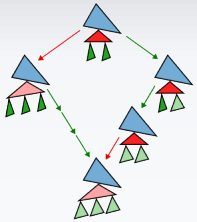
S–m–n теорема Клини

Пусть e — обозначение кода e, а $[\![e]\!]$ — функция, которую он вычисляет.

Теорема

Существует вычислимая функция S_n^m (специализатор, частичный вычислитель) такая, что
$\forall p, x_i, y_i([\![p]\!](x_1, \ldots, x_m, y_1 \ldots, y_n) =$
$[\![[\![S_n^m]\!](p, x_1, \ldots, x_m)]\!](y_1, \ldots, y_n))$.

- Доказательство — Рефал-стиль! Захардкодим x_i в p.
- Существуют S_1^1–функции, превращающие алгоритм наивного поиска подстроки в строке в KMP после специализации по подстроке.
- Генератор компиляторов — также S_1^1-случай. Достаточно взять текст интерпретатора L_1 в качестве y_1 для языка L_2, а программу перенести в x_1.

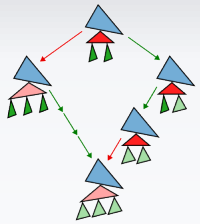


Вторая теорема Клини о рекурсии

Здесь также p, q — исходные коды программ.

Теорема

$\forall p \exists q \forall d (\llbracket q \rrbracket(d) = \llbracket p \rrbracket(q, d))$.



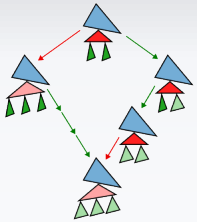
Вторая теорема Клини о рекурсии

Здесь также p, q — исходные коды программ.

Теорема

В любом достаточно мощном языке программирования программа может менять себя во время исполнения (рефлексивное программирование):
$\forall p \exists q \forall d([\![q]\!](d) = [\![p]\!](q, d))$.

- Средства снятия/навешивания функциональности: quote/unquote/eval, mu-функция, метаинтерпретация в прологе, etc.
- В компилируемых языках — модификация байт-кода.

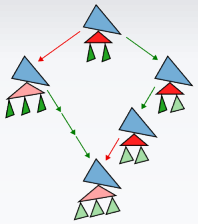


Лемма о генеричности

Лемма

Пусть M, N — термы λ-исчисления, причем вычисление M не завершается, а N имеет н.ф. Тогда для любого контекста C[ ]

$$C[M] =_\beta N \Rightarrow \forall L(C[L] =_\beta N)$$



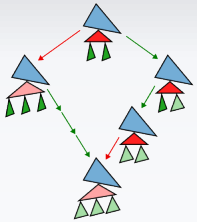
Лемма о генеричности

Лемма

Пусть M, N — термы λ-исчисления, причем вычисление M не завершается, а N имеет н.ф. Тогда для любого контекста C[ ]

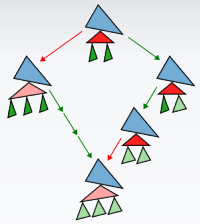
$$C[M] =_\beta N \Rightarrow \forall L(C[L] =_\beta N)$$

Неформально: чтобы отличить терм от других, его нужно (частично) вычислить.



Следствие леммы о генеричности

Возможно ли сравнивать *текстовые представления термов в чистом λ-исчислении?*



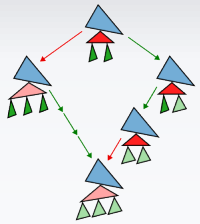
Следствие леммы о генеричности

Возможно ли сравнивать *текстовые представления термов в чистом* λ*-исчислении?*

Пусть существует комбинатор $\lambda x\, y.\mathsf{E}$, который возвращает $\mathbf{T}$, если буквальные представления x и y совпадают (в каком-либо смысле), и $\mathbf{F}$ иначе. Положим $\mathbf{\Omega} = (\lambda x.(x\ x))\ \lambda x.(x\ x)$, $\mathbf{I} = \lambda x.x$.

Тогда $\mathsf{E}\ \mathbf{\Omega}\ \mathbf{I} = \mathbf{F}$, $\mathsf{E}\ \mathbf{I}\ \mathbf{I} = \mathbf{T}$, но лемма о генеричности влечет, что $\forall \mathsf{N}(\mathsf{E}\ \mathsf{N}\ \mathbf{I}) = \mathbf{F}$.

Следовательно, E не определим в чистом λ-исчислении.

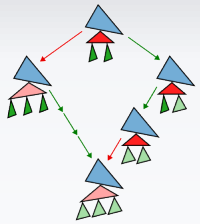


Анализ функций высших порядков

Здесь `unit = ()` (аналог `void`).

Рассмотрим функцию
`isAlwaysTrue :: ((unit → bool)→ bool)→ bool`
со следующей семантикой:

- `True`, если аргумент всегда возвращает `True`;
- `False`, если аргумент хотя бы на одном значении возвращает `False`.



Анализ функций высших порядков

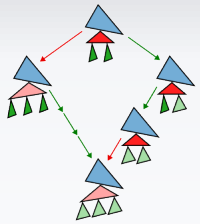
Здесь `unit = ()` (аналог `void`).

Рассмотрим функцию
`isAlwaysTrue :: ((unit → bool)→ bool)→ bool`
со следующей семантикой:

- `True`, если аргумент всегда возвращает `True`;
- `False`, если аргумент хотя бы на одном значении возвращает `False`.

Записать в λ-термах тривиально, потому что существуют только две нормальные формы, имеющие тип `unit → bool`:

```
isAlwaysTrue p = p (λ ().True)& p (λ ().False).
```



Анализ функций высших порядков

Здесь `unit = ()` (аналог `void`).

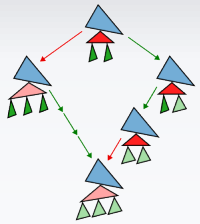
Рассмотрим функцию
`isAlwaysTrue :: ((unit → bool)→ bool)→ bool`
со следующей семантикой:

- `True`, если аргумент всегда возвращает `True`;
- `False`, если аргумент хотя бы на одном значении возвращает `False`.

Записать в λ-термах тривиально, потому что существуют только две нормальные формы, имеющие тип `unit → bool`:

```
isAlwaysTrue p = p (λ ().True)& p (λ ().False).
```

- Реализовать `isAlwaysTrue` в терминах машин Тьюринга нельзя.



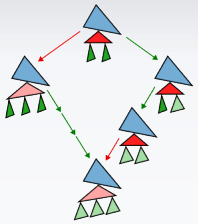
Опять формальный СТТ

Аксиомы эффективности

Пусть модель вычислений $\mathcal{C}$ удовлетворяет критериям:

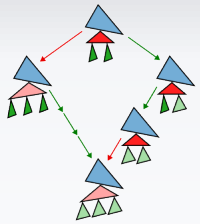
- Последовательное время: задано начальное множество состояний + функция перехода.
- Абстрактность состояний: состояние описывается конечным термом.
- Конечность пространства перебора: на каждом шаге вычислений существует лишь конечное множество термов, которые видны функции перехода.
- Инициальность. Множество начальных состояний — свободная алгебра, элементы которой можно записать с помощью конечной кодировки.

Тогда $\mathcal{C}$ вычисляет **функции** $\mathbb{N} \to \mathbb{N}$, вычислимые на ТМ.



Вероятностные вычисления

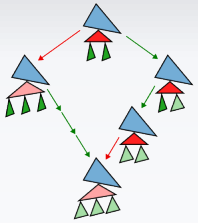
$\mathrm{pr}(S)$ — вероятность события S. Определим язык L, распознаваемый вероятностной ТМ M, следующим образом: $w \in L \Rightarrow \mathrm{pr}(\mathrm{accept}(M, w)) \geqslant 1 - 1/3$.



Вероятностные вычисления

$\mathrm{pr}(S)$ — вероятность события S. Определим язык L, распознаваемый вероятностной ТМ M, следующим образом: $w \in L \Rightarrow \mathrm{pr}(\mathrm{accept}(M, w)) \geqslant 1 - 1/3$.

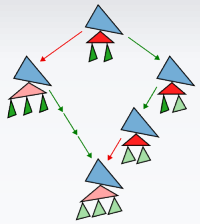
- Вероятностные ТМ эквивалентны стандартным ТМ;
- Выигрыш — на уровне сложности: см. определение простоты числа n по Миллеру–Рабину (в Bounded Probabilistic Polytime, $\mathcal{O}(n^3)$) vs. Aggraval–Kayal–Saxena (в Polytime, $\mathcal{O}(n^6)$ с очень большой мультипликативной константой).



Релятивистские вычисления

Что будет, если разрешить бесконечные вычисления?

Определим красно-зелёную TM (rgTM) как TM без конечных состояний, состояния которой поделены на два непересекающихся множества. Скажем, что слово w принимается rgTM $\mathcal{M}$, если $\mathcal{M}(w)$ стабилизируется в зелёных состояниях.



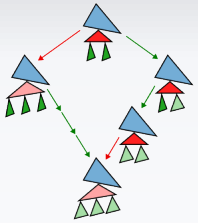
Релятивистские вычисления

Что будет, если разрешить бесконечные вычисления?

Определим красно-зелёную TM (rgTM) как TM без конечных состояний, состояния которой поделены на два непересекающихся множества. Скажем, что слово w принимается rgTM $\mathcal{M}$, если $\mathcal{M}(w)$ стабилизируется в зелёных состояниях.

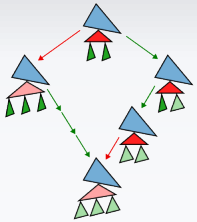
Решим проблему останова с помощью rgTM. Рассмотрим стандартную TM $\mathcal{M}$ и слово w. Объявим зелёными состояния, в которых $\mathcal{M}$ остановилась за x шагов.

- запускаем $\mathcal{M}(w)$;
- перешли в красное $\Rightarrow$ увеличиваем x, переходим в зелёное состояние и начинаем вычисление заново.



МТ с оракулами

Объявим оракулом предикат $\mathrm{P}(\mathrm{L}, w)$ такой, что $\mathrm{P}(\mathrm{L}, w) = \mathrm{True} \Leftrightarrow w \in \mathrm{L}$.

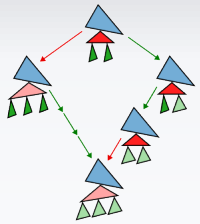


МТ с оракулами

Объявим оракулом предикат $P(L, w)$ такой, что $P(L, w) = True \Leftrightarrow w \in L$.

Тьюринг-сводимость

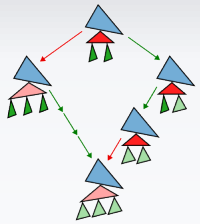
Скажем, что язык L_A Тьюринг–сводится к языку L_B ($L_A \geqslant_T L_B$), если ТМ с оракулом для L_B разрешает принадлежность к L_A.



Кванторная сложность

Определим кванторную сложность предиката P как количество перемен кванторов в его предварённой форме. Скажем, что $P \in \Sigma^0_n$, если P начинается с $\exists$ и имеет сложность n, и $P \in \Pi^0_n$, если P начинается с $\forall$ и имеет сложность n.

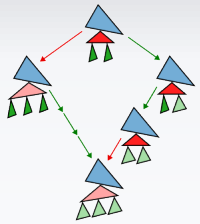
- $P(z) = \forall x, y(x \geqslant z \,\&\, y < x \,\&\, x * y = z \Rightarrow y = 1)$ на $\mathbb{N}$?



Кванторная сложность

Определим кванторную сложность предиката P как количество перемен кванторов в его предварённой форме. Скажем, что $P \in \Sigma^0_n$, если P начинается с $\exists$ и имеет сложность n, и $P \in \Pi^0_n$, если P начинается с $\forall$ и имеет сложность n.

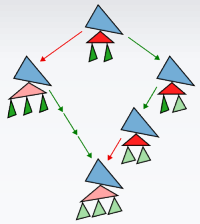
- $P(z) = \forall x, y(x \geqslant z \,\&\, y < x \,\&\, x * y = z \Rightarrow y = 1)$ на $\mathbb{N}$? Уровень Π^0_0 — ограниченные кванторы.



Кванторная сложность

Определим кванторную сложность предиката P как количество перемен кванторов в его предварённой форме. Скажем, что $P \in \Sigma_n^0$, если P начинается с $\exists$ и имеет сложность n, и $P \in \Pi_n^0$, если P начинается с $\forall$ и имеет сложность n.

- $P(z) = \forall x, y(x \geqslant z \,\&\, y < x \,\&\, x * y = z \Rightarrow y = 1)$ на $\mathbb{N}$? Уровень Π_0^0 — ограниченные кванторы.
- $A_M(w) = \exists t(\mathrm{accept}(M, w, t))$, где $\mathrm{accept}(M, w, t)$ — это «M принимает слово w за t шагов». Уровень Σ_1^0 (проблема останова) — перечислимые множества.



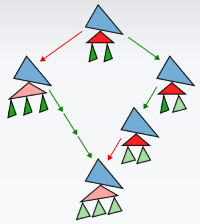
Кванторная сложность

Определим кванторную сложность предиката P как количество перемен кванторов в его предварённой форме. Скажем, что $P \in \Sigma_n^0$, если P начинается с $\exists$ и имеет сложность n, и $P \in \Pi_n^0$, если P начинается с $\forall$ и имеет сложность n.

- $P(z) = \forall x, y(x \geqslant z \,\&\, y < x \,\&\, x * y = z \Rightarrow y = 1)$ на $\mathbb{N}$? Уровень Π_0^0 — ограниченные кванторы.
- $A_M(w) = \exists t(\mathrm{accept}(M, w, t))$, где $\mathrm{accept}(M, w, t)$ — это «M принимает слово w за t шагов». Уровень Σ_1^0 (проблема останова) — перечислимые множества.

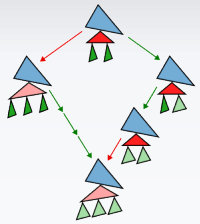
Все предикаты под кванторами — разрешимые!

Задача разрешения P (поиска точной позиции проблемы в арифметической иерархии) связана с поиском ограниченных кванторов в логической спецификации P.



Арифметическая иерархия

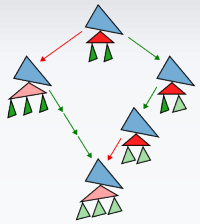
- Уровни Σ^0_n и Π^0_n не сводятся друг к другу ни при каком $n \neq 0$.
- $\Sigma^0_n \subset \Sigma^0_{n+1}$, $\Pi^0_n \subset \Pi^0_{n+1}$, причём включение строгое (Σ^0_{n+1} решает проблему останова для Σ^0_n).



Арифметическая иерархия

- Уровни Σ_n^0 и Π_n^0 не сводятся друг к другу ни при каком $n \neq 0$.
- $\Sigma_n^0 \subset \Sigma_{n+1}^0$, $\Pi_n^0 \subset \Pi_{n+1}^0$, причём включение строгое (Σ_{n+1}^0 решает проблему останова для Σ_n^0).

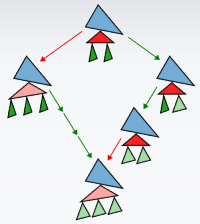
- Пустота: $\mathrm{EMPTY}(M) = \forall w, t(\neg\, \mathrm{accept}(M, w, t))$ — уровень Π_1^0. (что, если заменить TM M на PDA?)



Арифметическая иерархия

- Уровни Σ^0_n и Π^0_n не сводятся друг к другу ни при каком $n \neq 0$.
- $\Sigma^0_n \subset \Sigma^0_{n+1}$, $\Pi^0_n \subset \Pi^0_{n+1}$, причём включение строгое (Σ^0_{n+1} решает проблему останова для Σ^0_n).

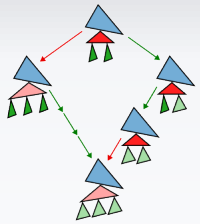
- Пустота: $\mathsf{EMPTY}(M) = \forall w, t(\neg\, \mathrm{accept}(M, w, t))$ — уровень Π^0_1. (что, если заменить TM M на PDA?)
- Завершаемость: $\mathsf{TOTAL}(M) = \forall w \exists t(\mathrm{accept}(M, w, t))$ — уровень Π^0_2 (монстры, т.е. сложно анализируемые TRS, появляются из-за неоднородности анализа).
- $\mathsf{FINITE}(M) = \exists n \forall w, t(|w| \leqslant n \vee \neg\, \mathrm{accept}(M, w, t))$ — на Σ^0_2.
- Поиск накачек:



Арифметическая иерархия

- Уровни Σ^0_n и Π^0_n не сводятся друг к другу ни при каком $n \neq 0$.
- $\Sigma^0_n \subset \Sigma^0_{n+1}$, $\Pi^0_n \subset \Pi^0_{n+1}$, причём включение строгое (Σ^0_{n+1} решает проблему останова для Σ^0_n).

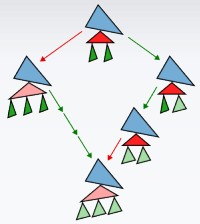
- Пустота: $\mathrm{EMPTY}(M) = \forall w, t(\neg\, \mathrm{accept}(M, w, t))$ — уровень Π^0_1. (что, если заменить TM M на PDA?)
- Завершаемость: $\mathrm{TOTAL}(M) = \forall w \exists t(\mathrm{accept}(M, w, t))$ — уровень Π^0_2 (монстры, т.е. сложно анализируемые TRS, появляются из-за неоднородности анализа).
- $\mathrm{FINITE}(M) = \exists n \forall w, t(|w| \leqslant n \vee \neg\, \mathrm{accept}(M, w, t))$ — на Σ^0_2.
- Поиск накачек:$\mathrm{PUMP}(M) = \exists p \forall w \exists x_1, x_2, y_1, y_2, z \forall i \exists t(|w| \geqslant p \Rightarrow w = x_1 y_1 z y_2 x_2 \;\&\; |y_1 z y_2| \leqslant p \;\&\; \mathrm{accept}(M, x_1 y_1^i z y_2^i x_2, t))$



Арифметическая иерархия

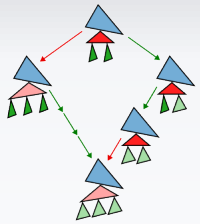
- Уровни Σ^0_n и Π^0_n не сводятся друг к другу ни при каком $n \neq 0$.
- $\Sigma^0_n \subset \Sigma^0_{n+1}$, $\Pi^0_n \subset \Pi^0_{n+1}$, причём включение строгое (Σ^0_{n+1} решает проблему останова для Σ^0_n).

- Пустота: $\mathsf{EMPTY}(M) = \forall w, t(\neg\, \mathrm{accept}(M, w, t))$ — уровень Π^0_1. (что, если заменить TM M на PDA?)
- Завершаемость: $\mathsf{TOTAL}(M) = \forall w \exists t(\mathrm{accept}(M, w, t))$ — уровень Π^0_2 (монстры, т.е. сложно анализируемые TRS, появляются из-за неоднородности анализа).
- $\mathsf{FINITE}(M) = \exists n \forall w, t(|w| \leqslant n \vee \neg\, \mathrm{accept}(M, w, t))$ — на Σ^0_2.
- Поиск накачек:$\mathsf{PUMP}(M) = \exists p \forall w \exists x_1, x_2, y_1, y_2, z \forall i \exists t(|w| \geqslant p \Rightarrow w = x_1 y_1 z y_2 x_2 \;\&\; |y_1 z y_2| \leqslant p \;\&\; \mathrm{accept}(M, x_1 y_1^i z y_2^i x_2, t))$— Σ^0_3 (разбиение ограниченное). $\mathsf{CFG}(M)$ — также задача уровня Σ^0_3.



Оракулы и иерархия

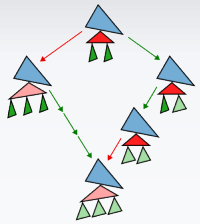
Ещё раз вернёмся к задаче определения завершаемости. Положим $\mathrm{TOTAL}(M) = \forall w A_M(w))$. Тогда $\mathrm{TOTAL}(M)$ — это Π_1^0 с оракулом в Σ_1^0.



Оракулы и иерархия

Ещё раз вернёмся к задаче определения завершаемости. Положим $\mathrm{TOTAL}(M) = \forall w A_M(w))$. Тогда $\mathrm{TOTAL}(M)$ — это Π_1^0 с оракулом в Σ_1^0.

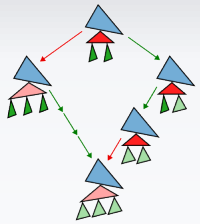
Общий принцип: применяя оракул уровня n к МТ, получаем перечислимость уровня $n+1$ (или её дополнение).



Оракулы и иерархия

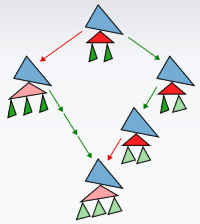
Ещё раз вернёмся к задаче определения завершаемости. Положим $\mathrm{TOTAL}(M) = \forall w A_M(w))$. Тогда $\mathrm{TOTAL}(M)$ — это Π_1^0 с оракулом в Σ_1^0.

Общий принцип: применяя оракул уровня n к МТ, получаем перечислимость уровня $n+1$ (или её дополнение). rgTM распознают языки на уровнях Σ_2^0, Π_2^0 арифметической иерархии (имея оракулы в Σ_0^1, Π_1^0).



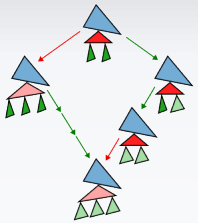
Неразрешимость и формальные языки

- Релятивистские вычисления не избавляют от неразрешимых задач. Разрешение проблемы останова в любой модели (в т.ч. более мощной, чем ТМ) — задача, неразрешимая в рамках самой этой модели.
- Эквивалентность по Тьюрингу–Чёрчу касается только интерпретации (прямого динамического анализа программ). На уровне метаинтерпретации (использования преобразований высшего порядка) модели существенно различаются.
- Сведение семантического анализа к синтаксическому (переход от динамического анализа к статическому) удаляет как минимум один уровень сложности в арифметической иерархии (замена неограниченного квантора ограниченным).



Горячие области применения ТФЯ

- Проверка правильности моделей программ;
- Встроенная статическая оптимизация программ;
- Статическая типизация и статический анализ как метод управления проектами;
- Формальное(!!) обучение формальных языков;
- Применение формальных языков для порождения корректных датасетов и преобразования датасетов;
- Автоматы для бесконечных языков и автоматы, моделирующие игровое поведение;
- Алгебраические представления формальных языков.



Благодарность

Посвящается всем студентам ИУ51Б и ИУ52Б, кто:

- вовремя и хорошо сдавал лабораторные;
- активно решал задачи и готовился к рубежным контролям;
- разбирался в теории, читал статьи и строил оптимальные алгоритмы...

Спасибо за ваш труд! Вы соавторы этого курса.