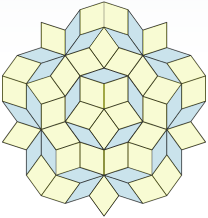


Введение в теорию моделей и SAT+SMT

Летняя практика, Переславль-Залесский

21 июля 2022 г.

Сигнатура с сортами

Определение

Сигнатура — множество пар вида $\langle$имя, местность$\rangle$. Алгебра $\mathcal{A}$ — набор носителей (сортов), выделенных элементов и функций первого порядка в сигнатуре данных носителей. На формальном языке,
$\mathcal{A} = \langle \{\mathcal{N}_i\}, \{c_i\}, \{f_i : \mathcal{N}_{i_1} \times \ldots \mathcal{N}_{i_{k_i}} \to \mathcal{N}'_i\} \rangle$.

Сигнатура с сортами

Определение

Сигнатура — множество пар вида $\langle$имя, местность$\rangle$. Алгебра $\mathcal{A}$ — набор носителей (сортов), выделенных элементов и функций первого порядка в сигнатуре данных носителей. На формальном языке, $\mathcal{A} = \langle\{\mathcal{N}_i\}, \{c_i\}, \{f_i : \mathcal{N}_{i_1} \times \ldots \mathcal{N}_{i_{k_i}} \to \mathcal{N}'_i\}\rangle$.

Иными словами, в алгебре у функций сигнатуры не бестиповые, а оснащены простыми типами.

- $f(x) = (x\ x)$ — нет сортов;
- $f(x, y) = \mathit{if}\ x\ \mathit{then}\ y * 2\ \mathit{else}\ y$ — есть сорта.

Сигнатура с сортами

Определение

Сигнатура — множество пар вида ⟨имя, местность⟩. Алгебра $\mathcal{A}$ — набор носителей (сортов), выделенных элементов и функций первого порядка в сигнатуре данных носителей. На формальном языке, $\mathcal{A} = \langle \{\mathcal{N}_i\}, \{c_i\}, \{f_i : \mathcal{N}_{i_1} \times \ldots \mathcal{N}_{i_{k_i}} \to \mathcal{N}_i'\} \rangle$.

Иными словами, в алгебре у функций сигнатуры не бестиповые, а оснащены простыми типами.

- $f(x) = (x\ x)$ — нет сортов;
- $f(x, y) = if\ x\ then\ y * 2\ else\ y$ — есть сорта.

Зачем многосортные алгебры нужны в CS?

- Сигнатура Σ — синтаксис ЯП;
- $\Sigma(X) + \Gamma_X$ (контекст) — множество термов ЯП.
- Алгебра над Σ — семантика ЯП.

Универсальные алгебры

Контекст — это способ отображения переменных в их значения. Выполнимость ϕ в контексте Γ пишем как $\Gamma \vdash \phi$. Например, $x : 3 \vdash x + 3 = 6$.

- В CS выделенные значения — функции от нуля аргументов (конструкторы).
- Каждой константе в Σ соответствует ровно один носитель из $\mathcal{A}$;
- Каждому функциональному символу в Σ соответствует отображение с такой же сигнатурой.
- Каждому присвоению сорта (утверждению о типизации) в контексте Γ вида $x_i : T_i$ соответствует окружение — отображение из x_i в множество носителей $\mathcal{A}$.

Интерпретации

Пусть η — окружение. Значение $\mathcal{A}[\![M]\!]\eta$ (читаем: интерпретация M) определяется рекурсивно.

- $\mathcal{A}[\![x]\!]\eta = \eta(x)$
- $\mathcal{A}[\![f(M_1, \ldots\ M_n)]\!]\eta = f^{\mathcal{A}}(\mathcal{A}[\![M_1]\!]\eta, \ldots, \mathcal{A}[\![M_n]\!]\eta)$.

Интерпретации

Пусть η — окружение. Значение $\mathcal{A}[\![M]\!]\eta$ (читаем: интерпретация M) определяется рекурсивно.

- $\mathcal{A}[\![x]\!]\eta = \eta(x)$
- $\mathcal{A}[\![f(M_1, \dots\ M_n)]\!]\eta = f^{\mathcal{A}}(\mathcal{A}[\![M_1]\!]\eta, \dots, \mathcal{A}[\![M_n]\!]\eta)$.

- Множество натуральных чисел — интерпретация термов над сигнатурой $\{s(\bullet), z\}$ — единственный одноместный конструктор + константа.
- Множество двоичных деревьев — интерпретация термов над сигнатурой $\{\langle\bullet, \bullet\rangle, e\}$ — единственный двухместный конструктор + константа.
- Свободная алгебра — каждый функциональный символ интерпретируется собой.

Интерпретации

Пусть η — окружение. Значение $\mathcal{A}[\![M]\!]\eta$ (читаем: интерпретация M) определяется рекурсивно.

- $\mathcal{A}[\![x]\!]\eta = \eta(x)$
- $\mathcal{A}[\![f(M_1, \ldots\ M_n)]\!]\eta = f^{\mathcal{A}}(\mathcal{A}[\![M_1]\!]\eta, \ldots, \mathcal{A}[\![M_n]\!]\eta)$.

- Множество натуральных чисел — интерпретация термов над сигнатурой $\{s(\bullet), z\}$ — единственный одноместный конструктор + константа.
- Множество двоичных деревьев — интерпретация термов над сигнатурой $\{\langle\bullet, \bullet\rangle, e\}$ — единственный двухместный конструктор + константа.
- Свободная алгебра — каждый функциональный символ интерпретируется собой.

Лемма о подстановке: $[\![M[x := N]]\!]\eta = [\![M]\!](\eta[x := [\![N]\!]\eta])$.

Модели

Выполнимость уравнения в окружении η принято обозначать $\mathcal{A}, \eta \models M = N[\Gamma]$, где η согласовано с Γ (т.е. значения переменных имеют правильные типы).

- Выполнимость в модели: $\mathcal{A} \models M = N[\Gamma]$ (для любого окружения).
- Общезначимость: $\models M = N[\Gamma]$ (для любого окружения в любой алгебре).

Свободные модели

Рассмотрим сигнатуру Σ и множество формул логики первого порядка P над Σ. Алгебра $\mathcal{A}$ называется свободной моделью (loose model) над $\langle \Sigma, \mathrm{P} \rangle$, если $\mathcal{A} \models \mathrm{P}$.

Свободные модели

Рассмотрим сигнатуру Σ и множество формул логики первого порядка P над Σ. Алгебра $\mathcal{A}$ называется свободной моделью (loose model) над $\langle \Sigma, P \rangle$, если $\mathcal{A} \models P$.

Проанализируем свободные модели спецификации:

сорта:	bool
сигнатура:	True, False : bool
	$\Rightarrow$: bool $\to$ bool
аксиомы:	$\forall x(True \Rightarrow x = x)$
	$\forall x(False \Rightarrow x = True)$

Свободные модели

Рассмотрим сигнатуру Σ и множество формул логики первого порядка P над Σ. Алгебра $\mathcal{A}$ называется свободной моделью (loose model) над $\langle \Sigma, P \rangle$, если $\mathcal{A} \models P$.

Проанализируем свободные модели спецификации:

сорта:	bool
сигнатура:	True, False : bool
	$\Rightarrow$: bool $\rightarrow$ bool
аксиомы:	$\forall x(True \Rightarrow x = x)$
	$\forall x(False \Rightarrow x = True)$

Много проблем: термы смешиваются между собой, а также могут появиться термы, которых нет в спецификации.

Инициальные алгебры

Пусть $\mathcal{C}$ — множество алгебр над сигнатурой Σ (Σ-алгебр) и $\mathcal{A} \in \mathcal{C}$. $\mathcal{A}$ называется инициальной, если для любой $\mathcal{A}' \in \mathcal{C}$ существует единственный гомоморфизм из $\mathcal{A}$ в $\mathcal{A}'$.

Инициальные алгебры

Пусть $\mathcal{C}$ — множество алгебр над сигнатурой Σ (Σ-алгебр) и $\mathcal{A} \in \mathcal{C}$. $\mathcal{A}$ называется инициальной, если для любой $\mathcal{A}' \in \mathcal{C}$ существует единственный гомоморфизм из $\mathcal{A}$ в $\mathcal{A}'$.

- «Никакого мусора» — в $\mathcal{A}$ должно быть так мало различных элементов, насколько возможно.
- «Никакой путаницы» — в $\mathcal{A}$ элементы должны быть равны только если без этого равенства не обойтись.

Пример инициальной модели

Уравнения — аксиомы инициальной модели. Поскольку каждая сигнатура определяет уникальный терм, и никаких других термов нет, то равенство можно понимать как факторизацию: термы попадают в один класс эквивалентности, если можно доказать, что они равны, и из этого класса рассматривается только один представитель.

сорта:	nat
сигнатура:	zero : nat
	succ, pred: nat $\to$ nat
уравнения:	pred(succ(x)) = x

Пример инициальной модели

Уравнения — аксиомы инициальной модели. Поскольку каждая сигнатура определяет уникальный терм, и никаких других термов нет, то равенство можно понимать как факторизацию: термы попадают в один класс эквивалентности, если можно доказать, что они равны, и из этого класса рассматривается только один представитель.

сорта:	nat
сигнатура:	zero : nat
	succ, pred: nat $\to$ nat
уравнения:	pred(succ(x)) $=$ x

Появляются термы вида pred(zero) (желаемые ли?). Также несократимые термы вида succ(pred(zero)) — точно не желаемые.

Спецификация списков

сорта:	list, atom, bool	
сигнатура:	true, false : bool	nil : list
	cond_x : bool $\to$ x $\to$ x$\to$ x	
	isempty? : list $\to$ bool	
	cons : atom $\to$ list $\to$ list	
	cdr : list $\to$ list	
	car : list $\to$ atom	
уравнения:	car(cons x l) = x	cdr(cons x l) = l
	isempty? nil = true	isempty? (cons x l) = false
	cond_a true x y = x	cond_a false x y = y
	cond_b true v1 v2 = v1	cond_b false v1 v2 = v2
	cond_l true l1 l2 = l1	cond_l false l1 l2 = l2

Спецификация списков

сорта:	list, atom, bool	
сигнатура:	true, false : bool	nil : list
	cond_x : bool $\to$ x $\to$ x$\to$ x	
	isempty? : list $\to$ bool	
	cons : atom $\to$ list $\to$ list	
	cdr : list $\to$ list	
	car : list $\to$ atom	
уравнения:	car(cons x l) = x	cdr(cons x l) = l
	isempty? nil = true	isempty? (cons x l) = false
	cond_a true x y = x	cond_a false x y = y
	cond_b true v1 v2 = v1	cond_b false v1 v2 = v2
	cond_l true l1 l2 = l1	cond_l false l1 l2 = l2

Выполняется ли уравнение (cons (car l)(cdr l)) = l в инициальной модели?

Спецификация списков

сорта:	list, atom, bool	
сигнатура:	true, false : bool	nil : list
	cond_x : bool $\to$ x $\to$ x$\to$ x	
	isempty? : list $\to$ bool	
	cons : atom $\to$ list $\to$ list	
	cdr : list $\to$ list	
	car : list $\to$ atom	
уравнения:	car(cons x l) = x	cdr(cons x l) = l
	isempty? nil = true	isempty? (cons x l) = false
	cond_a true x y = x	cond_a false x y = y
	cond_b true v1 v2 = v1	cond_b false v1 v2 = v2
	cond_l true l1 l2 = l1	cond_l false l1 l2 = l2

Выполняется ли уравнение (cons (car l)(cdr l)) = l в инициальной модели? Ответ: нет, проблема с термами вида (cdr nil) и (car nil).

Работа над ошибками

- Ничего не делаем. Пусть (car nil) и (cdr nil) будут новыми термами в инициальной алгебре. К чему это приведёт? (спойлер: к добавлению бесконечного множества ошибочных термов разных сортов)

Работа над ошибками

- Ничего не делаем. Пусть (car nil) и (cdr nil) будут новыми термами в инициальной алгебре. К чему это приведёт? (спойлер: к добавлению бесконечного множества ошибочных термов разных сортов)
- Null-неопределённости. Положим, что (car nil) произвольно, (cdr nil)=nil. Уничтожается индуктивное равенство между списками.

Работа над ошибками

- Ничего не делаем. Пусть (car nil) и (cdr nil) будут новыми термами в инициальной алгебре. К чему это приведёт? (спойлер: к добавлению бесконечного множества ошибочных термов разных сортов)
- Null-неопределённости. Положим, что (car nil) произвольно, (cdr nil)=nil. Уничтожается индуктивное равенство между списками.
- Введение терма-ошибки.

Наивная обработка ошибок

уравнения: $\text{car nil} = \text{error}_{a}$
$\text{cdr nil} = \text{error}_{l}$
$\text{cons error}_{a}\ \text{l} = \text{error}_{l}$
$\text{cons x error}_{l} = \text{error}_{l}$
$\text{car error}_{l} = \text{error}_{a}$
$\text{cdr error}_{l} = \text{error}_{l}$

Наивная обработка ошибок

уравнения:	$\mathrm{car\ nil} = \mathrm{error}_{\mathfrak{a}}$
	$\mathrm{cdr\ nil} = \mathrm{error}_{\mathfrak{l}}$
	$\mathrm{cons\ error}_{\mathfrak{a}}\ \mathrm{l} = \mathrm{error}_{\mathfrak{l}}$
	$\mathrm{cons\ x\ error}_{\mathfrak{l}} = \mathrm{error}_{\mathfrak{l}}$
	$\mathrm{car\ error}_{\mathfrak{l}} = \mathrm{error}_{\mathfrak{a}}$
	$\mathrm{cdr\ error}_{\mathfrak{l}} = \mathrm{error}_{\mathfrak{l}}$
	$\mathrm{isempty?\ error}_{\mathfrak{l}} = \mathrm{error}_{\mathfrak{b}}$
	$\mathrm{cond}_{\mathfrak{a}}\ \mathrm{error}_{\mathfrak{b}}\ \mathrm{x\ y} = \mathrm{error}_{\mathfrak{a}}$
	$\mathrm{cond}_{\mathfrak{b}}\ \mathrm{error}_{\mathfrak{b}}\ \mathrm{x\ y} = \mathrm{error}_{\mathfrak{b}}$
	$\mathrm{cond}_{\mathfrak{l}}\ \mathrm{error}_{\mathfrak{b}}\ \mathrm{x\ y} = \mathrm{error}_{\mathfrak{l}}$

Наивная обработка ошибок

уравнения:	$\mathsf{car\ nil} = \mathsf{error}_a$
	$\mathsf{cdr\ nil} = \mathsf{error}_l$
	$\mathsf{cons\ error}_a\ \mathsf{l} = \mathsf{error}_l$
	$\mathsf{cons\ x\ error}_l = \mathsf{error}_l$
	$\mathsf{car\ error}_l = \mathsf{error}_a$
	$\mathsf{cdr\ error}_l = \mathsf{error}_l$
	$\mathsf{isempty?\ error}_l = \mathsf{error}_b$
	$\mathsf{cond}_a\ \mathsf{error}_b\ \mathsf{x\ y} = \mathsf{error}_a$
	$\mathsf{cond}_b\ \mathsf{error}_b\ \mathsf{x\ y} = \mathsf{error}_b$
	$\mathsf{cond}_l\ \mathsf{error}_b\ \mathsf{x\ y} = \mathsf{error}_l$

Проблема с аксиомой car (cons x l) = x — из-за неё можно доказать, что в данной модели любой терм равен ошибке.

Кардинальность моделей

Скажем, что множества M_1 и M_2 имеют одинаковую кардинальность, если существует биекция из M_1 в M_2.

Теорема Кантора

Булеан 2^M всякого множества M имеет кардинальность, большую, чем у M.

Доказательство: Предположим, что требуемая биекция q существует. Возьмём $\mathcal{T} = \{y \in M \mid y \notin q(y)\}$. Теперь рассмотрим w такой, что $q(w) = \mathcal{T}$.

Теоремы Левенгейма–Сколема

О повышении мощности

Если алгебраическая спецификация S имеет модель хотя бы $\mathbb{N}$-кардинальности, тогда для любой кардинальности Ξ, превосходящей кардинальность этой модели, найдётся модель S кардинальности, не меньшей Ξ.

Теоремы Левенгейма–Сколема

О повышении мощности

Если алгебраическая спецификация S имеет модель хотя бы $\mathbb{N}$-кардинальности, тогда для любой кардинальности Ξ, превосходящей кардинальность этой модели, найдётся модель S кардинальности, не меньшей Ξ.

О понижении мощности

Если алгебраическая спецификация имеет контрмодель, тогда она имеет конечную контрмодель.

Основа современных методов анализа программ — Satisfiability Modulo Theories (поиск конечной контрмодели).

Бескванторный SMT-фрагмент

- Можно объявлять параметры: `(declare-fun x () String)`.
- Далее параметры связываются допущениями: `(assert (str.contains x "a"))`.

Когда описание построено, у его отрицания ищется конечная контрмодель. Таким образом, в отрицании нет формул с кванторами существования.

Примеры

```
(declare-fun x () String)
(declare-fun y1 () String)
(declare-fun y2 () String)
(declare-fun y3 () String)
(declare-fun z1 () String)
(declare-fun z2 () String)
(declare-fun z3 () String)

(assert (or
       (= x (str.++ y1 "A" y2 "B" y3))
       (= x (str.++ z1 "B" z2 "A" z3))))
(assert (or
       (not (str.contains x "A"))
       (not (str.contains x "B"))))
```

Примеры

```
(declare-fun z () String)
(declare-fun x () String)

(assert (= x (str.++ "a" z z) ))
(assert (= x (str.replace_all z "bb" "a" ) ))
```

Примеры

```
(declare-fun x () String)

(assert (not (str.contains x "AAB")))
(assert (str.contains
               (str.replace_all x "AB" "") "AB"))
```

DPLL

(Алгоритм Девиса–Патнема–Логемана–Лавленда)

- Основная решающая процедура SAT-солверов.
- Манипулирует условиями как пропозициональными переменными.
- Два основных подалгоритма (угадывание и распространение информации).
  - Если A_i одна в дизъюнкте, присваиваем ей значение, порождающее **T** (тривиализуем).
  - Если A_i входит во все дизъюнкты с одним знаком, тривиализуем все эти дизъюнкты.
  - Иначе строим гипотезу про значение очередной переменной и проверяем её на противоречие.

Принимает формулу в конъюнктивной нормальной форме.

Пример работы DPLL

$$(B \vee \neg A) \ \& \ (\neg C \vee \neg B) \ \& \ (A \vee C \vee \neg B) \ \& \ (A \vee D)$$

Пример работы DPLL

$$(\mathrm{B} \vee \neg\, \mathrm{A})\ \&\ (\neg\, \mathrm{C} \vee \neg\, \mathrm{B})\ \&\ (\mathrm{A} \vee \mathrm{C} \vee \neg\, \mathrm{B})\ \&\ (\mathrm{A} \vee \mathrm{D})$$

- Последний дизъюнкт тривиализуется подстановкой $\mathrm{D} := \mathbf{T}$.

Пример работы DPLL

$$(B \vee \neg A) \,\&\, (\neg C \vee \neg B) \,\&\, (A \vee C \vee \neg B) \,\&\, (A \vee D)$$

- Последний дизъюнкт тривиализуется подстановкой $D := \mathbf{T}$.
- Угадываем $B^d := \mathbf{T}$. Получаем следующую формулу:
  $\neg C \,\&\, (A \vee C)$

Пример работы DPLL

$$(B \vee \neg A) \mathbin{\&} (\neg C \vee \neg B) \mathbin{\&} (A \vee C \vee \neg B) \mathbin{\&} (A \vee D)$$

- Последний дизъюнкт тривиализуется подстановкой $D :=$ **T**.
- Угадываем $B^d :=$ **T**. Получаем следующую формулу: $\neg C \mathbin{\&} (A \vee C)$
- Устанавливаем $C :=$ **F** и распространяем: A

Пример работы DPLL

$$(B \vee \neg A) \mathbin{\&} (\neg C \vee \neg B) \mathbin{\&} (A \vee C \vee \neg B) \mathbin{\&} (A \vee D)$$

- Последний дизъюнкт тривиализуется подстановкой $D := \mathbf{T}$.
- Угадываем $B^d := \mathbf{T}$. Получаем следующую формулу: $\neg C \mathbin{\&} (A \vee C)$
- Устанавливаем $C := \mathbf{F}$ и распространяем: A
- Формула SAT, желаемая подстановка: $[A := \mathbf{T};\ B := \mathbf{T};\ C := \mathbf{F};\ D := \mathbf{T}]$.

DPLL(T)

Анализ логических формул над теориями.

Основное отличие от чистого DPLL:

- Вызывает солвер для теории T при угадывании (и в конце, если угадываний не было).

Если солвер обнаружил конфликт между условиями C_1 и C_2, тогда в формулу добавляется дизъюнкт $\neg C_1 \vee \neg C_2$.

Пример работы DPLL(T)

$((x+1>0) \vee (x+y>0))\ \&\ ((x<0) \vee (x+y>4))\ \&$
$\neg(x+y>0)$

Пример работы DPLL(T)

$((x + 1 > 0) \vee (x + y > 0)) \mathbin{\&} ((x < 0) \vee (x + y > 4)) \mathbin{\&} \neg(x + y > 0)$

- Распространяем $x + y > 0 := \mathbf{F}$.

Пример работы DPLL(T)

$((x+1>0) \vee (x+y>0))$ & $((x<0) \vee (x+y>4))$ & $\neg(x+y>0)$

- Распространяем $x+y>0 :=$ **F**.
- Распространяем $x+1>0 :=$ **T**.

Пример работы DPLL(T)

$((x+1>0) \vee (x+y>0))$ & $((x<0) \vee (x+y>4))$ & $\neg(x+y>0)$

- Распространяем $x+y>0 := \mathbf{F}$.
- Распространяем $x+1>0 := \mathbf{T}$.
- Угадываем $x<0 := \mathbf{T}$ и вызываем солвер. Солвер находит противоречие между $x<0$ и $x+1>0$. Добавляем подходящий дизъюнкт в формулу и откатываемся до $x<0 := \mathbf{F}$.
  $\underline{((x+1>0) \vee (x+y>0))}$ & $((x<0) \vee (x+y>4))$ & $\underline{\neg(x+y>0)}$ & $(\neg(x<0) \vee \neg(x+1>0))$
- Распространяем $x+y>4$, все дизъюнкты разобраны. Вызываем солвер. Находится противоречие, и точек отката уже нет. Формула UNSAT.

Объединение нескольких теорий

Объединение теорий способом Нельсона–Оппена:

- Разделить переменные (с введением дополнительных);
- Для каждого из подмножеств найти своё решение с помощью DPLL(T);
- Если оба решения существуют, согласовать их на разделённых переменных.

Объединение нескольких теорий

Объединение теорий способом Нельсона–Оппена:

- Разделить переменные (с введением дополнительных);
- Для каждого из подмножеств найти своё решение с помощью DPLL(T);
- Если оба решения существуют, согласовать их на разделённых переменных.

На языке классов типов получится следующее.

- Даны два разных типа, однако оба из них принадлежат классу Eq (то есть имеют равенство).
- Составные выражения одного типа рассматриваются как унитарные для другого.
- Строятся две отдельные модели, причём с дополнительными условиями на Eq.
- Происходит возврат к классу Eq и модель проверяется на противоречие.

Пример

$(((\mathrm{car}\ x) + 3 = y) \vee (x = (\mathrm{cons}\ y + 1\ \mathrm{nil}))) \ \&\ ((\mathrm{car}\ x) > y + 1)$

Пример

$$(((\mathrm{car}\ x) + 3 = y) \vee (x = (\mathrm{cons}\ y + 1\ \mathrm{nil}))) \ \&\ ((\mathrm{car}\ x) > y + 1)$$

- Разделяем переменные так, чтобы условия были на две теории отдельно:
  $((u_1 + 3 = y) \vee (x = (\mathrm{cons}\ u_2\ \mathrm{nil})))) \ \&\ (u_1 > y + 1) \ \&\ (u_1 = (\mathrm{car}\ x)) \ \&\ (u_2 = y + 1)$

Пример

$(((\mathrm{car}\ x) + 3 = y) \vee (x = (\mathrm{cons}\ y + 1\ \mathrm{nil}))) \ \&\ ((\mathrm{car}\ x) > y + 1)$

- Разделяем переменные так, чтобы условия были на две теории отдельно:
  $((u_1 + 3 = y) \vee (x = (\mathrm{cons}\ u_2\ \mathrm{nil})))) \ \&\ (u_1 > y + 1) \ \&\ (u_1 = (\mathrm{car}\ x)) \ \&\ (u_2 = y + 1)$
- Распространяем три последних условия: $(u_1 > y + 1) = \mathbf{T}$, $(u_1 = (\mathrm{car}\ x)) = \mathbf{T}$, $(u_2 = y + 1) = \mathbf{T}$.

Пример

$(((\mathrm{car}\ x) + 3 = y) \vee (x = (\mathrm{cons}\ y + 1\ \mathrm{nil}))) \ \&\ ((\mathrm{car}\ x) > y + 1)$

- Разделяем переменные так, чтобы условия были на две теории отдельно: $((u_1 + 3 = y) \vee (x = (\mathrm{cons}\ u_2\ \mathrm{nil})))) \ \&\ (u_1 > y + 1) \ \&\ (u_1 = (\mathrm{car}\ x)) \ \&\ (u_2 = y + 1)$
- Распространяем три последних условия: $(u_1 > y + 1) = \mathbf{T}$, $(u_1 = (\mathrm{car}\ x)) = \mathbf{T}$, $(u_2 = y + 1) = \mathbf{T}$.
- Угадываем $(u_1 + 3 = y) = \mathbf{T}$ и вызываем солверы для IA и для LIA. Первый Ок, на втором получаем противоречие с условием $u_1 > y + 1$.
- Добавляем условие $(\neg(u_1 + 3 = y) \vee \neg(u_1 > y + 1))$ и откатываемся к $(u_1 + 3 = y) = \mathbf{F}$.

Пример

$(((\mathrm{car}\ x) + 3 = y) \vee (x = (\mathrm{cons}\ y + 1\ \mathrm{nil}))) \ \&\ ((\mathrm{car}\ x) > y + 1)$

- Разделяем переменные так, чтобы условия были на две теории отдельно: $((u_1 + 3 = y) \vee (x = (\mathrm{cons}\ u_2\ \mathrm{nil})))) \ \&\ (u_1 > y + 1) \ \&\ (u_1 = (\mathrm{car}\ x)) \ \&\ (u_2 = y + 1)$
- Распространяем три последних условия: $(u_1 > y + 1) = \mathbf{T}$, $(u_1 = (\mathrm{car}\ x)) = \mathbf{T}$, $(u_2 = y + 1) = \mathbf{T}$.
- Угадываем $(u_1 + 3 = y) = \mathbf{T}$ и вызываем солверы для IA и для LIA. Первый Ок, на втором получаем противоречие с условием $u_1 > y + 1$.
- Добавляем условие $(\neg(u_1 + 3 = y) \vee \neg(u_1 > y + 1))$ и откатываемся к $(u_1 + 3 = y) = \mathbf{F}$.
- Распространяем $(x = (\mathrm{cons}\ u_2\ \mathrm{nil})) = \mathbf{T}$. Разбирать больше нечего, вызываем солверы для IA и для LIA. Оба Ок.

Пример

$(((\mathrm{car}\ x) + 3 = y) \vee (x = (\mathrm{cons}\ y + 1\ \mathrm{nil}))) \ \&\ ((\mathrm{car}\ x) > y + 1)$

- Разделяем переменные так, чтобы условия были на две теории отдельно:
  $((u_1 + 3 = y) \vee (x = (\mathrm{cons}\ u_2\ \mathrm{nil})))) \ \&\ (u_1 > y + 1) \ \&\ (u_1 = (\mathrm{car}\ x)) \ \&\ (u_2 = y + 1)$
- Угадываем $(u_1 + 3 = y) = \mathbf{T}$ и вызываем солверы для IA и для LIA. Первый Ок, на втором получаем противоречие с условием $u_1 > y + 1$.
- Добавляем условие $(\neg(u_1 + 3 = y) \vee \neg(u_1 > y + 1))$ и откатываемся к $(u_1 + 3 = y) = \mathbf{F}$.
- Распространяем $(x = (\mathrm{cons}\ u_2\ \mathrm{nil})) = \mathbf{T}$. Разбирать больше нечего, вызываем солверы для IA и для LIA. Оба Ок.
- Теперь требуется согласовать условия на разделённые переменные u_1 и u_2 (по равенству и неравенству). IA-солвер говорит, что $u_1 = u_2$; однако LIA-солвер выводит $u_1 \neq u_2$.

Пример

$(((\mathrm{car}\ x) + 3 = y) \vee (x = (\mathrm{cons}\ y + 1\ \mathrm{nil}))) \ \&\ ((\mathrm{car}\ x) > y + 1)$

- Разделяем переменные так, чтобы условия были на две теории отдельно: $((u_1 + 3 = y) \vee (x = (\mathrm{cons}\ u_2\ \mathrm{nil})))) \ \&\ (u_1 > y + 1) \ \&\ (u_1 = (\mathrm{car}\ x)) \ \&\ (u_2 = y + 1)$
- Угадываем $(u_1 + 3 = y) = \mathbf{T}$ и вызываем солверы для IA и для LIA. Первый Ок, на втором получаем противоречие с условием $u_1 > y + 1$.
- Добавляем условие $(\neg(u_1 + 3 = y) \vee \neg(u_1 > y + 1))$ и откатываемся к $(u_1 + 3 = y) = \mathbf{F}$.
- Распространяем $(x = (\mathrm{cons}\ u_2\ \mathrm{nil})) = \mathbf{T}$. Разбирать больше нечего, вызываем солверы для IA и для LIA. Оба Ок.
- Противоречие, модель UNSAT.

Свойства метода Нельсона–Оппена

- Корректен только для мономорфных сигнатур!
- Алгоритм всегда завершается.
- Если возвращается противоречие, то модель точно UNSAT.
- В противном случае модель точно SAT, если каждая из комбинируемых теорий стабильно инфинитна.

Теория T стабильно инфинитна $\Leftrightarrow$ каждая бескванторная формула теории T выполнима в бесконечной модели для T.

Пример, когда всё сложнее

$$(|(x\text{"abc"})| = |y| + 3)\ \&\ (xw = y)\ \&\ (\text{contains}\ w\ \text{"a"})$$

- После разделения переменных обе теории имеют модели.
- Однако при согласовании разделённых переменных мало информации о том, что выводится в терминах одной теории — от теории строк необходимы дополнительные условия:

Пример, когда всё сложнее

$(|(x\text{"}abc\text{"})| = |y| + 3)\ \&\ (xw = y)\ \&\ (\text{contains}\ w\ \text{"}a\text{"})$

- После разделения переменных обе теории имеют модели.
- Однако при согласовании разделённых переменных мало информации о том, что выводится в терминах одной теории — от теории строк необходимы дополнительные условия:$|y| = |x| + |w|$, $|x\text{"}abc = x + 3$, $|w| > 0$.
- Теории строк и LIA связаны дополнительными функциями, определёнными только в многосортной алгебре.

Язык строковых условий

- Предикаты:
  - `(str.contains str_arg str_sub)`
  - `(= str1 str2)`

Аргумент `str_is` функций `str.replace(.)*` и `str_sub` предиката `str.contains` не должен быть пустым словом (пустое слово в `.smt2` — это `""`).

Язык строковых условий

- Предикаты:
  - `(str.contains str_arg str_sub)`
  - `(= str1 str2)`
- Функции:
  - `(str.replace str_arg str_is str_to)`;
  - `(str.replace_all str_arg str_is str_to)`;
  - `(str.len str)` (базовая, но не является предметом практического задания);
  - `(str.++ str_arg+)`.

Аргумент `str_is` функций `str.replace(.)*` и `str_sub` предиката `str.contains` не должен быть пустым словом (пустое слово в `.smt2` — это `""`).

Общий алгоритм обработки строк

- Формируется первый блок условий на длины и отправляется в LIA-солвер;
- Оттуда возвращается в нормализованном виде;
- Производится расщепление по анализу длин;
- Производится нормализация;
- После всех указанных действий опять производится извлечение условий на длины и проверка из LIA-солвером.

Расщепление: анализ длин

- Конкатенация — очевидно (сумма).
- `str.contains` — неравенство.
- `str.replace` — уравнение с расщеплением внутри теории строк.
- `str.replace_all` — ??? уравнение с расщеплением???

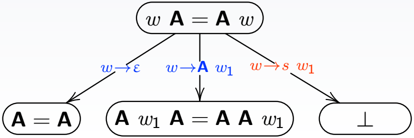
Преобразование Нильсена

Лемма Леви

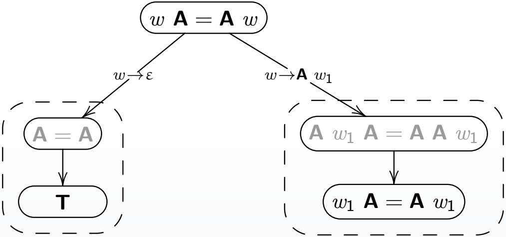
Если уравнение в словах имеет вид $x\,\Phi_1 = y\,\Phi_2$, тогда выполнено хотя бы одно из условий $x = y\,x' \vee y = x\,y'$.

Ветвление по подстановкам вида $x \to y\,x'$ и $y \to x\,y'$ называется преобразованием Нильсена. В классическом варианте переменные, вводимые для записи суффиксов, сохраняют имя исходных переменных.

Дерево решения уравнения

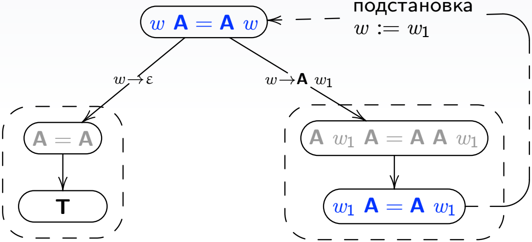


Если w не пуст, то начинается с **A**, либо уравнение не выполняется.



После подстановок $w \to \varepsilon$, $w \to \mathbf{A}\ w_1$ равные термы слева и справа сокращаются. Ветви дерева, приводящие к противоречию, отбрасываются.

Свертка дерева решения уравнения в граф



Уравнение w_1 **A** $=$ **A** w_1 повторяет исходное с точностью до переименования w_1 в w. Его развертка происходит точно так же.

Множество корней: $w \in$ **A***.

Что плохо умеют SMT-солверы

- Продвинутое сопоставление с образцом (в том числе решение уравнений в словах общего вида);
- Анализ кратности (т.к. алгебра целых чисел сводится к алгебре действительных, а потом рассматривается расщепление по целым значениям).

Много задач

```
(declare-fun x () String)
(declare-fun y () String)

(assert (= (str.++ x x y y y) (str.++ "BB" x x)))
```

```
(declare-fun z () String)

(assert (str.contains z "B"))
(assert (= (str.++ z "A") (str.++ "A" z) ))
```

Много задач

```
(declare-fun i () String)
(declare-fun x () String)

(assert (not (str.contains x "a")))
(assert (= i (str.replace_all x "a" "b" ) ))
(assert (not (= i x)))
```

```
(declare-fun x () String)

(assert (= (str.++ x x "AB" x "AB")
           (str.++ "BBAA" x x x)))
```

Много задач

```
(declare-fun x () String)
(declare-fun y () String)

(assert (= (str.++ "AB" x) (str.++ x "BA")))
(assert (= (str.++ y y) x))
```

```
(declare-fun x () String)
(declare-fun y () String)
(assert (= (str.++ "AB" x) (str.++ x y)))
(assert (not (str.contains x "AB")))
(assert (not (= x "A")))
(assert (not (= x "")))
```

Много задач

```
(declare-fun x () String)

(assert (str.contains (str.++ x x) "AB"))
(assert (not (str.contains x "AB")))
(assert (not (str.contains x "BA")))
```

```
(declare-fun x () String)

(assert (str.contains (str.replace_all x "AB" "") "CA"))
(assert (not (str.contains x "ABC")) )
(assert (not (str.contains x "CA")) )
```

Много задач

```
(declare-fun x () String)
(declare-fun y () String)

(assert (and (and
               (str.contains (str.++ x y) "AB")
               (not (= x (str.++ y "A")))
              )
             (not
               (or (str.contains y "B")
                   (str.contains (str.++ y x) "AB"))
               )))
```

Много задач

```
(declare-fun x () String)
(declare-fun y () String)
(declare-fun n1 () String)
(declare-fun n2 () String)
(declare-fun n3 () String)

(assert (not (str.contains (str.++ n1 n2 n3) "A")))
(assert (= (str.++ x x) (str.++ y y y)))
(assert (= x (str.++ n1 "A" n2 "A" n3)))
```

Источники проблем

- `str.contains` под отрицанием;
- нелинейные конкатенации с двух сторон от равенства;
- `str.replace_all` с неявным расщеплением.

Более точная характеризация?

Почему так странно?

Неразрешимая задача — задача, существование алгоритма (в общепринятом смысле) решения которой приводит к противоречию.

- Фрагмент `str.++`, `str.replace_all` с равенством неразрешим.
- Фрагмент только `str.++` с равенством PSPACE-полон...
- ...и NP-полон даже в предположении, что каждый параметр не входит в совокупность условий более чем дважды.
- Насчёт фрагмента `str.++`, `str.len` никто ничего не знает.