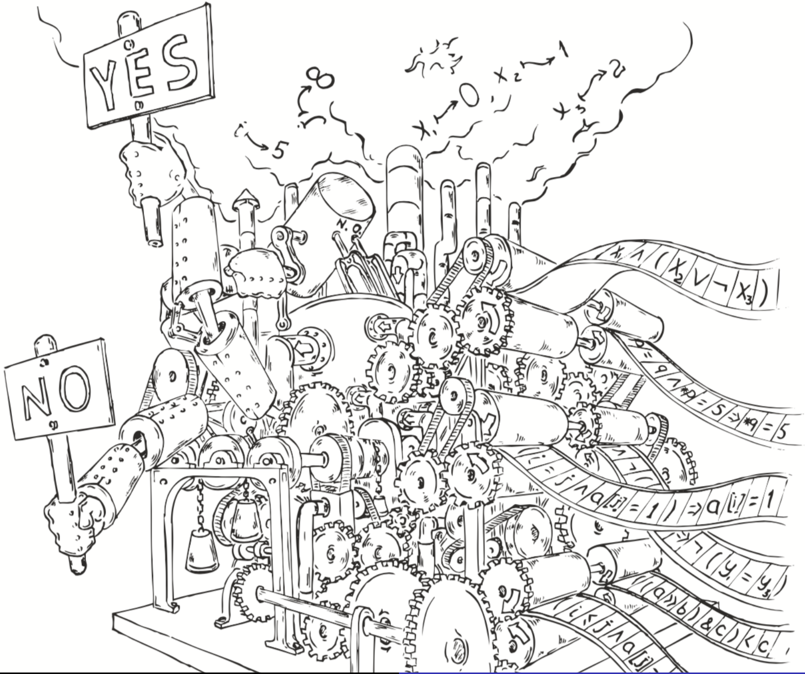

YES
NO

Задачи разрешимости логических формул и приложения
Лекция 4. SMT решатель. Davis–Putnam–Logemann–Loveland(Theory) фреймворк.

Роман Холин

Московский государственный университет

Москва, 2022

SMT решатель

- Что, если нам хочется задавать более сложные вопросы решателю? Например, разрешима ли такая формула: $(x = 4) \wedge ((y = 7) \vee (x = y))$
- Или такая: $(x + y = 3) \wedge (y - z = 7) \wedge (z * 2 = 4)$
- Или такая: $(lenght(s) = 3) \wedge (s[0] =\text{'a'}) \wedge (s[1] =\text{'b'}) \wedge (s[2] =\text{'c'})$

SMT решатель

- Что, если нам хочется задавать более сложные вопросы решателю? Например, разрешима ли такая формула: $(x = 4) \wedge ((y = 7) \vee (x = y))$
- Или такая: $(x + y = 3) \wedge (y - z = 7) \wedge (z * 2 = 4)$
- Или такая: $(lenght(s) = 3) \wedge (s[0] = \text{'a'}) \wedge (s[1] = \text{'b'}) \wedge (s[2] = \text{'c'})$

Пропозиционной логики для этого не достаточно. Для описания таких систем используется логика первого порядка и теории.

- равенства и неинтерпритируемых функций
- линейной арифметики
- векторы битов
- массивов

Теория первого порядка

- Сигнатура $\Sigma = (R, F, C, \rho)$
  R - множество символов для отношений (предикатов)
  F - множество функциональных символов
  C - множество констант
  Функция ρ, сопоставляющая элементам R и F их арность
- Переменные
- Логические операции: $\vee, \wedge, \rightarrow, \neg$
- Кванторы: $\forall, \exists$
- Скобки, запятые

Теория первого порядка

- Терм - функциональный символ, либо $f(t_1, \ldots, t_{\rho(f)})$, где f - символ арности $\rho(f)$, t_i - термы
- Атом - $p(t_1, \ldots t_{\rho(p)})$, где p - предикатный символ арности $\rho(p)$, t_i - термы
- Формула - либо атом, либо $\neg f_1$, $f_1 \vee f_2$, $f_1 \wedge f_2$, $f_1 \to f_2$, $\forall x f_1$, $\exists x f_1$, где f_1 и f_2 - формулы

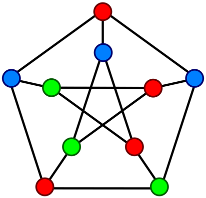
Теория первого порядка

- Задать модель:
  задать множество D - домен
  задать функцию σ, т.ч. каждому предикатному символу сопоставляет предикат, функциональному символу сопоставляет функцию, а константам сопоставлет элемент из D

Теория первого порядка

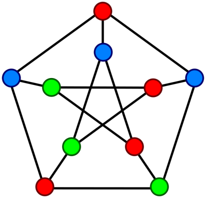
- Задать модель:
  задать множество D - домен
  задать функцию σ, т.ч. каждому предикатному и функциональному символу сопоставить предикат и функцию
- Пусть s - функция, которая сопоставляет каждой переменной некоторое значение из D
- Интерпретация формул относительно функции s - индуктивно вычислить формулу.

Раскраска графа



Имеется граф $G = (V, E)$. Можно ли его вершини расскрасить в k цветов так, чтобы никакие соседние вершины не были раскрашены в один и тот же цвет?

Раскраска графа



Имеется граф $G = (V, E)$. Можно ли его вершини расскрасить в k цветов так, чтобы никакие соседние вершины не были раскрашены в один и тот же цвет?

- x_i - целые
- $1 \leq x_i \leq c$
- $x_i \neq x_j$ для $(x_i, x_j) \in E$

Эквивалентность программ

Программа a:

```
int i, out_a = in;
for (int i = 0; i < 2; ++i) out_a = out_a * inn;
return out_a;
```

Программа b:

```
int out_b = (in * in) * in;
return out_b;
```

Эквивалентность программ

Программа a:

```
int i, out_a = in;
for (int i = 0; i < 2; ++i) out_a = out_a * in;
return out_a;
```

Программа b:

```
int out_b = (in * in) * in;
return out_b;
```

$\phi_a := (out\_a_0 = in\_a_0) \wedge (out\_a_1 = out\_a_0 * in\_a_0) \wedge (out\_a_2 = out\_a_1 * in\_a_0)$

$\phi_b := out\_b_0 = (in\_b_0 * in\_b_0) * in\_b_0$

Эквивалентность программ

Программа a:

```
int i, out_a = in;
for (int i = 0; i < 2; ++i) out_a = out_a * in;
return out_a;
```

Программа b:

```
int out_b = (in * in) * in;
return out_b;
```

$\phi_a := (out\_a_0 = in\_a_0) \wedge (out\_a_1 = out\_a_0 * in\_a_0) \wedge (out\_a_2 = out\_a_1 * in\_a_0)$

$\phi_b := out\_b_0 = (in\_b_0 * in\_b_0) * in\_b_0$

Чтобы программы были эквивалентны, должно выполняться:

$(in\_a_0 = in\_b_0) \wedge \phi_a \wedge \phi_b \to out\_a_2 = out\_b_0$

Определения

- $at(\phi)$ - множество атомов в формуле ϕ
- $at_i(\phi)$ - некоторый заданный порядок этих атомов
- Атом a сопоставим с булевой переменной $e(a)$ (такую процедуру будем называть булевское кодирование)
- $e(t)$ - булева формула, полученную булевым кодированием каждого атома формулы t (такую формулу будем называть пропозиционным скилетом формулы t)

Пример

- $\phi := x = y \vee x = z$
- $e(x = y) = b_1$
- $e(x = z) = b_2$
- $e(\phi) = b_1 \vee b_2$

Определения

- α - некоторая (возможно, частичная) оценка формулы $e(\phi)$
- $\alpha = \{e(at_1) \rightarrow FALSE, e(at_2) \rightarrow TRUE\}$
- $Th(at_i, \alpha) = at_i$, если $\alpha(at_i) = TRUE$, $\neg at_i$ иначе
- $Th(\alpha) = \{Th(at_i, \alpha) | e(at_i), \alpha\}$
- $\overline{Th(\alpha)}$ - конъюнкция всех элементов их $Th(\alpha)$

Ленивый алгоритм решение SMT задач

Пусть нам данн алгоритм (назовём его *Deduction*), который может решить $\overline{Th(\alpha)}$

- Вычислим $B = e(\phi)$

Пусть нам данн алгоритм (назовём его *Deduction*), который может решить $\overline{Th(\alpha)}$

- Вычислим $B = e(\phi)$
- Отправим B SAT-решателю

Ленивый алгоритм решение SMT задач

Пусть нам данн алгоритм (назовём его *Deduction*), который может решить $\overline{Th(\alpha)}$

- Вычислим $B = e(\phi)$
- Отправим B SAT-решателю
- Если получили "UNSAT то возвращаем "UNSAT"

Ленивый алгоритм решение SMT задач

Пусть нам данн алгоритм (назовём его *Deduction*), который может решить $\overline{Th(\alpha)}$

- Вычислим $B = e(\phi)$
- Отправим B SAT-решателю
- Если получили "UNSAT то возвращаем "UNSAT"
- Если получили "UNKNOWN то возвращаем "UNKNOWN"

Ленивый алгоритм решение SMT задач

Пусть нам данн алгоритм (назовём его *Deduction*), который может решить $\overline{Th(\alpha)}$

- Вычислим $B = e(\phi)$
- Отправим B SAT-решателю
- Если получили "UNSAT то возвращаем "UNSAT"
- Если получили "UNKNOWN то возвращаем "UNKNOWN"
- Если получили некоторую α, то вычисляем *Deduction* от $\overline{Th(\alpha)}$

Ленивый алгоритм решение SMT задач

Пусть нам данн алгоритм (назовём его *Deduction*), который может решить $\overline{Th(\alpha)}$

- Вычислим $B = e(\phi)$
- Отправим B SAT-решателю
- Если получили "UNSAT то возвращаем "UNSAT"
- Если получили "UNKNOWN то возвращаем "UNKNOWN"
- Если получили некоторую α, то вычисляем *Deduction* от $\overline{Th(\alpha)}$
- Если получили "SAT то возвращаем "SAT"

Ленивый алгоритм решение SMT задач

Пусть нам данн алгоритм (назовём его *Deduction*), который может решить $\overline{Th(\alpha)}$

- Вычислим $B = e(\phi)$
- Отправим B SAT-решателю
- Если получили "UNSAT то возвращаем "UNSAT"
- Если получили "UNKNOWN то возвращаем "UNKNOWN"
- Если получили некоторую α, то вычисляем *Deduction* от $\overline{Th(\alpha)}$
- Если получили "SAT то возвращаем "SAT"
- Если получили "UNKNOWN то возвращаем "UNKNOWN"

Ленивый алгоритм решение SMT задач

Пусть нам данн алгоритм (назовём его *Deduction*), который может решить $\overline{Th(\alpha)}$

- Вычислим $B = e(\phi)$
- Отправим B SAT-решателю
- Если получили "UNSAT то возвращаем "UNSAT"
- Если получили "UNKNOWN то возвращаем "UNKNOWN"
- Если получили некоторую α, то вычисляем *Deduction* от $\overline{Th(\alpha)}$
- Если получили "SAT то возвращаем "SAT"
- Если получили "UNKNOWN то возвращаем "UNKNOWN"
- Если получили "UNSAT то $B = B\wedge$ "блокирующий дизюнкт"t и начинаем со второго пункта

function $\textsc{Lazy}(\phi)$
  $B := e(\phi)$
  while true **do**
    $< \alpha, res >:= SAT - Solver(B)$
    if res = «Unsatisfiable» **then**
      return «Unsatisfiable»;
    else
      $< t, res >:= Deduction(\overline{Th(\alpha)})$
      if res = «Satisfiable» **then**
        return «Satisfiable»
      end if
      $B := B \wedge e(t);$
    end if
  end while
end function

Блокирующий дизюнкт

Так же называют леммой
Свойства:

- t - тавтология в T
- t - состоит только из атомов из ϕ
- t - "блокирует"α, т.е. при отправлении B SAT-решателю мы не сможем снова получить α

Блокирующий дизюнкт

Так же называют леммой
Свойства:

- t - тавтология в T
- t - состоит только из атомов из ϕ
- t - "блокирует"α, т.е. при отправлении B SAT-решателю мы не сможем снова получить α

Пример:

- Пусть $\alpha = \{e(at_1) \to FALSE, e(at_2) \to TRUE\}$
- $t := e(at_1) \vee \neg e(at_2)$

Есть и другие способы найти блокирующий дизюнкт

Ленивый алгоритм решение SMT задач

- В процессе работы ленивого алгоритма может возникнуть много блокирующих дизюнктов
- Многие теории можно свести к SAT задаче, но обычно существуют более эффективные алгоритмы решения теорий

Пример

$$x = y \wedge ((y = z \wedge \neg(x = z)) \vee x = z)$$

```
function CDCL
    while true do
        while BCP() = "conflict" do
            backtrack-level := Analyze-Conflict()
            if backtrack-level < 0 then
                return "Unsatisfiable"
            end if
            BackTrack(backtrack-level)
            if ¬ Decide() then
                return "Satisfiable"
            end if
        end while
    end while
end function
```


CDCL
LAZY-SMT
CDCL LAZY-SMT
DPLL(T)
imgflip.com

- Пусть B_i - формула, которая получилась Lazy на i-ом шаге
- По построению, B_i - подформула B_{i+1}
- Давайте не будем заново вызывать SAT решатель, а просто встроем в SAT решатель вызов решателя теории (и после вызова решателя теорий будем добавлять блокирующий дизъюнкт)

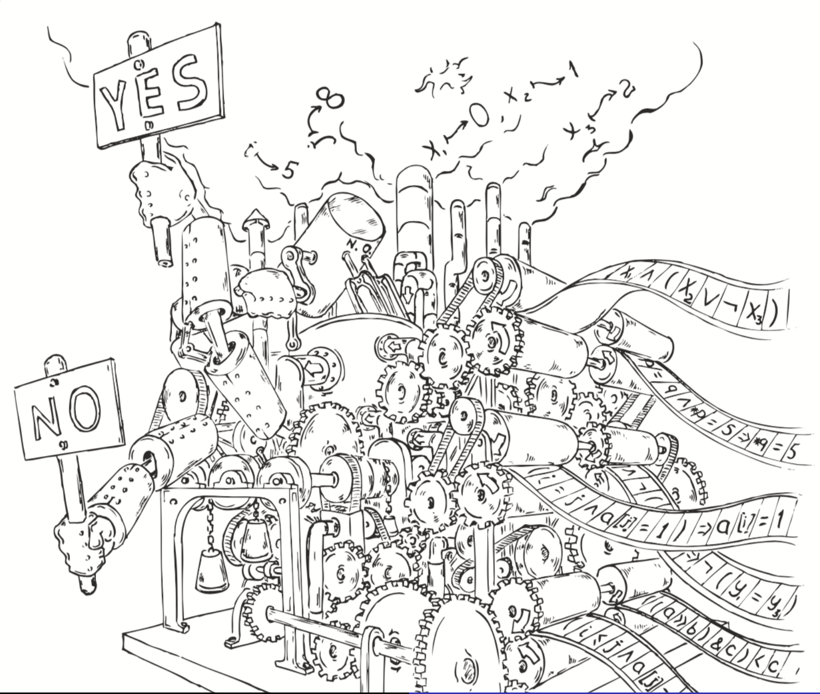
Lazy-CDCL(T)

```
AddClauses(cnf(e(ϕ)))
while true do
    while BCP() = «conflict» do
        backtrack-level := Analyze-Conflict()
        if backtrack-level < 0 then
            return «Unsatisfiable»
        else
            BackTrack(backtrack-level)
        end if
        if ¬ Decide() then
            < t, res >:= Deduction(\overline{Th(α)})
            if res = «Satisfiable» then
                return «Satisfiable»
            end if
            AddClauses(e(t))
        end if
    end while
```

- Что, если уже при данной частичной оценке формула в теории T уже не выполнима?
- Давайте всегда вызывать Deduction и добавлять к формуле невыолнимое ядро формулы

DPLL(T)

AddClauses(cnf(e(ϕ)))
while true **do**
    repeat
        while BCP() = «conflict» **do**
            backtrack-level := Analyze-Conflict()
            if backtrack-level < 0 **then**
                return «Unsatisfiable»
            else
                BackTrack(backtrack-level)
            end if
            $< t, res >:= Deduction(\overline{Th(\alpha)})$; AddClauses(e(t))
        end while
    until t = true
    if α is a full assignment **then**
        return «Satisfiable»
    end if
    Decide()


YES
NO