



Задачи разрешимости логических формул и  
приложения  
Лекция 7. Алгоритм  
Davis–Putnam–Logemann–Loveland (Theory).  
Логика равенства и неинтерпретируемых  
функций

Роман Холин

Московский государственный университет

Москва, 2021

- Во многих системах SAT решатель - лишь часть этой системы
- Запросы к решателю могут быть похожи
- Например, пусть есть робот, который находится на клеточном поле, которое является лабиринтом. Может ли робот выйти из него за  $k$  шагов?

- Во многих системах SAT решатель - лишь часть этой системы
- Запросы к решателю могут быть похожи
- Например, пусть есть робот, который находится на клеточном поле, которое является лабиринтом. Может ли робот выйти из него за  $k$  шагов?

Информацию можно использовать следующим образом:

- Использовать дизъюнкты

- Во многих системах SAT решатель - лишь часть этой системы
- Запросы к решателю могут быть похожи
- Например, пусть есть робот, который находится на клеточном поле, которое является лабиринтом. Может ли робот выйти из него за  $k$  шагов?

Информацию можно использовать следующим образом:

- Использовать дизъюнкты
- Использовать эвристические параметры

```
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  $\neg$  Decide() then
      return "Satisfiable"
    end if
  end while
end while
end function
```

```

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

```





- 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 встроим в SAT решатель вызов решателя теории (и после вызова решателя теорий будем добавлять блокирующий дизъюнкт)

```
AddClauses(cnf(e( $\phi$ )))  
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  $\neg$  Decide() then  
     $\langle t, res \rangle := Deduction(\overline{Th(\alpha)})$   
    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

```
AddClauses(cnf(e( $\phi$ )))  
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  
       $\langle t, res \rangle := Deduction(\overline{Th(\alpha)}); AddClauses(e(t))$   
    end while  
  until t = true  
  if  $\alpha$  is a full assignment then  
    return «Satisfiable»  
  end if  
Decide()
```

$formula : formula \vee formula | \neg formula | (formula) | atom$

$atom : term = term$

$term : identifier | constant$

# Логика равенства (EQ)

$formula : formula \vee formula | \neg formula | (formula) | atom$

$atom : term = term$

$term : identifier | constant$

Задача разрешимости формулы логики равенства - NP-полная

$$\phi = \phi'$$

В  $\phi'$  заменить все  $c_i$  на  $C_{c_i}$

Для всех  $i \neq j$  добавить в  $\phi'$  формулы  $C_{c_i} \neq C_{c_j}$

$$\phi' = \phi$$

В  $\phi'$  заменить все  $c_i$  на  $C_{c_i}$

Для всех  $i \neq j$  добавить в  $\phi'$  формулы  $C_{c_i} \neq C_{c_j}$

Далее будем считать, что в логике нет констант



# Логика неинтерпретируемых функций (UF)

$formula : formula \vee formula | \neg formula | (formula) | atom$   
 $atom : term = term | predicateSymbol(listof terms)$   
 $term : identifier | functionSymbol(listof terms)$

# Логика неинтерпретируемых функций (UF)

$formula : formula \vee formula | \neg formula | (formula) | atom$

$atom : term = term | predicateSymbol(listof terms)$

$term : identifier | functionSymbol(listof terms)$

Если predicateSymbol состоит только из  $\{=\}$ , то логику называют EUF

# Логика неинтерпретируемых функций(UF)

Как используется?

# Логика неинтерпретируемых функций(UF)

Как используется?

Если можем доказать в этой логике, то можем доказать и в общей формуле

# Логика неинтерпретируемых функций(UF)

Как используется?

Если можем доказать в EUF, то можем доказать и в частной логике

Обратное не верно: если формула не тавтология в EUF, то не факт, что это не тавтология в EUF

# 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;
```

# 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:

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

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;
```

Перепишем формулу в EUF. Будем считать, что  $*$  - это некоторый двуместный функциональный символ  $G(,)$



# 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;
```

Перепишем формулу в EUF. Будем считать, что  $*$  - это некоторый двуместный функциональный символ  $G(,)$

$$\phi_a := (out\_a_0 = in\_a_0) \wedge (out\_a_1 = G(out\_a_0, in\_a_0)) \wedge (out\_a_2 = G(out\_a_1, in\_a_0))$$
$$\phi_b := out\_b_0 = G(G(in\_b_0, in\_b_0), in\_b_0)$$

Чтобы программы были эквивалентны, должно выполняться:

$(in\_a_0 = in\_b_0) \wedge \phi_a \wedge \phi_b \rightarrow out\_a_2 = out\_b_0$ , но теперь нам не нужно знать природу  $G$

- Построить замыкание классов эквивалентности:  
Поместить в один класс эквивалентности термы, если  $t_1 = t_2$   
Слить все в один класс эквивалентности формулы  $F(t_i)$ , такие что  $F(t_i) = F(t_j)$  и  $t_i$  и  $t_j$  в одном классе эквивалентности  
Повторять, пока есть что сливать
- Если какие-то  $\phi_i$  и  $\phi_j$  в одном классе эквивалентности и есть неравенство  $\phi_i \neq \phi_j$ , то вернуть результат «Unsatisfiable». Иначе вернуть «Satisfiable»

