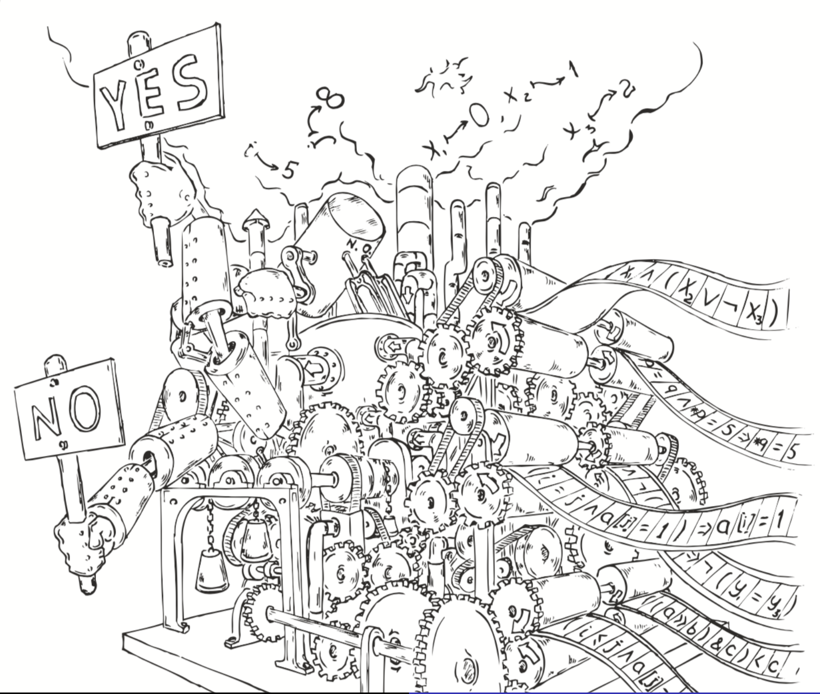

YES
NO
$i \to 5$
$x_1 \to 0$
$x_2 \to 1$
$x_3 \to 2$
$(x_2 \vee \neg x_3)$
$a[i] = 1$

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

Роман Холин

Московский государственный университет

Москва, 2021

Ещё небольшая оптимизация CDCL

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

Ещё небольшая оптимизация CDCL

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

Информацию можно использовать следующим образом:

- Использовать дизъюнкты

Ещё небольшая оптимизация CDCL

- Во многих системах SAT решатель - лишь часть этой системы
- Запросы к решателю могут быть похожи
- Например, пусть есть робот, который нахоидтся 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 ¬ Decide() then
                return "Satisfiable"
            end if
        end while
    end while
end function
```

Lazy

```
function LAZY(ϕ)
    B := e(ϕ)
    while true do
        < α, res >:= SAT − Solver(B)
        if res = «Unsatisfiable» then
            return «Unsatisfiable»;
        else
            < t, res >:= Deduction(¬Th(α))
            if res = «Satisfiable» then
                return «Satisfiable»
            end if
            B := B ∧ e(t);
        end if
    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)

```
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(¬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

Lazy-CDCL

```
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(¬Th(α)); AddClauses(e(t))
        end while
    until t = true
    if α is a full assignment then
        return «Satisfiable»
    end if
    Decide()
```

Логика равенства

$$formula : formula \vee formula | \neg formula | (formula) | atom$$
$$atom : term = term$$
$$term : identifier | constant$$

Логика равенства (EQ)

$$formula : formula \vee formula | \neg formula | (formula) | atom$$
$$atom : term = term$$
$$term : identifier | constant$$

Задача разрешимости формулы логики равенства - NP-полная

Удаление констант

$\phi = \phi'$
В ϕ' заменить все c_i на C_{c_i}
Для всех $i \neq j$ добавить в ϕ' формулы $C_{c_i} \neq C_{c_j}$

Удаление констант

$\phi' = \phi$
В ϕ' заменить все c_i на C_{c_i}
Для всех $i \neq j$ добавить в ϕ' формулы $C_{c_i} \neq C_{c_j}$
Далее будем считать, что в логике нет констант

Логика неинтерпретируемых функций (UF)

$$
\begin{aligned}
&formula : formula \vee formula | \neg formula | (formula) | atom \\
&atom : term = term | predicateSymbol(listofterms) \\
&term : identifier | functionSymbol(listofterms)
\end{aligned}
$$

Логика неинтерпретируемых функций (UF)

$formula : formula \vee formula | \neg formula | (formula) | atom$
$atom : term = term | predicateSymbol(listofterms)$
$term : identifier | functionSymbol(listofterms)$
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 фомруле

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:

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

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;

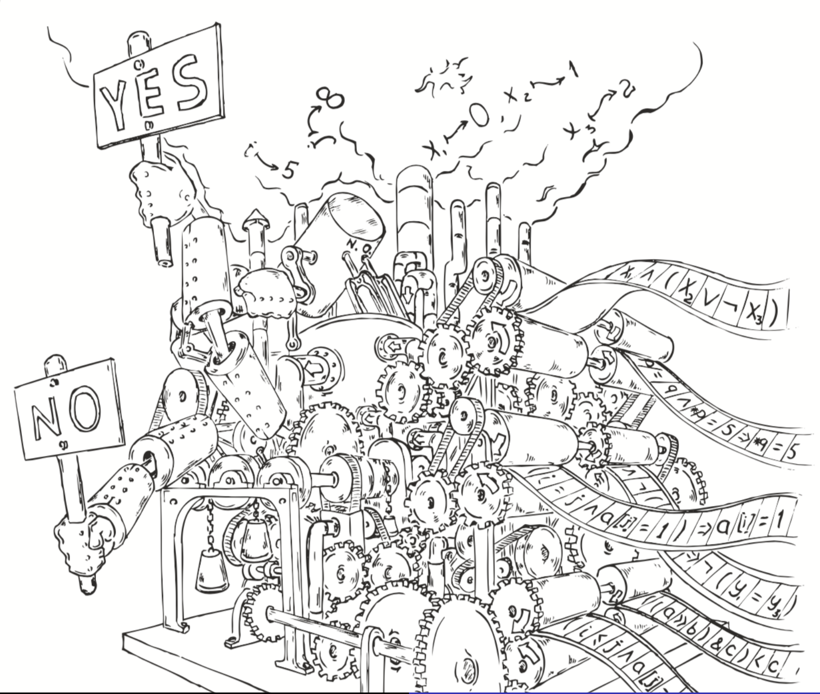
return out_b;

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

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

Замыкание классов эквивалентности

- 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 ϕ_i и ϕ_j в одном классе эквивалентности и есть неравенство $\phi_i \neq \phi_j$, то вернуть результат «Unsatisfiable». Иначе вернуть «Satisfiable»


YES
NO
N.O.
i → 5
x1 → 0
x2 → 1
x3 → 2