Algoritmos para o problema de Satisfatibilidade (SAT solvers)

[Cap 1.6 - M. Huth and M. Ryan. *Logic in Computer Science: Modelling and reasoning about systems*.]

Integrantes:
Francisco Etcheverria - 341816
Guilherme Pinheiro - 342346
Henrique Trein - 00341853
Luiz Fernando de Oliveira - 00337514

Professor: André Grahl Pereira
INF05508 – Lógica para Computação
2022/2

Introdução:

- Objeto de estudo desde 1960
- Satisfatibilidade
- NP - Completo (Cook e Levin)
- DPLL (Davis–Putnam–Logemann–Loveland)
  CDCL (Conflict-Driven Clause Learning)

NP - Completo:
Classe de problema que não possui soluções de tempo polinomial, pelo menos até os dias atuais (intratáveis). Todavia, dado um certificado de solução, pode-se verificar se está correto em tempo polinomial. São aplicáveis apenas a problemas de decisão (os quais têm como resposta “Sim/Não” “1/0” “T/F”). Entretanto, algumas vezes, podem ser relacionados a problemas de optimização.

$f(n)$	$n = 20$	$n = 40$	$n = 60$	$n = 80$	$n = 100$
n	$2{,}0\times10^{-11}$seg	$4{,}0\times10^{-11}$seg	$6{,}0\times10^{-11}$seg	$8{,}0\times10^{-11}$seg	$1{,}0\times10^{-10}$seg
n^2	$4{,}0\times10^{-10}$seg	$1{,}6\times10^{-9}$seg	$3{,}6\times10^{-9}$seg	$6{,}4\times10^{-9}$seg	$1{,}0\times10^{-8}$seg
n^3	$8{,}0\times10^{-9}$seg	$6{,}4\times10^{-8}$seg	$2{,}2\times10^{-7}$seg	$5{,}1\times10^{-7}$seg	$1{,}0\times10^{-6}$seg
n^5	$2{,}2\times10^{-6}$seg	$1{,}0\times10^{-4}$seg	$7{,}8\times10^{-4}$seg	$3{,}3\times10^{-3}$seg	$1{,}0\times10^{-2}$seg
2^n	$1{,}0\times10^{-6}$seg	1,0seg	13,3dias	$1{,}3\times10^{5}$séc	$1{,}4\times10^{11}$séc
3^n	$3{,}4\times10^{-3}$seg	140,7dias	$1{,}3\times10^{7}$séc	$1{,}7\times10^{19}$séc	$5{,}9\times10^{28}$séc

Definição de NP-Completo: tradução livre do livro “Introduction to Algorithms” de Thomas H. Cormen.
Tabela de dados: F. K. Miyazawa. Instituto de Computação/Unicamp, 2021.

⚠️ OS SOLUCIONADORES DE SATISFATIBILIDADE A SEREM DESCRITOS, SÃO DOCUMENTADOS NO LIVRO DE M. HUTH E M. RYAN *"Logic in Computer Science: Modelling and reasoning about systems"*.

Linear Solver:

- Baseado em propagação de marcas
- Eficiente mas incompleto
- Objetivo: determinar restrições para que a *fbf* seja verdadeira.
- Marcas: para que a *fbf* faça sentido, uma dada subfórmula precisa ter valor lógico X. Logo, é feita uma marcação.
- DAGs

1 .Transformação da fbf em uma *fbf* com apenas conjunções e negações:

$$T(p) = p$$
$$T(\phi_1 \wedge \phi_2) = T(\phi_1) \wedge T(\phi_2)$$
$$T(\phi_1 \rightarrow \phi_2) = \neg(T(\phi_1) \wedge \neg T(\phi_2))$$
$$T(\neg\phi) = \neg T(\phi)$$
$$T(\phi_1 \vee \phi_2) = \neg(\neg T(\phi_1) \wedge \neg T(\phi_2))$$

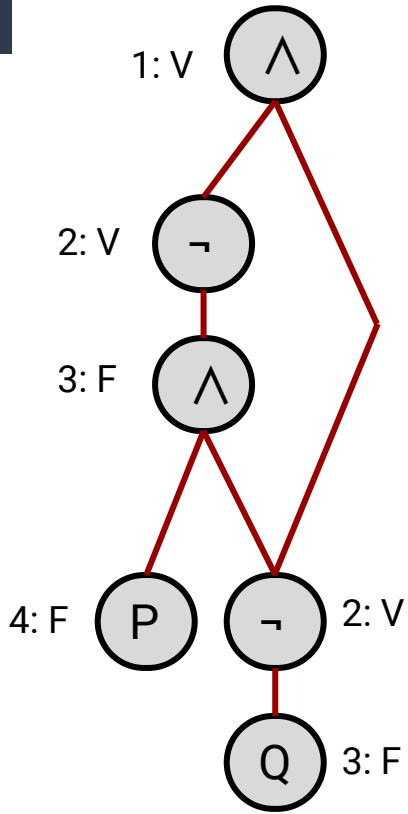
2. Criação da árvore sintática:

3. Criação do DAG:

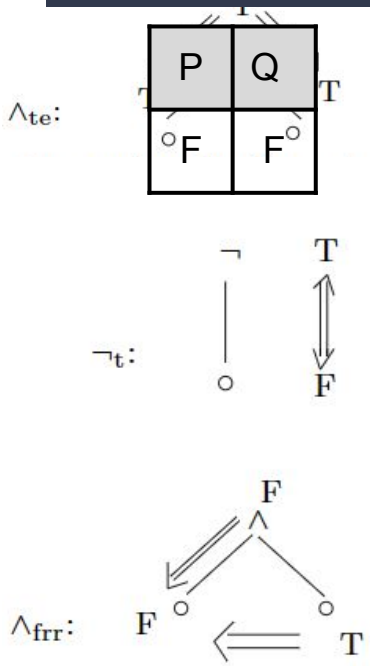
4. Propagação de marcas (T/F) com enumeração da lógica:
   - seguir as regras do slide 5

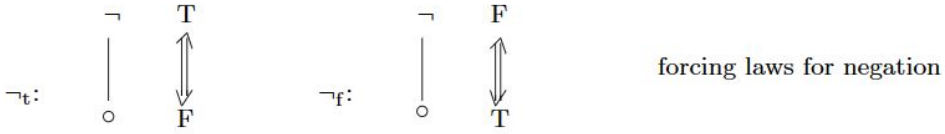
Exemplo: Solucionador Linear

$\neg(P \wedge \neg Q) \wedge \neg Q)$

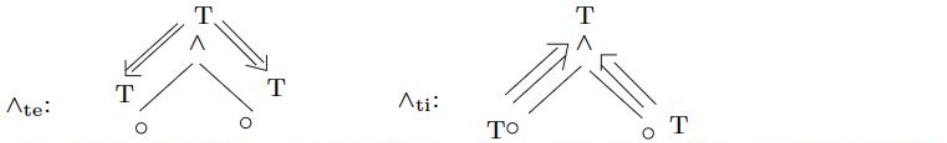


Fórmula satisfazível



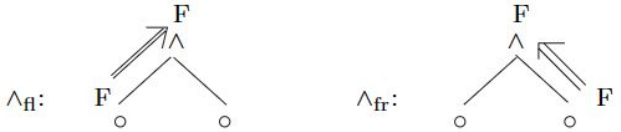


forcing laws for negation

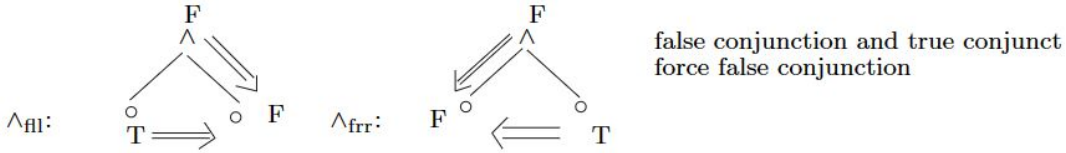


true conjunction forces true conjuncts

true conjunctions force true conjunction



false conjuncts
force false conjunction



false conjunction and true conjunct
force false conjunction

EXERCÍCIOS:

a) $\neg\neg(P \wedge \neg\neg(\neg P \vee Q))$

b) $(P \wedge \neg(P \wedge \neg Q)) \wedge \neg Q$

c) $(P \rightarrow Q) \wedge (P \rightarrow \neg Q) \wedge (P \vee R)$

PROBLEMA!

$$T(p) = p$$
$$T(\phi_1 \wedge \phi_2) = T(\phi_1) \wedge T(\phi_2)$$
$$T(\phi_1 \rightarrow \phi_2) = \neg(T(\phi_1) \wedge \neg T(\phi_2))$$
$$T(\neg\phi) = \neg T(\phi)$$
$$T(\phi_1 \vee \phi_2) = \neg(\neg T(\phi_1) \wedge \neg T(\phi_2))$$

Cubic Solver:

- Utilizado quando todas as marcas são consistentes, mas nem todos os nodos foram marcados.
- Continua sendo eficiente mas incompleto.

Após ser aplicado o solucionador linear e ainda existirem nodos não marcados, escolha um nó *n*.

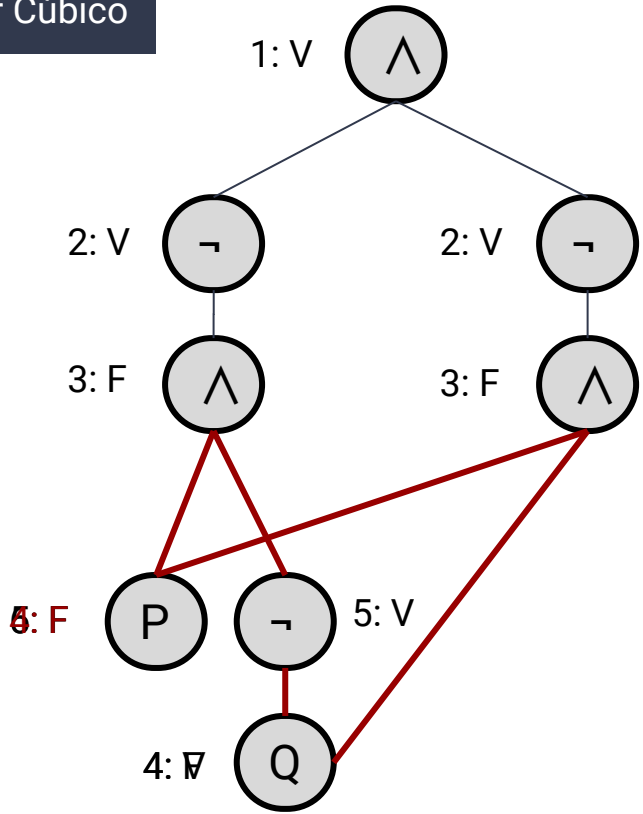
1. Determine marcas temporárias forçadas quando *n* = T.
2. Determine marcas temporárias forçadas quando *n* = F.

Se em ambos testes houverem contradições, o algoritmo para.
Caso contrário, todos os nodos que receberam a **mesma** valoração em ambos os testes passam a ter aquela valoração **permanentemente.**
O processo se repete até que:

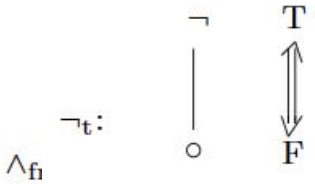
1. Existem marcas permanentes contraditórias, informando que nossa fórmula é **insatisfazível**.
2. Todos os nodos foram marcados e não existem contradições, logo nossa fórmula é **satisfazível**.
3. Todos os nodos não marcados permanentemente foram testados, mas não há valorações iguais para ambos testes de um nodo. Neste caso, terminamos a análise de modo **inconclusivo**.

Exemplo: Solucionador Cúbico

¬(P∧¬Q)∧¬(P∧Q)



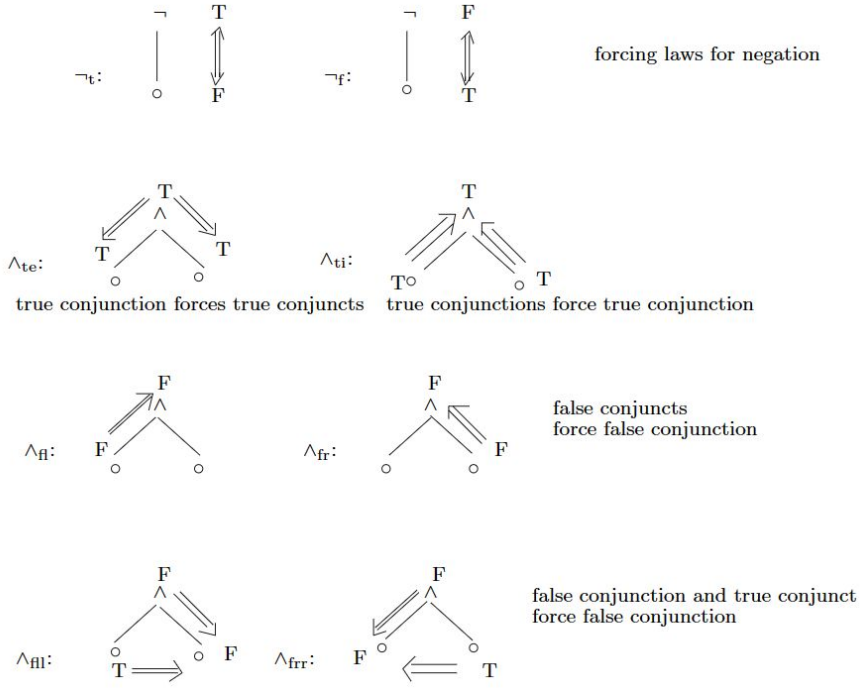
Q	P
V	F
F	F



EXERCÍCIOS:

a) $(P \rightarrow Q) \wedge (P \rightarrow \neg Q) \wedge (P \vee R)$

b) $\neg(((\neg P \wedge \neg(P \wedge Q)) \wedge \neg(\neg(P \wedge Q) \wedge R)) \wedge (\neg(\neg(P \wedge Q) \wedge R) \wedge \neg R))$



OTIMIZAÇÃO:

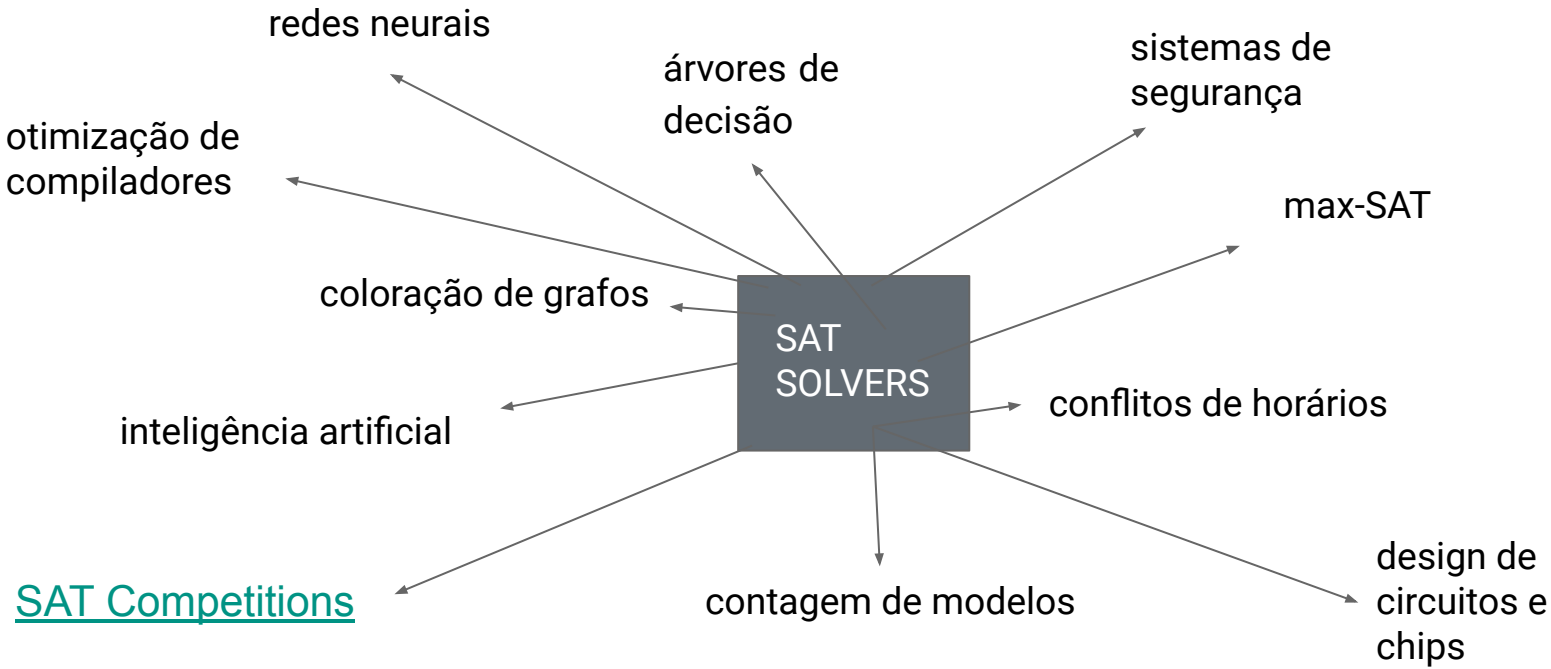
Considere o estado de um DAG logo após explorarmos as consequências de uma marca temporária em um nó de teste.

1. Se esse estado - marcas permanentes mais temporárias - contiver restrições contraditórias, podemos apagar todas as marcas temporárias e marcar permanentemente o nó de teste com a marca dual do seu teste. Ou seja, se marcar o nó *n* com *v* resultou em uma contradição, ele receberá uma marca permanente $\bar{v}$, onde $\bar{T} = F$ e $\bar{\forall} = T$; caso contrário,
2. Se esse estado conseguir marcar todos os nós com restrições consistentes, relatamos essas marcações como uma prova de satisfatibilidade e encerramos o algoritmo.

$$T(p) = p$$
$$T(\phi_1 \wedge \phi_2) = T(\phi_1) \wedge T(\phi_2)$$
$$T(\phi_1 \rightarrow \phi_2) = \neg(T(\phi_1) \wedge \neg T(\phi_2))$$

$$T(\neg\phi) = \neg T(\phi)$$
$$T(\phi_1 \vee \phi_2) = \neg(\neg T(\phi_1) \wedge \neg T(\phi_2))$$

Aplicações:


redes neurais
árvores de decisão
sistemas de segurança
otimização de compiladores
max-SAT
coloração de grafos
SAT SOLVERS
inteligência artificial
conflitos de horários
SAT Competitions
contagem de modelos
design de circuitos e chips

BIBLIOGRAFIA:

M. Huth and M. Ryan. Logic in Computer Science: Modelling and reasoning about systems. Cambridge University Press, 2004.

Licenciatura em Engenharia Informática e de Computadores (LEIC), Técnico Lisboa.“Algoritmos de SAT”. Resumos LEIC-A.

S. Prince. SAT Solvers I: Introduction and applications. Borealis AI, 2020.

M. Finger. SAT Solvers, A Brief Introduction. Instituto de Matemática e Estatística - USP

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. “Introduction to Algorithms” - Third Edition.

F. K. Miyazawa. Instituto de Computação/Unicamp, 2021.