Program verification

Nelma Moreira

Departamento de Ciência de Computadores da FCUP

Decidable first order theories and SMT Solvers
Lecture 19

- SAT consegue codificar operações and relações em inteiros de precisão limitada
  - usando representação como vector de bits
  - representando adição, etc..., como circuitos Booleanos

  Assim como outros tipos de dados and estruturas finitas
- Não consegue codificar tipos não limitados (reais) ou estruturas de dados infinitas (pilhas, listas)
- Aritmética de precisão limitada pouco eficiente para valores grandes
- Existem procedimentos de decisão eficientes para estas teorias que trabalham sobre conjunções
- Usam estratégias de procura eficientes sobre resolutores SAT
- São chamados resolutores SMT (*Satisfiability Modulo Theories*).

- Um conjunto infinito de variáveis $x_1, x_2, \ldots$
- Símbolos lógicos: conectivas Booleanas ($\wedge$, $\vee$, $\implies$, $\neg, \ldots$), quantificadores ($\forall$, $\exists$) and parentisis
- Símbolos não lógicos: alfabeto (ou assinatura) Σ para símbolos de função and de predicados
- Sintaxe: para termos and para fórmulas

$$t ::= x \mid a \mid f(t, \ldots, t)$$

$$\varphi ::= R(t_1, \ldots, t_n) \mid t_1 = t_2 \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \to \varphi) \mid \forall x\varphi \mid \exists x\varphi$$

- Semântica: uma Σ estrutura A com um domínio, interpretação dos símbolos não lógicos and uma atribuição de elementos do domínio às variáveis não quantificadas.
- Uma fórmula φ é *satisfazível* se existe uma estrutura A de Σ em que φ é verdadeira ($A \models \varphi$)
- Uma fórmula é *válida* se para todas as estruturas A de Σ ,φ é verdadeira ($\models \varphi$)

Seja Σ um alfabeto duma linguagem de primeira ordem.

- Uma teoria $\mathcal{T}$ é um conjunto fórmulas fechadas (sem variáveis livres) tal que $\mathcal{T} \models \varphi$ então $\varphi \in \mathcal{T}$.
- Uma $\mathcal{T}$-estrutura é uma Σ estrutura que valida todas as fórmulas de $\mathcal{T}$
- Uma fórmula φ é $\mathcal{T}$-satisfazível se é satisfazível numa $\mathcal{T}$-estrutura; o mesmo para $\mathcal{T}$-válida
- Uma teoria é finitamente (recursivamente) axiomatizável se existe um conjunto finito (recursivo) $\mathcal{A} \subseteq \mathcal{T}$ (axiomas) tal que $\forall \varphi, \varphi \in \mathcal{T}$ sse $\mathcal{A} \models \varphi$
- uma teoria é completa se para toda a fórmula fechada φ, $\mathcal{T} \models \varphi$ ou $\mathcal{T} \models \neg\varphi$.
- Uma teoria $\mathcal{T}$ é decidível se para toda a fórmula é possivel decidir se $\mathcal{T} \models \varphi$.
- Uma teoria axiomatizável e completa é decidivel

- Dada uma Σ estrutura A, $Th(A) = \{\varphi \mid A \models \varphi\}$ é completa
- Essas teorias definidas semanticamente são importantes para podermos raciocinar sobre domínios da matemática (números naturais, inteiros, estruturas algébricas, etc.), mas é importante que sejam axiomatizáveis.
-
- Um fragmento duma teoria $\mathcal{T}$ é um subconjunto de fórmulas de $\mathcal{T}$ com uma restrição sintática:
  - só conjunção de literais ;
  - sem quantificadores (i.e. apenas com todas as variáveis quantificadas universalmente), etc.

1. Teoria da igualdade and funções não interpretadas
2. Teoria da aritmética - Axiomas de Peano
3. Teoria da aritmética de Presburger (só com adição)
4. Teoria dos inteiros
5. Teoria dos reais and racionais
6. Teoria de Conjuntos de Zermelo-Frankle
7. Teoria da Geometria
8. Teoria de Grupos
9. Teoria das expressões (linguagens) regulares (Álgebras de Kleene)

$$\forall x.x = x$$
$$\forall x, y.x = y \rightarrow y = x$$
$$\forall x, y, z.x = y \wedge y = z \rightarrow x = z$$
$$\forall \bar{x}, \bar{y}.x_1 = y_1 \wedge \cdots x_n = y_n \rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$$
$$\forall \bar{x}, \bar{y}.x_1 = y_1 \wedge \cdots x_n = y_n \rightarrow P(x_1, \ldots, x_n) \rightarrow P(y_1, \ldots, y_n)$$

Os dois últimos axiomas são *congruências* ou *consistências*: funcional and de predicados. Indecidível. Fragmento sem quantificadores, decidível. Fragmento só com funções onde conjunções correspondem ao fragmento $\mathcal{EUF}$, decidível.

Any effectively generated (i.e. recursively enumerable) theory capable of expressing elementary arithmetic cannot be both consistent and complete. In particular,for any consistent, effectively generated formal theory that proves certain basic arithmetic truths, there is an arithmetical statement that is true, but not provable in the theory.

Seja $\Sigma = \{0, 1, +, x, =, <\}$. Os axiomas são factos básicos dos números naturais ($\mathbb{N} \models \mathcal{PA}$):

1. $\forall x(x + 1 \neq 0)$
2. $\forall x \forall y(x + 1 = y + 1 \rightarrow x = y)$
3. $0 + 1 = 1$
4. $\forall x\ x + 0 = x$
5. $\forall x \forall y\ x + (y + 1) = (x + y) + 1$
6. $\forall x\ x \times 0 = 0$
7. $\forall x \forall y\ x \times (y + 1) = (x \times y) + x$
8. (princípio da indução) $(Q(0) \wedge (\forall x(Q(x) \rightarrow Q(x + 1)) \rightarrow \forall x Q(x)$

Undecidable (Teorema da Incompletude de Gödel). Even the quantifier-free fragment of TPA is undecidable! (Matiyasevich, 1970)

- Se excluirmos os axiomas 6 and 7 obtemos a aritmética de Presburger $T_{\mathbb{N}}$ que é decidível.
- Esta teoria é muito usada em demonstração automática and relacionada com a teoria dos autómatos.
- A teoria dos inteiros, $T_{\mathbb{Z}}$, $(\Sigma = \{\ldots, -1, 1, 0, 1, 2, \ldots +, \times, =, <\})$ reduz-se à teoria de Presburger.
- Suponhamos

$$\forall w, x. \exists y, z. x + 2y - z - 13 > -3w + 5$$

, podemos introduzir variváveis v_p and v_n (em $\mathbb{N}$) para cada variável v

$$\forall w_p, w_n, x_p, x_n. \exists y_p, y_n, z_p, z_n. (x_p - x_n) + 2(y_p - y_n) - (z_p - z_n) - 13 > -3(w_p - w_n) + 5,$$

mudar de lado os $-$ para

$$\forall w_p, w_n, x_p, x_n. \exists y_p, y_n, z_p, z_n. x_p + 2y_p + z_n + 3w_p > x_n + 2y_n + z_p + 13 + 3w_n + 5$$

e codificar em unário.

Seja $\Sigma = \{0, 1, +, x, =, \geq\}$. A teoria dos reais (ou algebra elementar) é :

- com a adição um grupo abeliano $(\mathbb{R}, +, 0)$, $+$ associative and commutative, 0 identity and all elements have inverse $(-)$.:
- com a multiplicação um anel $(\mathbb{R}, +, \times, 1, 0)$: $\times$ associative and distributes over addition, 1 identidade.
- and a field : $\times$ commutative, $1 \neq 0$, elementos não 0 têm inverso multiplicativo.
- fechado: $\geq$ ordem total

1. $\forall x, y, z. x \geq y \implies x + z \geq y + z$
2. $\forall x, y. x \geq 0 \wedge y \geq 0 \implies xy \geq 0$
3. $\forall x. \exists y. x = y^2 \vee x = -y^2$
4. for each odd integer n , polynomials of odd degree have at least one root.

$$\forall \vec{x}. \exists y. y^n + x_1 y^{n-1} + \cdots + x_{n-1} y + x_n = 0$$

Tarski proved that $\mathcal{T}_{\mathbb{R}}$ was decidable in the 1930s, although the Second World War prevented his publishing the result until 1956. Collins (1975) proposed the more efficient technique of cylindrical algebraic decomposition (CAD) AD runs in time proportionate to $2^{2^k|F|}$, for some constant k and for $|F|$ the length of F..

Tem a assinatura $\Sigma = \{0, 1, +, =, \geq\}$ e corresponde a $\mathcal{T}_{\mathbb{R}}$ sem multiplicação.

1. $\forall x, y. x \geq y \wedge y \geq x \implies x = y$
2. $\forall x, y, z. x \geq y \wedge y \geq z \implies x \geq z$
3. $\forall x, y. x \geq y \vee y \geq x$
4. $\forall x, y, z. (x + y) + z = x + (y + z)$
5. $\forall x. x + 0 = x$
6. $\forall x. x + (-x) = 0$
7. $\forall x, y. x + y = y + x$
8. $\forall x, y, z. x \geq y \implies x + z \geq y + z$
9. for each positive integer n, $\forall x. nx = 0 \implies x = 0$
10. for each positive integer n, $\forall x. \exists y. x = ny$

Models are divisible torsion-free abelian groups. (Axiom 9).

A assinatura é $\Sigma_L = \{cons, head, tail, atom, =\}$.

$$\begin{aligned} & \mathcal{T}_E \\ & \forall x, y.\ head(cons(x, y)) = x \\ & \forall x, y.\ tail(cons(x, y)) = y \\ \forall y.\ \neg atom(y) \implies\ & cons(head(y), tail(y)) = y \\ & \forall x, y \neg atom(cons(x, y)) \end{aligned}$$

- $atom(x)$ é um predicado que é verdade se o argumento x é um singleton.
- Em Lisp *head* é *car* (contents of address register) e *cdr* (contents of decrement register).
- Os axiomas de $\mathcal{T}_E$ garantem que *head* e *tail* são congruências funcionais e *atom* congruência de predicado.
- Satisfazibilidade de fragmento sem quantificadores decidível.
- Podem ser estendidos as outras estruturas de dados recursivas $\mathcal{T}_{RDS}$

A assinatura é $\Sigma_A = \{read, write, =\}$. Os arrays correspondem a funções que podem ser modificadas. Temos que $read(a, i)$, corresponde a $a[i]$, e $write(a, i, v)$, corresponde a $a[i \leftarrow v]$.

$$\mathcal{T}_E$$
$$\forall a, i, j. i = j \rightarrow read(a, i) = read(a, j)$$
$$\forall a, i, j, v. i = j \rightarrow read(write(a, i, v), j) = v$$
$$\forall a, i, j, v. \neg(i = j) \rightarrow read(write(a, i, v), j) = read(a, j)$$
$$\forall a, b. (\forall i. read(a, i) = read(b, i)) \rightarrow a = b$$

Indecidível. Fragmento sem quantificadores, decidível. Sem o último axioma (extensionalidde) ($\mathcal{T}_A$) nem esse fragmento era decidível.

- São específicos a uma determinada teoria
- Determinam se uma determinada fórmula é inconsistente, satisfazível ou válida
- Podem trabalhar sobre conjunções de fórmulas ou determinar se uma fórmula é consequência de outras
- Podem utilizar heurísticas para acelerar procedimentos, mas têm sempre que dar uma resposta correcta and terminar (ou seja têm de ser íntegros and completos)

- Existem muitas teorias úteis decidíveis (ou pelo menos fragmentos):
  - Igualdade com símbolos funcionais não interpretados $\mathcal{EUF}$

$$x = y \wedge f(f(f(x))) = f(x) \rightarrow f(f(f(f(f(y))))) = f(x)$$

  - Updates de funções, registos and tuplos
  - Aritmética linear sobre inteiros and racionais ($\mathcal{LIA}, \mathcal{LRA}$, programação linear, Simplex)

$$x \leq y \wedge x \leq 1 - y \wedge 2x \geq 1 \rightarrow 4x = 2$$

  - Lógica diferencial (caso especial rápido)

$$x - y < c$$

  - Vectores de bits: operações aritméticas módulo
  - Listas e outras estruturas de dados recursivas.
  - Apontadores
- Combinações de teorias decidiveis são em geral também decidiveis

Decidable theories

Theory	**Complexity**
PL	NP-complete
$T_{\mathbb{N}}$, $T_{\mathbb{Z}}$	$\Omega\left(2^{2^n}\right)$, $O\left(2^{2^{2^{kn}}}\right)$
$T_{\mathbb{R}}$	$O\left(2^{2^{kn}}\right)$
$T_{\mathbb{Q}}$	$\Omega\left(2^n\right)$, $O\left(2^{2^{kn}}\right)$
T^{+}_{RDS}	not elementary recursive

Complexities for quantifier-free, conjunctive fragments of theorie

Theory	**Complexity**	**Theory**	**Complexity**
PL	$\Theta(n)$	T_{E}	$O(n \log n)$
$T_{\mathbb{N}}$, $T_{\mathbb{Z}}$	NP-complete	$T_{\mathbb{R}}$	$O\left(2^{2^{kn}}\right)$
$T_{\mathbb{Q}}$	PTIME	T^{+}_{RDS}	$\Theta(n)$
T_{RDS}	$O(n \log n)$	T_{A}	NP-complete

Let

$$x + 2 = y \implies f(read(write(a, x, 3), y - 2) = f(y - x + 1)$$

What theories are involved here?

- equality and uninterpreted functions , $\mathcal{T}_E$
- arrays, $\mathcal{T}_A^=$
- arithmetic, $\mathcal{T}_{\mathbb{Z}}$

Sejam $\mathcal{T}_1$ e $\mathcal{T}_2$ duas teorias com assinaturas Σ_1 eΣ_2; e conjunto de axiomas A_1 e A_2.

The combined theory $\mathcal{T}_1 \cup \mathcal{T}_2$ tal que $\Sigma_1 \cap \Sigma_2 = \{=\}$ is given by:

- assinatura $\Sigma_1 \cup \Sigma_2$
- axiomas: $A_1 \cup A_2$

Nelson & Oppen, 1979

Satisfiability of the quantifier-free fragment of $\mathcal{T}_1 \cup \mathcal{T}_2$ is decidable if

- satisfiability of the quantifier-free fragment of $\mathcal{T}_1$ is decidable
- satisfiability of the quantifier-free fragment of $\mathcal{T}_2$ is decidable
- and certain technical requirements are met

- estendem os resolutores SAT para lógica de primeira ordem
- Utilizam-se procedimentos individuais ou combinados para decidir conjunções de fórmulas com base nas suas teorias decidiveis
- SMT permite estrutura proposicional geral
- Devem explorar estratégias de procura em resolutores SAT modernos
- Os termos são substituídos por variáveis proposicionais
- Encontrar uma solução num resolutor SAT
- Se encontrar, devolve às variáveis a sua interpretação and envia a fórmula para o procedimento de decisão adequado para ela,

Seja prop(φ) uma função que mapeia $\varphi \in \mathcal{T}$ (em CNF and sem quantificadores) numa fórmula proposicional (substituindo fórmulas atómicas por variáveis proposicionais) and unprop a função inversa. Dada uma valorização ρ para prop(φ) seja

$$\varphi(\rho) = \{\text{unprop}(p_i) \mid \rho(p_i) = \top\} \cup \{\neg\text{unprop}(p_i) \mid \rho(p_i) = \bot\}$$

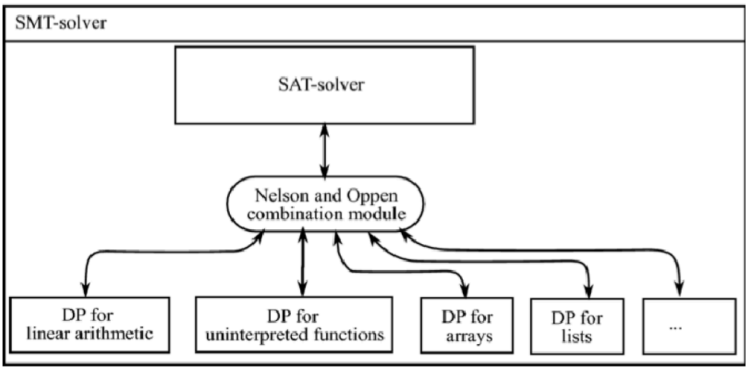
```
SMT-Solver(φ) {
    A := prop(φ)
    loop
        (r, ρ) := SAT(A)
        if r= unsat then return unsat
        (θ, r):=DP_T(φ(ρ))
        if r=sat then return sat
        C := ⋁_{B∈θ} ¬prop(B)
        A := A ∧ C
}
```

onde $\theta \subseteq \varphi(\rho)$ corresponde a fórmulas não satisfaziveis. Então é acrescentado a A o equivalente proposicional de θ (C) para garantir que a atribuição ρ não é de novo escolhida por SAT. DP_T é um algoritmo de decisão para $\mathcal{T}$.

Resolutores SMT eficientes

- Os procedimentos individuais de decisão devem ser rápidos
- Necessitam de uma interacção rápida and eficiente com o resolutor SAT
- Os resolutores SAT devem ser rápidos
- SAT com igualdade integrada para propagação rápida
- Nas últimas decadas houve progressos muito significativos tanto em eficiencia e expressividade
- Actualmente existem muitos resolutores SMT eficientes
- Reconhecidos por competição
- Algums resolutores: Z3, CVC4, Alt-Ergo, Yices 2, MathSAT, Boolector

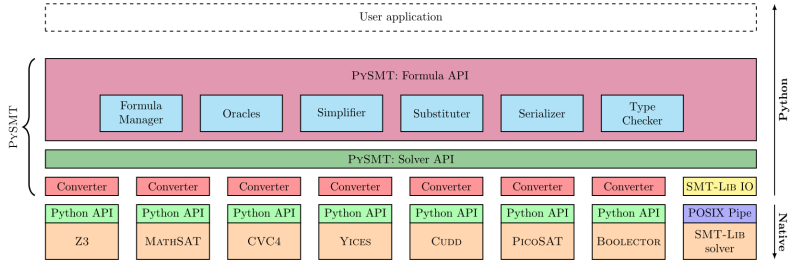
Basic architecture



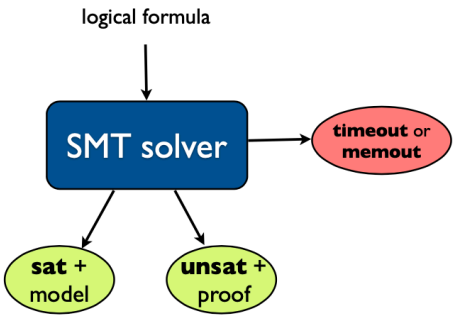
- SMT-LIB: The Satisfiability Modulo Theories Library
  `http://smtlib.cs.uiowa.edu`
- SMT-COMP: The Satisfiability Modulo Theories Competition
  `https://github.com/SMT-COMP`
  `http://www.smtcomp.org`
- Decision procedures - an algorithmic point of view
  `https://www.decision-procedures.org/`
- SAT Association `http://satassociation.org/sat-smt-school.html`
- SAT/SMT Examples `https://sat-smt.codes`

- Catalog of theory declarations - semi-formal specification of theories of interest
  - A theory defines a vocabulary of sorts and functions. The meaning of the theory symbols are specified in the theory declaration.
- Catalog of logic declarations - semi-formal specification of fragments of (combinations of) theories
  - A logic consists of one or more theories, together with some restrictions on the kinds of expressions that may be used within that logic.
- Library of benchmarks
- Utility tools (parsers, converters, ...)
- Useful links (documentation, solvers, ...)
- SMT-LIB language expresses logical statements in a many-sorted first-order logic.

- The pySMT library allows a Python program to communicate with several SMT solvers based on a common language.
- This makes it possible to code a problem independently of the SMT solver, and run the same problem with several SMT solvers.



φ valid $\iff$ $\neg\varphi$ unsatisfiable



Se um resolutor não encontrar solução outro poderá encontrar.

Pretende-se resolver combinações como

$$(x_1 = x_2 \vee x_1 = x_3) \wedge (x_1 = x_2 \vee x_2 = x - 4) \wedge x_1 \neq x_3 \wedge x_1 \neq x_4$$
$$(x_1 + 2x_3 < 5) \vee \neg(x_3 \leq 1) \wedge (x_2 \geq 3)$$
$$(i = j \wedge a[j] = 1) \wedge \neg(a[i] = 1)$$

Vamos considerar teorias decidíveis sem quantificadores, T, para as quais existe um algoritmo de decisão DP_T para o fragmento conjuntivo (conjunção de fórmulas atómicas).

- Corresponde à teoria da igualdade $\mathcal{T}_E$ só com variáveis (e constantes que podem ser eliminadas) and sem quantificadores

$$\varphi := \varphi \wedge \varphi \mid (\varphi) \mid \neg\varphi \mid t = t$$
$$t := x|c$$

- tem a mesma expressividade and complexidade da lógica proposicional

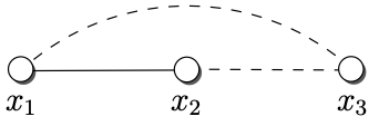
Exerc.

Descreve um algoritmo para eliminar constantes duma fórmula com igualdades. $\diamond$

Algoritmo de decisão para a lógica da igualdade equacional (conjunções), DP_T

- Seja φ uma conjunção de igualdades and desigualdades
- Construir um grafo $G = (N, E_=, E_\neq)$ onde
- N são as variváveis de φ,
- $E_=$, arestas (x_i, x_j) correspondendo a igualdades $x_i = x_j \in \varphi$ (representam-se a tracejado)
- $E_\neq$, arestas (x_i, x_j) correspondendo a desigualdades $x_i \neq x_j \in \varphi$ (representam-se com traço a cheio)
- φ não é satisfazível se and só se existe uma aresta $(v_1, v_2) \in E_\neq$ tal que v_2 é atingível de v_1 por arestas de $E_=$.

Para $x_2 = x_3 \wedge x_1 = x_3 \wedge x_1 \neq x_2$, concluímos que não é satisfazível



Existem duas abordagens para a combinação Booleana de fórmulas atómicas duma teoria

- *eager*
  - tradução numa fórmula proposisional equisatisfazível
  - que é resolvida por um resolutor SAT
- *lazy*
  - codificar a fórmula numa fórmula proposicional
  - utilizar um resoutor SAT baseado em DPLL
  - usar um procedimento de decisão para a teoria (DP_T) para refinar a fórmula e guiar o resolutor SAT
- a abordagem lazy parece funcionar melhor

Em particular, no caso da fórmula φ conter outras conectivas o melhor é integrar D_T num resolutor SAT.

- Supor φ em forma normal negativa (NNF)
- $at(\varphi)$ conjunto de fórmulas atómicas sobre Σ; $at_i(\varphi)$ i-ésima fórmula atómica
- A cada fórmula atómica $a \in at(\varphi)$ associar $e(a)$ uma variável proposicional, chamada o *codificador*
- A função de *codificação* e estende-se a φ, sendo $e(\varphi)$ a fórmula proposicional que resulta de substituir cada fórmula atómica pela sua variável proposicional.
- Por exemplo se $\varphi := (x = y \vee x = z)$ então $e(\varphi) := e(x = y) \vee e(x = z)$

Seja

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

Temos

$$e(\varphi) := e(x = y) \wedge ((e(y = z) \wedge \neg(e(x = z))) \vee e(x = z)) := \mathcal{B}$$

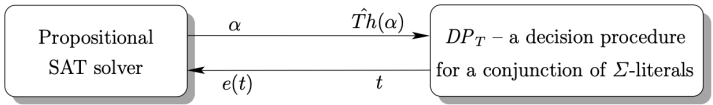
Usando um resolutor SAT obtêm-se uma valorização para $\mathcal{B}$:

$$\alpha := \{e(x = y) \rightarrow \text{true}, e(y = z) \rightarrow \text{true}, e(x = z) \rightarrow \text{false}\}$$

O algoritmo DP_T verifica se a conjunção de literais que corresponde a α é satisfeita , i. e.

$$\hat{Th}(\alpha) = (x = y) \wedge (y = z) \wedge x \neq z$$

Neste caso não é, logo $\neg\hat{Th}(\alpha)$ é uma tautologia and é o valor retornado por DP_T. Podemos, fazer a conjunção de $e(\neg\hat{Th}(\alpha)) \wedge \mathcal{B}$ and voltar a chamar o resolutor de SAT de modo que a mesma valorização α não possa ser retornada dado que não satisfazia $e(\neg\hat{Th}(\alpha))$ (*bloqueador*).



Seja a nova atribuição

$$\alpha' := \{e(x = y) \to \mathsf{true}, e(y = z) \to \mathsf{true}, e(x = z) \to \mathsf{true}\}$$

que corresponde a

$$\hat{Th}(\alpha') := (x = y) \wedge (y = z) \wedge x = z$$

que é satisfazível, o que prova que a fórmula original φ é satisfazível.

Formalmente dada uma codificação $e(\varphi)$ and uma valorização α, para cada codificador $e(at_i)$ temos

$$Th(at_i, \alpha) = \begin{cases} at_i & \alpha(e(at_i)) = \text{true} \\ \neg at_i & \alpha(e(at_i)) = \text{false} \end{cases}$$

e sendo o conjunto de literais

$$Th(\alpha) = \{ Th(at_i, \alpha) \mid at_i \in \varphi \}$$

então $\hat{Th}(\alpha)$ é a conjunção dos literais em $Th(\alpha)$.

Seja DEDUCTION o algoritmo DP_T com possível a geração de um bloqueador, $t = \neg \hat{Th}(\alpha)$.

Algorithm 3.3.1: LAZY-BASIC

Input: A formula φ
Output: "Satisfiable" if φ is satisfiable, and "Unsatisfiable" otherwise

1. **function** LAZY-BASIC(φ)
2. $\quad \mathcal{B} := e(\varphi)$;
3. $\quad$ **while** (TRUE) **do**
4. $\quad\quad \langle \alpha, res \rangle :=$ SAT-SOLVER($\mathcal{B}$);
5. $\quad\quad$ **if** $res =$"Unsatisfiable" **then return** "Unsatisfiable";
6. $\quad\quad$ **else**
7. $\quad\quad\quad \langle t, res \rangle :=$ DEDUCTION($\hat{Th}(\alpha)$);
8. $\quad\quad\quad$ **if** $res =$"Satisfiable" **then return** "Satisfiable";
9. $\quad\quad\quad \mathcal{B} := \mathcal{B} \wedge e(t)$;

Consider the following three requirements on the formula t that is returned by Deduction:

1. t is valid in $\mathcal{T}$.
2. The atoms in t are restricted to those appearing in φ
3. The encoding of t contradicts α, i.e., $e(t)$is a blocking clause

Consider the following three requirements on the formula t that is returned by Deduction:

1. t is valid in $\mathcal{T}$.
2. The atoms in t are restricted to those appearing in φ
3. The encoding of t contradicts α, i.e., $e(t)$is a blocking clause

the first requirement 1. guarantes soundness. The second and third requirements 2. e 3.

are sufficient to guaranteeing termination.

Consider the following three requirements on the formula t that is returned by Deduction:

1. t is valid in $\mathcal{T}$.
2. The atoms in t are restricted to those appearing in φ
3. The encoding of t contradicts α, i.e., $e(t)$is a blocking clause

the first requirement 1. guarantes soundness. The second and third requirements 2. e 3.

are sufficient to guaranteeing termination.

Dois podem ser enfraquecidos:

- Basta que t implique φ
- Em t podem ocorrer outras fórmulas atómicas

Consider the following three requirements on the formula t that is returned by Deduction:

1. t is valid in $\mathcal{T}$.
2. The atoms in t are restricted to those appearing in φ
3. The encoding of t contradicts α, i.e., $e(t)$is a blocking clause

the first requirement 1. guarantes soundness. The second and third requirements 2. e 3.

are sufficient to guaranteeing termination.

Dois podem ser enfraquecidos:

- Basta que t implique φ
- Em t podem ocorrer outras fórmulas atómicas

Para além de considerar um SAT incremental, é mais eficiente integrar o algoritmo DEDUCTION num algoritmo CDCL.

Algorithm 3.3.2: LAZY-CDCL

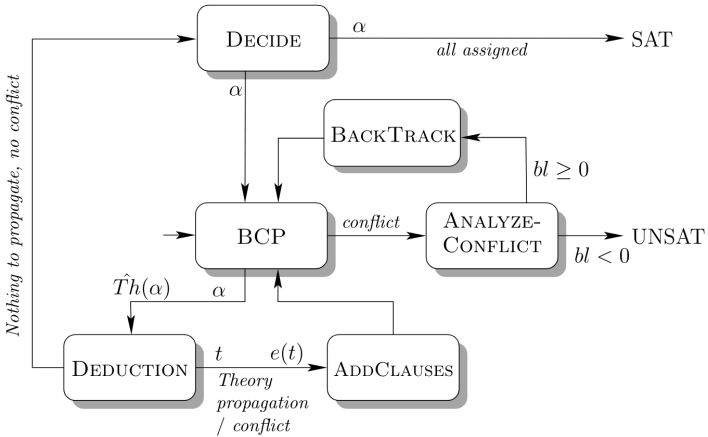
Input: A formula φ
Output: "Satisfiable" if the formula is satisfiable, and "Unsatisfiable" otherwise

```
1.  function LAZY-CDCL
2.      ADDCLAUSES(cnf(e(φ)));
3.      while (TRUE) do
4.          while (BCP() = "conflict") do
5.              backtrack-level := ANALYZE-CONFLICT();
6.              if backtrack-level < 0 then return "Unsatisfiable";
7.              else BackTrack(backtrack-level);
8.          if ¬DECIDE() then                          ▷ Full assignment
9.              ⟨t, res⟩:=DEDUCTION(T̂h(α));            ▷ α is the assignment
10.             if res="Satisfiable" then return "Satisfiable";
11.             ADDCLAUSES(e(t));
```

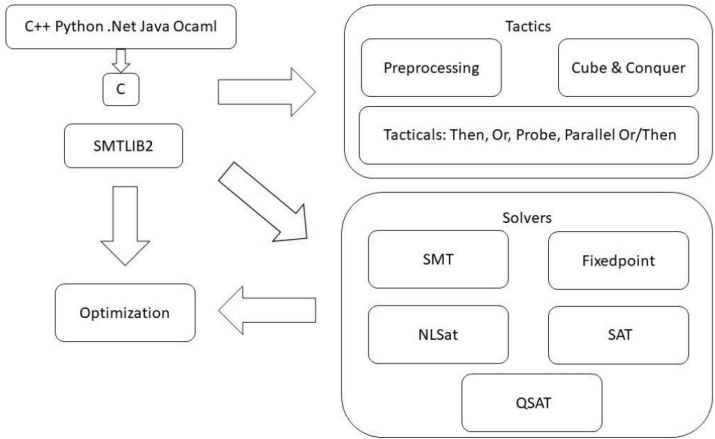
This algorithm uses a procedure ADDCLAUSES, which adds new clauses to the current set of clauses at run time.

Suponhamos que φ tem uma variável inteira x_1 e as fórmulas atómicas $x_1 < 0$ e $x_1 > 10$. Se $e(x_1 > 10) \leftarrow$ true e $e(x_1 < 0) \leftarrow$ true haverá uma contradição mas que é é detectada depois de ser obtida uma valorização completa. Contudo isso pode ser melhorado, se a chamada a DEDUCTION se fizer antes. Isso permite

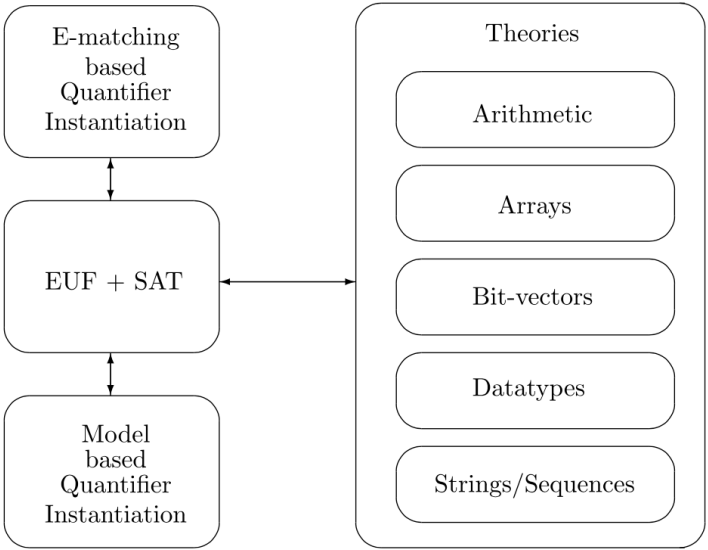
- Contradizer valorizações parciais mais cedo
- Implicações de literais ainda não atribuídos podem ser comunicados ao resolutor SAT. A esta técnica chama-se propagação da teoria.
- Por exemplo, se $e(x_1 > 10) \leftarrow$ true poderemos inferir que $e(x_1 < 0) \leftarrow$ false e assim evitar o conflito.


DECIDE
α
all assigned
SAT
α
BACKTRACK
bl ≥ 0
BCP
conflict
ANALYZE-CONFLICT
UNSAT
bl < 0
T̂h(α)
α
Nothing to propagate, no conflict
DEDUCTION
t
e(t)
Theory propagation / conflict
ADDCLAUSES

- Z3 `https://github.com/Z3Prover/z3`
- Z3 `https://z3prover.github.io/papers/programmingz3.html`
- `https://z3prover.github.io/papers/z3internals.html`
- Python : pip install z3-solver
- Tutorial:
  `https://ericpony.github.io/z3py-tutorial/guide-examples.htm`


C++ Python .Net Java Ocaml
C
SMTLIB2
Optimization
Tactics
Preprocessing
Cube & Conquer
Tacticals: Then, Or, Probe, Parallel Or/Then
Solvers
SMT
Fixedpoint
NLSat
SAT
QSAT

Z3 Arquitectura d(um) resolutor SMT



```
x = Real('x')
y = Real('y')
z = Real('z')
s = Solver()
s.add(3*x + 2*y - z == 1)
s.add(2*x - 2*y + 4*z == -2)
s.add(-x + 0.5*y - z == 0)
print(s.check())
print(s.model())
```

- Variáveis lógicas são criadas indicando qual o tipo (`Sort`): `Real`, `Bool`, `Int`, ou de um tipo novo declarado:

```
S = DeclareSort('S')
f = Function('f', S, S)
x = Const('x', S)
y = Const('y', S)
z  = Const('z', S)
s = Solver()
s.add(Or(x!=y,Or(f(x)==f(y),f(x)!=f(z))))
print(s.check())
print(s.model())
solve(Or(x!=y,Or(f(x)==f(y),f(x)!=f(z)))
```

- `solve()` cria um `Solver`, adiciona a fórmula e verifica se é satisfazível retornando uma solução (`model`).
- `Const` e `Function` definem funções de zero ou mais variáveis, respectivamente

- a linguagem standard para escrever programas para resolutores SMT é o SMT-LIB (parecida com o LISP), mas aqui usamos apenas o interface em Python

```
x, y = Ints('x y')
s = Solver()
s.add((x % 4) + 3 * (y / 2) > x - y)
print(s.sexpr())
```

- produz

```
(declare-fun y () Int)
(declare-fun x () Int)
(assert (> (+ (mod x 4) (* 3 (div y 2))) (- x y)))
```

- Quantificadores: `ForAll`, `Exists`

```
solve([y == x + 1, ForAll([y], Implies(y <= 0, x < y))])
```

A primeira ocorrência de y é livre, a segunda ligada.

```
(set-logic QF UFLIA)
(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun z () Int)
(assert (distinct x y z))
(assert (> (+ x y) (* 2 z)))
(assert (>= x 0))
(assert (>= y 0))
(assert (>= z 0))
(check-sat)
(get-model)
(get-value (x y z))
```

Usando % z3 exemplo1.smt2

```
sat
(
  (define-fun x () Int
    3)
  (define-fun z () Int
    1)
  (define-fun y () Int
    0)
)
((x 3)
 (y 0)
 (z 1))
```

pyz3: s.from_file("exemplo1.smt2")

- help(class) or help(function)
- *describe_tactics*.
-


Nikolai Bjorner and Leonardo de Moura.
Z3 Theorem Prover.
Rise, Microsft, 2015.


Armin Biere, Marjin Heulen, Hans van Maaren, and Tobis Walsh.
Handbook of Satisfiability.
IOS Press, second edition, 2021.


Aaron R. Bradley and Zohar Manna.
The Calculus of Computation: Decision Procedures with Applications to Verification.
Springer Verlag, 2007.


Daniel Kroening and Ofer Strichman.
Decision Procedures:An Algorithmic Point of View.
Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016.