Especificação e Prova

Podemos classificar os métodos de prova em três categorias:

- Provas não assistidas computacionalmente: as demonstrações são efectuadas à mão e podem ser descritas informalmente.
- Ferramentas que introduzem um sistema formal para formular demonstrações: o uso de linguagem natural não é permitido.
- Ferramentas de prova computacionais.

Lógicas:

- lógica proposicional, lógica de primeira ordem, lógica de ordem superior
- lógica clássica versus lógica intuicionistica
- lógicas temporais

Ferramentas de prova

- Sistemas de demonstração automática: dependem da decidibilidade de fragmentos da lógica que lhes serve de base
  - ELAN: reescrita de primeira ordem
  - ACS2: lógica de primeira ordem
  - Resolutores SMT (Satisfability Module Theory): Yices, CVC3, Z3, Alt-Ergo, Simplify: algoritmos decisão para inteiros, reais, "arrays", etc.
  - Ao contrário de model-checkers permitem racionalizar acerca de conjuntos infinitos
- Sistemas de demonstração assistida: consideram lógicas mais expressivas e potencialmente não-decidíveis (Coq, Mizar, Isabelle, Agda, Matita, etc)
  - Combinam duas capacidades: verificação de provas; construção assistida de provas
  - Na maior parte dos assistentes de prova as demonstrações são construídas interactivamente aplicando um conjunto de tácticas: case, elim, change, rewrite, simpl, discriminate, injection, induction.

Anotação de Programas

- Formalismos baseados em Lógica de Hoare
- Desenvolvimento de software baseado na noção de contracto

  - Linguagens como SPEC# (extende o C#) ou SPARK (um subconjunto do ADA para o desenvolvimento de software critico);
  - No contexto da linguagem Java: sistemas de anotações baseados na linguagem de anotações JML tais como Esc/Java, KeY e Krakatoa
  - No contexto da linguagem C: Frama-C (baseado na linguagem de anotações ACSL), ou VCC (permite a verificação de aspectos concorrentes de programas em C)
  - etc...

Demonstradores automáticos de teoremas baseados em SAT-DPLL

- Um resolutor SAT determina se uma dada fórmula $\mathcal{B}$ é satisfazível, devolvendo nesse caso uma atribuição de variáveis que satifaz a fórmula.
- Os resolutores SAT modernos conseguem resolver fórmulas em FNC (forma normal conjuntiva, conjunto de claúsulas) com centenas de milhares ou milhões de variáveis em tempo útil.
- Existem instâncias mais pequenas para as quais não é possível obter soluções
- É difícil prever sem tentar quais as instâncias difíceis de resolver

Demonstradores automáticos de teoremas baseados em SAT-DPLL

- Agrupados em duas categorias:
  - Baseados na plataforma *Davis-Putman-Loveland-Logemann* (DPLL): as ferramentas atravessam e retrocessem numa árvore binária cujos nós representam atribuições parciais a e as folhas atribuições totais
  - Baseados em procura estocástica: os resolutores "adivinham" uma atribuição total e no caso da avaliação da fórmula retornar falso, vai trocando alguns dos valores seguindo uma heuristica (normalmente conta o número de cláusulas não satisfazíveis e escolhe a troca de variáveis que minimisa esse número)
- Os resolutores DPLL são considerados melhores na maioria dos casos.
- Outra vantagem dos resolutores DPLL é a sua completude.

Algoritmo de Davis-Putman

Versão inicial de 1960 que ainda é a base de muitos dos algoritmos mais eficientes actualmente...ver `http://www.satlive.org/index.jsp` ou Concurso `http://www.satcompetition.org/`.

A ideia base do algoritmo é considerar para cada variável os possíveis valores de verdade e simplificar a fórmula de acordo com essas atribuições até se poder concluir que ela é ou não satisfazível.

Algoritmo de Davis-Putman

Propagação Unitária

Seja S um conjunto de cláusulas. Um conjunto S' é obtido de S por propagação unitária se S' se obtém de S por repetição da seguinte transformação: se S contém uma cláusula unitária l então

1. remover de S todas as cláusulas da forma $l \vee C'$
2. substituir em S cada cláusula da forma $\tilde{l} \vee C'$ pela cláusula C'.

Algoritmo de Davis-Putman

```
DLL(S) {
   input: conjunto de clausulas S
   output: satisfazivel ou nao satisfazivel
   S := propaga(S)
   if S vazio then return satisfazivel
   if S contem F then return nao satisfazivel
   l := selecciona_literal(S)
   if DLL(S ∪ {l}) = satisfazivel
      then return satisfazivel
      else return DLL(S U {l̃})
}
```

Aplicação do Algoritmo

$$S = \{\neg p \vee \neg q, \neg p \vee q, p \vee \neg q, p \vee q\} \tag{1}$$

Como não tem cláusulas unitárias, temos de selecionar um literal, por exemplo, $\neg p$.

Por propagação:

$$\begin{aligned}\{\neg p \vee \neg q, \neg p \vee q, p \vee \neg q, p \vee q, \neg p\} \Rightarrow \\ \{\neg q, q\} \Rightarrow \{\mathrm{F}\}\end{aligned} \tag{2}$$

Aplicação do Algoritmo

Temos ainda de considerar $S \cup \{p\}$. Neste caso a propagação é:

$$\begin{aligned}\{\neg p \vee \neg q, \neg p \vee q, p \vee \neg q, p \vee q, p\} \Rightarrow \\ \{\neg q, q\} \Rightarrow \{\mathrm{F}\}\end{aligned} \tag{3}$$

E o algoritmo retorna nao satisfazivel .

Algoritmo DPLL-SAT (actual)

Input: Uma fórmula proposicional $\mathcal{B}$ em FNC
Output: Satisfazível ou Não-satisfazível

```
function DPLL
   if BCP() = "conflito" then return "Não-satisfazível";
   while (VERDADE) do
     if not DECIDE() then return "Satisfazível";
     else
        while (BCP() = "conflito") do
          nivel-retrocesso := ANALISA-CONFLICTO();
          if nivel-retrocesso<0 then return "Não-satisfazível";
          else RETROCEDE(nivel-retrocesso);
```

Algoritmo DPLL-SAT - Componentes

Nome:	DECIDE()
Output:	FALSO se e só se não existem variáveis para atribuir
Descrição:	Escolhe uma variável não atibuída e um valor de verdade

Nome:	BCP()
Output:	"conflito" se e só se é encontrado um conflito
Descrição:	Propagação: Repete a regra da cláusula unitária até haver um conflito ou não haverem mais implicações.

Algoritmo DPLL-SAT - Componentes

Nível de decisão: nível 0 claúsulas unárias; nível $i+1$, se decide o valor duma variável no nível i.

Nome:	ANALISA-CONFLITO()
Output:	Menos 1 se é detectado um conflito no nível 0 (a fórmula é não-satisfazível). Caso contrário, é indicado o nível de decisão para o qual o resolutor deve retroceder

Nome:	RETROCEDE(*nd*)
Descrição:	O nível de decisão passa a ser *nd* e apaga todas as atribuições em níveis superiores a *nd*

Resolutores SMT

- SAT consegue codificar operações e relações em inteiros de precisão limitada
  - usando representação como vector de bits

  - representando adição, etc..., como circuitos booleanos

  Assim como outros tipos de dados e 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

Chamados de resolutores SMT *Satisfability modulo theories*.

Teorias de Lógica de Primeira Ordem

Uma teoria $\mathcal{T}$ é um conjunto fórmulas fechadas. Uma teoria é finitamente axiomatizável se existe um conjunto finito $\mathcal{A} \subseteq \mathcal{T}$ (axiomas) tal que $\forall \phi, \phi \in \mathcal{T}$ sse $\phi \vdash \mathcal{A}$.

1. Teoria da igualdade e funções não interpretadas
2. Teoria da aritmética - Axiomas de Peano
3. Teoria de Conjuntos de Zermelo-Frankle
4. Teoria da Geometria
5. Teoria de inteiros
6. Teoria de Grupos

Axiomas de Peano

Os axiomas são factos básicos dos números naturais:

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) $(Q(0) \; \wedge \; (\forall x (Q(x) \rightarrow Q(x + 1)) \rightarrow \forall x Q(x)$

Não é decidível (Teorema da Incompletitude de Gödel)

Teoria da igualdade $\mathcal{T}_E$

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

Indecidível. Fragmento sem quantificadores, decidível

Teoria dos *Arrays*, $\mathcal{T}_A$

$read(a, i)$ and $write(a, i, v)$

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

Indecível. Fragmento sem quantificadores, decidível.

Procedimentos de decisão

- São especifícos 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 acelarar procedimentos, mas têm sempre que dar uma resposta correcta e terminar (ou seja têm de ser íntegros e completos)

Teorias decidíveis

- Existem muitas teorias úteis decidíveis (ou pelo menos fragmentos):

  - Igualdade com símbolos funcionais não interpretados

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

  - Updates de funções, registos e tuplos
  - Aritmética linear sobre inteiros e racionais

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

  - Lógica diferencial (caso especial rápido)

$$x - y < c$$

  - Vectores de bits: operações aritméticas módulo
  - Listas outras estruturas de dados recursivas.

- Combinações de teorias decidíceis são em geral também decidíceis

Resolutores SMT

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

Resolutores SMT

Seja $\mathsf{prop}(\phi)$ uma função que mapeia ϕ numa fórmula proposicional (substituinto fórmulas atómicas por variávies proposicionais) e unprop a função inversa. Dada uma valorização ρ para $\mathsf{prop}(\phi)$ seja

$$\Phi(\rho) = \{\mathsf{unprop}(p_i) \mid \rho(p_i) = \top\} \cup \{\neg\mathsf{unprop}(p_i) \mid \rho(p_i) = \bot\}$$

```
SMT-Solver(φ) {
    A := prop(φ)
    loop
        (r, ρ) := SAT(A)
        if r= unsat then return unsat
```

```
    (r, θ):=TSolver(Φ(ρ))
    if r=sat then return sat
    C := ⋁_{B∈θ} ¬prop(B)
    A := A ∧ C
}
```

onde:

- `TSolver` é um resolutor para conjunções de fórmulas atómicas de diversas teorias
- θ um subconjunto não satisfazível de $\Phi(\rho)$ (se existir)

Resolutores SMT eficiente

- Os procedimentos individuais de decisão devem ser rápidos
- Necessitam de uma interacção rápida e eficiente com o resolutor SAT
- Os resolutores SAT devem ser rápidos
- SAT com igualdade integrada para propagação rápida
- Existem muitos resolutores SMT eficientes
- Reconhecidos por competição

Resolutores SMT - Links

- SMT-LIB: The Satisfiability Modulo Theories Library

  `http://smtlib.org/`

- SMT-COMP: The Satisfiability Modulo Theories Competition

  `http://www.smtcomp.org/2011/`

- Decision procedures - an algorithmic point of view

  `http://www.decision-procedures.org/`

- Summer School on SAT/SMT resolutores e as suas aplicações

  `http://people.csail.mit.edu/vganesh/summerschool/`