软件分析

可满足性模理论
Satisfiability Modulo Theories

熊英飞
北京大学

从SAT到SMT

- SAT问题回答某个命题逻辑公式的可满足性，如：
  - $A \wedge B \vee \neg C$
- 但实际中的公式却往往是这样的：
  - $a + b < c \wedge f(b) > c \vee c > 0$
- 如何判断这样公式的可满足性?

- 从逻辑学角度来看，$a + b < c$或者$f(b) > c$都是逻辑系统中不直接处理的符号，需要公理才能推理
- 理论(Theory)：针对一个领域的一组公理
- 可满足性模理论Satisfiability Modulo Theories：
  - 给定一组理论，根据给定逻辑，求在该组理论解释下公式的可满足性
  - 现有理论通常针对一阶理论，即公理都是一阶的

常见理论举例：EUF

- Equality with Uninterpreted Functions
- 公理:
  - $a_i = b_i \Longrightarrow f(a_1 \dots a_n) = f(b_1 \dots b_n)$
  - $a = b \Leftrightarrow \neg(a \neq b)$
  - 等号的自反、对称和传递性
- 如：$a * \big(f(b) + f(c)\big) = d \wedge$
  $b * \big(f(a) + f(c)\big) \neq d \wedge a = b$
  - f, *和+都看做是未定义的函数
- 可直接推出矛盾

常见理论举例

- 算术
  - a+10<b
  - 2x+3y+4z=10
- 数组
  - read(write(a, i, v), i)=v
- 位向量Bit Vectors
  - $a[0] = b[1] \land a = c \land b[1] \neq c[0]$

SMT历史

- 70、80年代：出现了基本算法混合不同理论，但求解能力有限
- 2000年前后：SAT速度大幅提升，转为以SAT为中心的方法
  - 1999-：Eager方法，将SMT问题编码成SAT问题
  - 2000-：Lazy方法，交互调用SAT求解器和各种专用求解器
    - 中科院软件所张健老师发表了最早的Lazy方法论文之一，开发了BoNuS工具
      - [ZhangJ 2000] Jian Zhang. Specification Analysis and Test Data Generation by Solving Boolean Combinations of Numeric Constraints. Proc. APAQS 2000, pp.267-274.
    - 限于当时国内的学术影响力和BoNuS后续发展情况，国外的文献目前更多引用如下几个系统作为Lazy方法的开创论文
      - TSAT[2000], CVC[2002], MathSAT[2002]
      - 其中MathSAT是作者听了张健老师报告之后开发

Eager方法

- 将SMT问题编码成SAT问题
- 例：将EUF编码成SAT
  - $f(a)=c \land f(b)\neq c \land a\neq b$
- 引入符号替代函数调用
  - A替代$f(a)$，B替代$f(b)$
  - 原式变为
    - $A=c \land B\neq c \land a\neq b$
  - 同时根据公理添加约束
    - $a=b \to A=B$
- 引入布尔变量替代等式
  - $P_{A=c} \land \neg P_{B=c} \land \neg P_{a=b}$
  - $P_{a=b} \to P_{A=B}$
  - 编码方式蕴含了第二条公理和对称性
- 根据自反性添加约束
  - 如$P_{A=A}$，本例中不需要
- 同时为传递性添加约束
  - $P_{A=c} \land P_{B=c} \to P_{A=B}$
  - $P_{A=B} \land P_{B=c} \to P_{A=c}$
  - $P_{A=B} \land P_{\mathrm{A}=c} \to P_{B=c}$
  - ……

Eager方法的问题

- 很多理论存在专门的求解算法，如
  - EUF可以用一个不动点算法不断合并等价类求解
  - 线性方程组存在专门算法求解
- 编码成SAT之后，SAT求解器无法利用这些算法

- 模块化程度不高
  - 每种理论都要设计单独的编码方法
  - 不同理论混合使用时要保证编码方法兼容

- 并不总是能编码为SAT

Lazy方法

- 黑盒混合SAT求解器和各种理论求解器
- 理论求解器：
  - 输入：属于特定理论的公式组，组内公式属于合取关系
    - EUF公式组：
      - $f(a) = c$
      - $f(b) \neq c$
      - $a \neq b$
    - 线性方程组：
      - a+b=10
      - a-b=4
  - 输出：SAT或者UNSAT

Lazy方法示例

$$\underbrace{g(a) = c}_{1} \wedge \left(\underbrace{f\big(g(a)\big) \neq f(c)}_{-2} \vee \underbrace{g(a) = d}_{3}\right) \wedge \underbrace{c \neq d}_{-4}$$

- 生成如下公式到SAT求解器
  - {1}, {-2, 3}, {-4}
- SAT求解器返回SAT和赋值{1, -2, -4}
- 生成如下公式组到EUF求解器
  - $g(a) = c$
  - $f\big(g(a)\big) \neq f(c)$
  - $c \neq d$
- EUF求解器返回UNSAT
- 生成如下公式到SAT求解器：{1}, {-2, 3}, {-4}, {-1, 2, 4}
- SAT求解器返回SAT和赋值{1, 2, 3, -4}
- EUF求解器返回UNSAT
- SAT求解器发现{1}, {-2, 3}, {-4}, {-1, 2, 4}, {-1, -2, -3, 4}不可满足

Lazy方法优点

- 同时利用SAT求解器和理论求解器的优势
- 模块化
  - 新的理论只需要实现公共接口就可以集成到SMT求解器中

- 目前主流SMT求解器中普遍采用Lazy方法

Lazy方法问题

- 考虑如下公式:

$$\underbrace{a = b}_{1} \wedge \left(\underbrace{f\big(g(a)\big) \neq f(c)}_{-2} \vee \underbrace{g(a) = d}_{3}\right) \wedge \underbrace{b \neq a}_{-4}$$

SAT	EUF
{1, -2, 3, -4}	UNSAT
{1, -2, -3, -4}	UNSAT
{1, 2, 3, -4}	UNSAT
UNSAT	

- 事实上，只要存在1和-4，该公式就不可能被满足
- 但EUF求解器无法将这一信息告诉SAT求解器
- 如何将定理信息传给SAT求解器？

复习：CDCL算法

```
cdcl() {
 assign=空赋值;
 while (true) {
  赋值推导(assign);
  if (推导结果有冲突) {
   if (assign为空) return false;
   添加新约束;
   撤销赋值;
  } else {
   if (推导结果是完整的) return true;
   选择一个未尝试的赋值x=1或者x=0;
   添加该赋值到assign;
}}}
```

- 红色部分是CDCL区别于穷举之处
- 能否加上理论指引？

给理论求解器添加接口函数

- propagate
  - 输入：
    - 属于当前理论的有限公式集合
    - 已知为真或为假的公式
  - 输出：新推出的公式和其前提条件

- 例如：
  - 输入：
    - 所有公式：$a = b$，$f(a) = f(b)$
    - 已知公式：$a = b$
  - 输出：
    - $a = b \Rightarrow f(a) = f(b)$

给理论求解器添加接口函数

- get_unsatisfiable_core
  - 输入：一组公式，已知冲突
  - 输出：该公式（尽可能小的）子集，仍然冲突

- 例如:
  - 输入:
    - $a = b,\ f(a) \neq f(b),\ b = c$
  - 输出:
    - $a = b,\ f(a) \neq f(b)$

DPLL(T)算法

打破SAT黑盒，以CDCL算法为中心集成理论求解器

```
dpll_t() {
 assign=空赋值;
 while (true) {
 if (!赋值推导和冲突检查(assign)) {
    if (assign为空) return false;
    添加新约束();
    撤销赋值;
  } else {
    if (推导结果是完整的) return true;
    选择一个未尝试的赋值x=1或者0;
    添加该赋值到assign;
}}}
```

```
赋值推导和冲突检查(assign) {
  do {
    命题逻辑推导(assign);
    if(推导发现冲突) return false;
    if(T求解器发现不可满足) return false;
    用T求解器推导(assign);
    if(推导发现冲突) return false;
  } while(推导出新赋值)
  return true;
}
添加新约束() {
  if(推导发现冲突) 矛盾集=冲突项的前驱;
  else 矛盾集=T求解器.get_unsatisfiable_core();
  添加约束(矛盾集取反);
}
```

DPLL(T)例子1

$$\underbrace{a = b}_{1} \wedge \left(\underbrace{f(g(a)) \neq f(c)}_{-2} \vee \underbrace{g(a) = d}_{3}\right) \wedge \underbrace{b \neq a}_{-4}$$

赋值	推导出的赋值	推导
{}	{1}	Unit Propagation
{}	{1, 4}	T-Propagation
{}	{1, 4, -4}	Unit Propgation
	矛盾	

DPLL(T)例子2

$$\underbrace{g(a) = c}_{1} \land \left(\underbrace{f\left(g(a)\right) \neq f(c)}_{-2} \lor \underbrace{g(a) = d}_{3}\right) \land (\underbrace{c \neq d}_{-4} \lor \underbrace{d = e}_{5})$$

赋值	推导出的赋值	推导
{}	{1}	Unit Propagation
{}	{1, 2}	T-Propagation
{}	{1, 2, 3}	Unit Propgation
{-4}	添加约束{-1, -3, 4}，撤销赋值	T求解器返回UNSAT，矛盾集{1, 3, -4}
{}	{1, 2, 3, 4, 5}	Unit Propagation & T-Propagation

DPLL(T)特点

- 理论求解器指导SAT搜索，效率有大幅提高
- 依然模块化
  - 理论求解器只需要多实现两个方法
  - 甚至不实现也可以，最多可能损失效率
    - propagate默认直接返回空
    - get_unsatisfiable_core默认直接返回原公式集合

混合多个理论

- DPLL(T)算法可以处理混合的多个理论，前提是不同理论的公式之间没有共享变量
  - $f(a) = f(b) \land a = b \land x + 1 = y - 1$
  - 简单对不同部分调用不同的理论求解器即可
- 但不能处理混合的情况
  - $f(a) \neq f(b) \land a + 1 = 2 + b - 1$
  - $f(a + 1) = f(1 + a) - 1$

- 如何混合多个理论形成单一的理论求解器?

手动推导

部分命题左边两边都能理解，称为接口属性

EUF

$$f(a+1) = f(1+a)$$

算术

$$\Downarrow$$

$$a + 1 = 1 + a$$

$$f(a+1) \neq f(1+a) - 1$$

解决方案

- 通过变形让不同理论位于不同的文字

$$f(a+1) = f(1+a) - 1$$

$$\Downarrow$$

$$e_1 = a + 1 \qquad e_3 = f(e_1)$$
$$e_2 = 1 + a \qquad e_4 = f(e_2)$$
$$e_3 = e_4 - 1$$

解决方案

- 不同理论之间通过接口属性交换信息
  - 接口属性：两种理论T_1和T_2都包含的命题集合
- 每种理论的求解器试图推导出所有接口属性

$$e_1 = a + 1$$
$$e_2 = 1 + a$$
$$e_3 = e_4 - 1$$

接口属性

1. $e_1 = e_2$
2. $e_3 = e_4$
3. 矛盾

$$e_3 = f(e_1)$$
$$e_4 = f(e_2)$$

- 如果任意一边推出矛盾，则不可满足
- 如果遍历所有的接口属性都没有矛盾，则可以满足

通常无限多，无法遍历

Nelson-Oppen方法

- 如果理论满足如下性质
  - 两个理论除了等号没有公共函数或谓词
  - 理论具备stably infinite属性
    - 即公理至少在某种无限域上成立
      - 公理：$\forall x. x = 1$只在{1}的有限域上成立
    - 正常理论都具备该属性
  - 理论是凸包，即
    - 如果$F \Rightarrow x_1 = y_1 \vee \cdots \vee x_n = y_n$，则有$\exists i. F \Rightarrow x_i = y_i$
    - EUF和线性方程组都是凸包
    - 线性整数不等式不是凸包
      - $0 \leq x \leq 1 \Rightarrow x = 0 \vee x = 1$

有限，可遍历

- 则接口属性只需要考虑变量之间的等价关系
- 由Nelson, Oppen等人在79、80的两篇论文中证明

Nelson-Oppen方法

- 给理论求解器添加接口方法：infer_equalities
  - 输入：
    - 一组公式F
    - 一组变量V
  - 输出：
    - 对于V变量所有可以推出的等价关系
- 比如：
  - 输入公式：$a = b, f(a) = x, f(b) = y$
  - 输入变量：a, b, x, y
  - 输出：$x = y, a = b$

- 实现：
  - 遍历V中的变量对x,y，然后求解$F \wedge x \neq y$，如果UNSAT说明x=y成立
  - 具体理论通常有高效的实现方式

第一步：变形约束

$$\begin{aligned} f(f(x)-f(y)) &= a \\ f(0) &= a+2 \\ x &= y \end{aligned}$$

- 反复按AST将其他理论的子树用变量代替

EUF

线性方程组

$$f(f(x)-f(y)) = a$$
$$f(0) = a+2$$
$$x = y$$

第一步：变形约束

$$
\begin{aligned}
f(f(x)-f(y)) &= a \\
f(0) &= a+2 \\
x &= y
\end{aligned}
$$

- 反复按AST将其他理论的子树用变量代替

EUF

$$
\begin{gathered}
f(e_1) = a \\
f(e_2) = e_3 \\
x = y
\end{gathered}
$$

线性方程组

$$
\begin{gathered}
e_1 = f(x) - f(y) \\
e_2 = 0 \\
e_3 = a + 2
\end{gathered}
$$

第一步：变形约束

$$\begin{aligned} f(f(x)-f(y)) &= a \\ f(0) &= a+2 \\ x &= y \end{aligned}$$

- 反复按AST将其他理论的子树用变量代替

EUF

$$f(e_1) = a$$
$$f(e_2) = e_3$$
$$x = y$$
$$f(x) = e_4$$
$$f(y) = e_5$$

线性方程组

$$e_1 = e_4 - e_5$$
$$e_2 = 0$$
$$e_3 = a + 2$$

第二步：基于接口属性求解

EUF

$$f(e_1) = a$$
$$f(e_2) = e_3$$
$$x = y$$
$$f(x) = e_4$$
$$f(y) = e_5$$

线性方程组

$$e_1 = e_4 - e_5$$
$$e_2 = 0$$
$$e_3 = a + 2$$

- 左右共享变量包括$V = \{e_1, e_2, e_3, e_4, e_5, a\}$
- 全部接口属性包括$P = \{\, x = y \mid x, y \in V \,\}$

第二步：基于接口属性求解

EUF

$$f(e_1) = a$$
$$f(e_2) = e_3$$
$$x = y$$
$$f(x) = e_4$$
$$f(y) = e_5$$

线性方程组

$$e_1 = e_4 - e_5$$
$$e_2 = 0$$
$$e_3 = a + 2$$

- EUF求解器返回SAT
- 线性求解器返回SAT
- EUF求解器推出$e_4 = e_5$

第二步：基于接口属性求解

EUF

$$f(e_1) = a$$
$$f(e_2) = e_3$$
$$x = y$$
$$f(x) = e_4$$
$$f(y) = e_5$$

线性方程组

$$e_1 = e_4 - e_5$$
$$e_2 = 0$$
$$e_3 = a + 2$$
$$e_4 = e_5$$

- 线性求解器返回SAT
- 线性求解器推出$e_1 = e_2$

第二步：基于接口属性求解

EUF

$$f(e_1) = a$$
$$f(e_2) = e_3$$
$$x = y$$
$$f(x) = e_4$$
$$f(y) = e_5$$
$$\mathrm{e}_1 = e_2$$

线性方程组

$$e_1 = e_4 - e_5$$
$$e_2 = 0$$
$$e_3 = a + 2$$
$$e_4 = e_5$$

- EUF求解器返回SAT
- EUF求解器推出$e_3 = a$

第二步：基于接口属性求解

EUF

$$f(e_1) = a$$
$$f(e_2) = e_3$$
$$x = y$$
$$f(x) = e_4$$
$$f(y) = e_5$$
$$\mathrm{e}_1 = e_2$$

线性方程组

$$e_1 = e_4 - e_5$$
$$e_2 = 0$$
$$e_3 = a + 2$$
$$e_4 = e_5$$
$$e_3 = a$$

- 线性求解器返回UNSAT
- 整体UNSAT

Nelson-Oppen方法：非凸包

- 非凸包的情况只用另外考虑等价关系的析取

- 任何时候遇到一个等价关系的析取式，依次尝试每个等价关系
  - 如果任意一个得出SAT，即整体SAT
  - 如果全部UNSAT，即整体UNSAT

Nelson-Oppen方法：非凸包

$$
\begin{aligned}
1 \leq \quad x \quad &\leq 2 \\
f(1) &= a \\
f(x) &= b \\
a &= b+2 \\
f(2) &= f(1)+3
\end{aligned}
$$

变形得到

Arithmetic			*EUF*		
1	$\leq$	x	$f(e_1)$	$=$	a
x	$\leq$	2	$f(x)$	$=$	b
e_1	$=$	1	$f(e_2)$	$=$	e_3
a	$=$	$b+2$	$f(e_1)$	$=$	e_4
e_2	$=$	2			
e_3	$=$	e_4+3			

Nelson-Oppen方法：非凸包

Arithmetic			*EUF*		
1	$\leq$	x	$f(e_1)$	$=$	a
x	$\leq$	2	$f(x)$	$=$	b
e_1	$=$	1	$f(e_2)$	$=$	e_3
a	$=$	$b+2$	$f(e_1)$	$=$	e_4
e_2	$=$	2			
e_3	$=$	e_4+3			

- 算术求解器返回SAT
- EUF求解器返回SAT
- EUF求解器推出$a = e_4$

Nelson-Oppen方法：非凸包

Arithmetic			*EUF*		
1	$\leq$	x	$f(e_1)$	$=$	a
x	$\leq$	2	$f(x)$	$=$	b
e_1	$=$	1	$f(e_2)$	$=$	e_3
a	$=$	$b+2$	$f(e_1)$	$=$	e_4
e_2	$=$	2			
e_3	$=$	e_4+3			
a	$=$	e_4			

- 算术求解器返回SAT
- 算术求解器推出$x = e_1 \vee x = e_2$

Nelson-Oppen方法：非凸包

Arithmetic			*EUF*		
1	$\leq$	x	$f(e_1)$	$=$	a
x	$\leq$	2	$f(x)$	$=$	b
e_1	$=$	1	$f(e_2)$	$=$	e_3
a	$=$	$b+2$	$f(e_1)$	$=$	e_4
e_2	$=$	2	x	$=$	e_1
e_3	$=$	e_4+3			
a	$=$	e_4			
x	$=$	e_1			

- 首先尝试$x = e_1$
- 添加$x = e_1$到左右两边的公式组（为什么？）
- EUF求解器返回SAT
- EUF求解器推导出$a = b$
- 算数求解器返回UNSAT

Nelson-Oppen方法：非凸包

Arithmetic			EUF		
1	$\leq$	x	$f(e_1)$	$=$	a
x	$\leq$	2	$f(x)$	$=$	b
e_1	$=$	1	$f(e_2)$	$=$	e_3
a	$=$	$b+2$	$f(e_1)$	$=$	e_4
e_2	$=$	2	x	$=$	e_2
e_3	$=$	e_4+3			
a	$=$	e_4			
x	$=$	e_2			

- 然后尝试$x = e_2$
- EUF求解器返回SAT
- EUF求解器推导出$b = e_3$
- 算数求解器返回UNSAT
- 整体UNSAT

SMT Solver的使用

- SMT-LIB
  - 标准的SMT输入格式
  - 被几乎所有的SMT Solver支持
  - 用于每年的SMT比赛中

SMT-LIB by Example

- > (declare-fun x () Int)
- > (declare-fun y () Int)
- > (assert (= (+ x (* 2 y)) 20))
- > (assert (= (- x y) 2))
- > (check-sat)
- sat
- > (get-value (x y))
- ((x 8)(y 6))
- > (exit)

Scope

- > (declare-fun x () Int)
- > (declare-fun y () Int)
- > (assert (= (+ x (* 2 y)) 20))
- > (push 1)
- > (assert (= (- x y) 2))
- > (check-sat)
- sat
- > (pop 1)
- > (push 1)
- > (assert (= (- x y) 3))
- > (check-sat)
- unsat
- > (pop 1)
- > (exit)

Defining a new type

- > (declare-sort A 0)
- > (declare-fun a () A)
- > (declare-fun b () A)
- > (declare-fun c () A)
- > (declare-fun d () A)
- > (declare-fun e () A)
- > (assert (or (= c a)(= c b)))
- > (assert (or (= d a)(= d b)))
- > (assert (or (= e a)(= e b)))
- > (push 1)
- > (distinct c d)
- > (check-sat)
- sat
- > (pop 1)
- > (push 1)
- > (distinct c d e)
- > (check-sat)
- unsat
- > (pop 1)
- > (exit)

(Recursive) Data Types

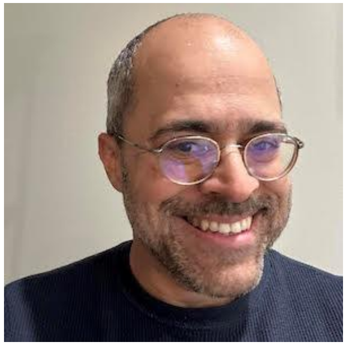
```
(declare-datatypes ((list (nil) (cons (hd
    Int) (tl list)))))
(declare-funs ((l1 list) (l2 list)))
(push)
(assert (not (= l1 nil)))
(assert (not (= (tl l1) nil)))
(assert (not (= l2 nil)))
(assert (= (hd l1) (hd l2)))
(assert (= (tl (tl l1)) (tl l2)))
(check-sat)
sat
(model)
("model" "l1 -> (cons 0 (cons 1 nil))
l2 -> (cons 0 nil)")
(pop)
(push)
(assert (not (= l1 nil)))
(assert (not (= l2 nil)))
(assert (= (hd l1) (hd l2)))
(assert (= (tl l1) (tl l2)))
(assert (not (= l1 l2)))
(check-sat)
unsat
(pop)
```

Quantifiers

```
(declare-fun IsNat (Int) Bool)
(assert (IsNat 1))
(assert (not (IsNat 0)))
(assert (forall (x Int) (=> (IsNat x) (IsNat (+ x 1)))))
(assert (forall (x Int) (=> (not (IsNat x)) (not (IsNat (- x 1))))))
(check-sat)
unknown
```

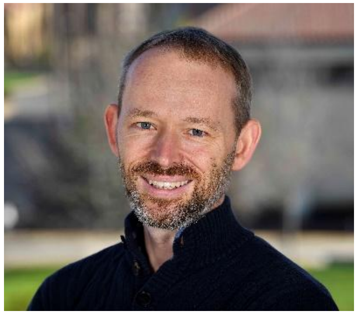
常见的SMT Solver

- Z3
  - 微软开发
  - 目前使用最广稳定性最好



Leonardo de Moura
Z3的核心开发人员

- CVC
  - 斯坦福大学和爱荷华大学开发
  - 持续更新，不断集成新的算法



Clark Barrett
CVC的主要领导者

课后作业

- 使用任意SMT Solver
  - 在线Z3： https://jfmc.github.io/z3-play/
- 发邮件给助教，回答如下问题：
  - 该SMT Solver的名字
  - 该SMT Solver支持的Theory
  - 构造该SMT Solver无法求解的约束，将运行结果截屏附在邮件中
  - 解释该SMT Solver为什么不能求解这个约束

参考资料

- Decision Procedures: An Algorithmic Point of View
  - Daniel Kroening and Ofer Strichman
  - Springer, 2008
- SMT-LIB
  - http://smtlib.cs.uiowa.edu/
- Z3教学网站
  - https://microsoft.github.io/z3guide/docs/logic/intro/