Capítulo 4: Verificação de Modelos baseada em SAT (3ª parte)

Cláusulas de Horn Restritas (CHC's)

> Esta secção é baseada nos textos *The Science, Art and Magic of Constraint Horn Clauses* , de Arie Gurfinkel e Nikolaj Bjorner, e *Inductive Approach to Spacer* , de Takeshi Tsukada e Hiroshi Unno.
> *Ver os manuscritos aqui.*

"Constraint Horn Clauses" (CHC's) são uma extensão de SMT's da Lógica de 1ª Ordem que,

- para além das variáveis que são entidades de ordem zero (isto é, variáveis que tomam valores nas teorias de base como `Bool`, `Int`, `BitVec`, `Arrays` ...) , usam também variáveis que são entidades de 1ª ordem: isto é, variáveis que tomam valores tomam valores sobre predicados e funções.
- permitem certas formas de quantificação sobre variáveis de ordem zero.

Sintaticamente formam-se CHC's a partir dos seguintes elementos sintáticos

- Uma teoria base T (inteiros, booleanos, vetores de bits, etc...)
- Variáveis v de ordem zero que tomam valores em T
- Termos de base $t(v)$ e predicados de base não quantificados $\varphi(v)$ na teoria T
- Variáveis $\mathrm{p}, \mathrm{h}, \cdots$ que denotam predicados representando conjuntos de termos $t(v)$.
  A aplicação do predicado p a termos $t(v)$ é aqui representada por

$$\mathrm{p}[\,t(v)\,]$$

  A forma genérica de uma CHC com n variáveis distintas de 1ª ordem $\mathrm{p_n}, \cdots, \mathrm{p_1}$ é

$$\forall\, v \;\cdot\; \varphi(v) \;\wedge\; \mathrm{p}_n[t_n(v)] \;\wedge\; \cdots \;\wedge\; \mathrm{p}_1[t_1(v)] \implies \mathrm{q}[t_0(v)]$$

  em que q *coincide com um dos* p_i *ou então é a constante* ***absurdo*** $\bot$.

Em particular quando $n = 1$ (existe uma única variável de 1ª ordem) a CHC diz-se **linear** . Uma CHC linear tem a forma simplificada

$$\forall\, v \cdot \varphi(v) \wedge \mathrm{p}[t_1(v)] \implies \mathrm{p}[t_0(v)] \quad \text{ou então}$$
$$\forall\, v \cdot \varphi(v) \wedge \mathrm{p}[t(v)] \implies \bot$$

Usando exclusivamente CHC's lineares é possível, como veremos nesta secção, estender o algoritmo PDR ("Property Directed Reachability") definido originalmente para máquinas com espaço de estados finitos para espaços de estado não-finitos.

O problema SAT em CHC's consiste em decidir se, dado um conjunto finito Γ deste tipo de fórmulas, existe uma atribuição de predicados às variáveis de 1ª ordem e uma atribuição de valores da teoria base às variáveis de ordem zero que faz com que, após a atribuição, todas as fórmulas $f \in \Gamma$ sejam válidas na teoria base.

O tipo de aplicações onde o problema SAT em CHC's é diretamente usado contém a verificação das *propriedades de segurança* em programas imperativos com ciclos *while* ou *propriedades de animação* neste tipo de programas. Esta abordagem pode ser estendida para verificação do mesmo tipo de propriedades em outros autómatos.

Dada a importância deste tipo de aplicações existe um grande esforço para construção de algoritmos de SAT CHC eficientes. Um dos primeiros é o algoritmo "Spacer" que está implementado como um "solver" específico dentro do Z3.

O seguinte exemplo ilustra o uso do algoritmo Spacer em Z3 para verificar que, dentro do ciclo "while" a variável x é sempre não-nula.

```
assert x > 0
on  : while x != 0
        x = x - 1
~on : stop
```

```
from z3 import *


# variáveis e constante de ordem zero
x , x_   = Ints('x x_')
```

```
on , on_ = Bools('on on_')
vars   = [x , on]
vars_  = [x_ , on_]
absurd = BoolVal(False)

# SFOTS
init   = And(0 < x, on)
t0     = And(on , x > 0  , x_ == x - 1 , on_ == on)
t1     = And(on , x == 0 , x_ == x , on_ == Not(on))
trans  = Or(t0 , t1)
error  = And( x != 0 , Not(on))

# Variável de 1ª ordem e suas instanciações 'antes' e 'depois'
das transições
Inv   = Function('inv', IntSort(), BoolSort(), BoolSort())
pre   = Inv(*vars)
pos   = Inv(*vars_)

# CHCs
chc0 = ForAll(vars , Implies(init , pre))
chc1 = ForAll(vars + vars_ , Implies(And(pre, trans), pos))
chc2 = ForAll(vars , Implies(And(pre , error), absurd))

# Solver
s   = SolverFor('HORN')          # novo solver para usar o algo
ritmo Spacer
s.add([chc0, chc1, chc2])        # adicionar as restrições ao s
olver
if s.check() == sat:             # SAT e imprimir resultado
    print(s.model().eval(pre))
```

```
# resultado
And(Not(x <= -1), Or(on, Not(x >= 1)))
```

Invariantes de ciclos *while*

Este exemplo sugere uma das aplicações mais simples dos CHC's: a procura de invariantes de ciclos. Recorde-se que, no caso geral, dado um programa imperativo cíclico do tipo

$$W \equiv \text{assume}\,\phi \;;\; \text{while}\; b \;\text{do}\; S \;\text{od}\;;\; \text{assert}\,\varphi$$

este programa é correcto se existe um predicado ϑ que verifica as seguintes três condições:

- $\phi \wedge b \Rightarrow \vartheta$ é tautologia
- $\vartheta \Rightarrow \text{wp}_S(\vartheta)$ é tautologia ou, equivalentemente, $\vartheta \wedge \text{T}_S \Rightarrow \vartheta'$ é tautologia.
  aqui wp_S e T_S são, respetivamente, o transformador "weakest pre-condition" associado ao corpo do ciclo S e a relação de transição associada ao mesmo comando.
- $\vartheta \wedge \neg b \wedge \neg \varphi \Rightarrow \bot$ é tautologia

Tal predicado designa-se por **invariante do ciclo**.

Escrito de forma quantificada usando um vetor x de variáveis de estado e ϑ como variável de 1ª ordem, estas condições escrevem-se como um sistema linear de CHC's

- $\forall\, x \cdot \phi(x) \wedge b(x) \Rightarrow \vartheta(x)$
- $\forall\, x, x' \cdot \vartheta(x) \wedge \text{T}_S(x, x') \Rightarrow \vartheta(x')$
- $\forall\, x, x' \cdot \vartheta(x) \wedge \neg b(x) \wedge \neg \varphi(x) \Rightarrow \bot$

Esta formalização codifica-se diretamente em Z3 seguindo o exemplo anterior e, caso a teoria de base onde estão escritos os predicados ϕ, φ, b, T_S suporte as CHC's, com ela é possível verificar se existe um invariante de ciclo ϑ e (caso exista) encontrar tal invariante.

Interpolador de Craig

Outra aplicação simples das CHC's, em SMT's adequadas, é a procura de um *interpolante de Craig* para dois predicados $A(x, w)$ e $B(y, w)$ que verifiquem

$$\forall\, x, w, y \cdot A(x, w) \wedge B(y, w) \Rightarrow \bot$$

O interpolador de Craig é um predicado $C(w)$ que depende apenas das variáveis que são comuns a A e a B e que verifica duas condições definidoras

- $\forall\, x, w \cdot A(x, w) \Rightarrow C(w)$
- $\forall\, y, w \cdot C(w) \wedge B(y, w) \Rightarrow \bot$

Estas condições têm a forma de CHC's que podem ser codificadas em Z3 se a teoria de base for adequada.

O seguinte código ilustra a utilização do algoritmo Z3 Spacer na procura dos interpeladores. O programa usa a função auxiliar `free_vars` para extrair as variáveis livres de um term Z3 qualquer.

```
from z3 import *

#Returns the set of all  uninterpreted constants in a formula
def free_vars(fml):
    seen = set([]) ; vars = set([])
    def fv(seen, vars, f):
        if f in seen:
            return
        seen |= { f }
        if f.decl().kind() == Z3_OP_UNINTERPRETED:
            vars |= { f }
        for ch in f.children():
            fv(seen, vars, ch)
    fv(seen, vars, fml)
    return vars

#Binary Craig Interpolation by reduction to CHC
def interpolate(A, B):
    As = free_vars(A)
    Bs = free_vars(B)
```

```
    shared = [s for s in As & Bs ]

    Itp = Function('Itp', [s.sort() for s in shared] + [BoolSo
rt()])
    C  = Itp(shared)
    # as regras definidoras do interpolador
    chc0 = ForAll([a for a in As], Implies(A, C))
    chc1 = ForAll([b for b in Bs], Implies(And(C, B), BoolVal
(False)))

    solver = SolverFor('HORN')
    solver.add([chc0, chc1])
    if solver.check() == sat:
        return solver.model().eval(C)
    return None
```