補題発見支援機構利用手引き

ver. 0.1

澤田 寿実
(株) 考作舎
tswd@kosakusha.com

2016/2/29

目次

1

補題発見支援機構の概要

CafeOBJ 言語による仕様記述では，通常組み込みモジュール**B OOL** で定義されているソート **B ool** の **true** および **false** を用いて真偽値を表現する．モジュール **B OOL** はそのような使い方を 想定して様々な論理演算子(**and**, **or**, **xor**, **imply** など)を導入しており，これらの論理演算子を用いて表現された項は， $C_1 \mathrm{xor} \ldots C_n$ の形をした標準形(XOR標準形)の形に (BOOLモジュールの等式によって)変換される（ここで b_n は，$b_1 \mathrm{and} \ldots b_m$ である．）

たとえば，**(B 1:B ool or B 2:B ool and B 3:B ool) implies B 4:B ool** という形の項は

```
((B 2 and B 3) xor (B 1 xor ((B 1 and (B 3 and B 2))
                          xor
                          (true xor ((B 1 and (B 2 and (B 3 and B 4)))
                                    xor ((B 1 and B 4) xor (B 3 and (B 2 and B 4))))))))
```

のような形の標準形に変換される．見ての通り元の項に比べて 意味的に同等ではあるが，XOR標準形は人間にとってその構造や 意味を直感的に把握するのが困難である．

証明過程においては，途中出現するさまざまな Bool 項について その真偽が想定したものとなっているかどうかが調べられるが， **true** や **false** に簡約されず， 上記のXOR標準形のままに残ることがある． 多くが何らかの仮定が不足だったり，想定していないケース漏れで あったりする． そのような場合，**true** や **false** に簡約されなかった 項を観察し，必要とされている前提条件を推測するのが一般に取られる 手段である． それらの仮定をうまく補題として取り出すことができると以降の証明が先まで進んで行く． その場合に障害となるのが，上で見た通りXOR標準形は項のサイズが 大きくなる傾向があること，また人間の日常的に慣れ親しんでいる 真偽判定のやり方と乖離があり，何を仮定すれば全体の Bool 項をより 簡略化できるかが了解しにくいということである．

この問題を解決するための支援機構として，Bool項検査開始コマンド(**binspect**) ならびに **binspect** で指定された項の真偽値が，それらを構成する 部分項の真偽値によりどのようになるかを調べるためのコマンド(**bresolve**, **bguess**)などを導入した． 以下ではこれらのコマンドについてその使用方法を説明する．

2

補題発見支援機構コマンド

2.1 bispect コマンド

- **binspect** コマンドは与えられたBool項を抽象化し，見やすい形にするとともに その項を **bresolve** や **bguess** コマンドの適用対象として設定する．
- 構文

```
binspectコマンド ::= binspect [in <ModuleName>]: <term> .
```

ここで，**<term>** は組み込みモジュールBOOLで定義されたソート Bool の項でなければならない．

- 実行例-1

```
binspect in NAT : (N1:Nat * N2:Nat) = (N1:Nat * N3:Nat) implies N2 = N3 .

(((N1 * N2) = (N1 * N3)) implies (N2 = N3)):B ool
--> (((N2:Nat * N1:Nat) = (N3:Nat * N1)) xor (true xor ((N2 = N3) and ((N2 * N1) = (N3 * N1)))))
** Abstracted boolean term:
  ((‘P -1:B ool and ‘P -2:B ool) xor (‘P -1 xor true))
  where
    ‘P -1 = ((N2:Nat * N1:Nat) = (N3:Nat * N1))
    ‘P -2 = (N2:Nat = N3:Nat)
```

- もとの項が **(‘P -1:B ool and ‘P -2:B ool) xor (‘P -1 xor true)** という XOR 標準形に変換され，また **N2:Nat = N3:Nat** のような部分項は **‘P -1** という擬似定数に変換され全体の構造がより視認されやすいようになっている．
- 実行例-2

```
CafeOB J> binspect in NAT : (N1:Nat = N2:Nat) and (N2:Nat = N3:Nat + 1)
           implies (N1 = N3 + 1) or (N2 = N1) .

(((N1 = N2) and (N2 = (1 + N3))) implies ((N1 = (1 + N3)) or (N2 = N1))):B ool
--> true
```

```
** Abstracted boolean term:
  true
```

- 上の例のように，変換結果が **true** または **false** となる場合は　その旨表示される．
- 実行例-3

```
CafeOB J> binspect in NAT : (N1:Nat * N2:Nat) = 0 or (N2 * N1) = 0 implies (N1 = 0 or N2 = 0) .

((((N1 * N2) = 0) or ((N2 * N1) = 0)) implies ((N1 = 0) or (N2 = 0))):B ool
--> (((N2:Nat * N1:Nat) = 0) xor (true xor (((N1 = 0) and ((N2 = 0) and ((N1 * N2) = 0)))
        xor (((N1 = 0) and ((N2 * N1) = 0)) xor ((N2 = 0) and ((N2 * N1) = 0))))))
** Abstracted boolean term:
  ((‘P -1:B ool and ‘P -3:B ool) xor ((‘P -1 and ‘P -2:B ool) xor ((‘P -1 and (‘P -3 and ‘P -2)) xor (‘P -1 xor
  where
    ‘P -1 = ((N2:Nat * N1:Nat) = 0)
    ‘P -2 = (N1:Nat = 0)
    ‘P -3 = (N2:Nat = 0)
```

- 同じ部分項は同じ擬似定数（上例では **N1:Nat * N2:Nat = 0**に対して**‘P -1**)で抽象化される．

2.2　bshow コマンド

- **bshow** コマンドは **binspect** コマンドで指定した項を印字する．
- 構文

```
bshowコマンド ::= bshow [{ grind | tree }]
```

- 実行例-1

bshow は先に binspet で指定したBool項を表示する．

```
CafeOB J> bshow
((‘P -1:B ool and ‘P -3:B ool) xor ((‘P -1 and ‘P -2:B ool) xor ((‘P -1 and (‘P -3 and ‘P -2)) xor (‘P -1 xor tr
where
  ‘P -1 = ((N2:Nat * N1:Nat) = 0)
  ‘P -2 = (N1:Nat = 0)
  ‘P -3 = (N2:Nat = 0)
```

- 実行例-2

オプション引数 **grind** を指定すると，XOR 標準形の部分項を構成する and で結ばれた項を集約して表示する．大きなサイズの項の構造を把握するために便利である．

```
CafeOB J> bshow grind

>> xor ***>
true
>> and --->
‘P -1 = ((N1:Nat * N2:Nat) = 0)
‘P -2 = (N1:Nat = 0)
‘P -3 = (N2:Nat = 0)
```

```
<----------
>> and --->
'P -1 = ((N2:Nat * N1:Nat) = 0)
'P -2 = (N1:Nat = 0)
<----------
>> and --->
'P -1 = ((N2:Nat * N1:Nat) = 0)
'P -3 = (N2:Nat = 0)
<----------
'P -1 = ((N2:Nat * N1:Nat) = 0)
<**********
```

- 実行例-3

  オプション引数の **tree** を指定すると，抽象化したBool項の木構造を 把握しやすい形式で印字する．

```
bshow tree

_xor_
    _and_
        'P -1:B ool
        'P -3:B ool
    _xor_
        _and_
            'P -1:B ool
            'P -2:B ool
        _xor_
            _and_
                'P -1:B ool
                _and_
                    'P -3:B ool
                    'P -2:B ool
            _xor_
                'P -1:B ool
                true
where
  'P -1 = ((N2:Nat * N1:Nat) = 0)
  'P -2 = (N1:Nat = 0)
  'P -3 = (N2:Nat = 0)
```

2.3 bresolveコマンド

- **bresolve** は **binspect** コマンドで解析対象として指定した項の xor の各部分項に **true** と**false** の組み合わせを代入し，結果として **true** となるような組み合わせを 探索し，その結果を表示する．
- 構文

```
bresolveコマンド ::= bresolve [ <number> | all ]
```

- 実行例-1
  オプション引数 <number> は true または false を割振る擬似定数の組み合わせの個数を指定する． 指定しない場合のデフォルトは1であり，この場合抽象化されたBool項に出現する個々の擬似定数にたいして true または false を代入し，全体の結果が true となるものを探す．

```
CafeOB J> bresolve

** (1) The following assignment(s) makes the term to be 'true'.
[1] { 'P -1:B ool |-> false }
where
  'P -1 = ((N2:Nat * N1:Nat) = 0)

[2] { 'P -2:B ool |-> true }
where
  'P -2 = (N1:Nat = 0)

[3] { 'P -3:B ool |-> true }
where
  'P -3 = (N2:Nat = 0)
```

- 実行例-2
  オプション引数 <number> に all を指定すると，全ての可能な true/false の組み合わせについて評価し，結果が true となる代入を探す．

```
bresolve all

** (1) The following assignment(s) makes the term to be 'true'.
[1] { 'P -1:B ool |-> false }
where
  'P -1 = ((N2:Nat * N1:Nat) = 0)

[2] { 'P -2:B ool |-> true }
where
  'P -2 = (N1:Nat = 0)

[3] { 'P -3:B ool |-> true }
where
```

```
  'P -3 = (N2:Nat = 0)

** (2) The following assignment(s) makes the term to be 'true'.
[1] { 'P -1:B ool |-> true, 'P -2:B ool |-> true }
[2] { 'P -1:B ool |-> false, 'P -2:B ool |-> true }
[3] { 'P -1:B ool |-> false, 'P -2:B ool |-> false }
where
  'P -1 = ((N2:Nat * N1:Nat) = 0)
  'P -2 = (N1:Nat = 0)

[4] { 'P -1:B ool |-> true, 'P -3:B ool |-> true }
[5] { 'P -1:B ool |-> false, 'P -3:B ool |-> true }
[6] { 'P -1:B ool |-> false, 'P -3:B ool |-> false }
where
  'P -1 = ((N2:Nat * N1:Nat) = 0)
  'P -3 = (N2:Nat = 0)

[7] { 'P -2:B ool |-> true, 'P -3:B ool |-> true }
[8] { 'P -2:B ool |-> false, 'P -3:B ool |-> true }
[9] { 'P -2:B ool |-> true, 'P -3:B ool |-> false }
where
  'P -2 = (N1:Nat = 0)
  'P -3 = (N2:Nat = 0)

** (3) The following assignment(s) makes the term to be 'true'.
[1] { 'P -1:B ool |-> true, 'P -2:B ool |-> true, 'P -3:B ool |-> true }
[2] { 'P -1:B ool |-> false, 'P -2:B ool |-> true, 'P -3:B ool |-> true }
[3] { 'P -1:B ool |-> true, 'P -2:B ool |-> false, 'P -3:B ool |-> true }
[4] { 'P -1:B ool |-> false, 'P -2:B ool |-> false, 'P -3:B ool |-> true }
[5] { 'P -1:B ool |-> true, 'P -2:B ool |-> true, 'P -3:B ool |-> false }
[6] { 'P -1:B ool |-> false, 'P -2:B ool |-> true, 'P -3:B ool |-> false }
[7] { 'P -1:B ool |-> false, 'P -2:B ool |-> false, 'P -3:B ool |-> false }
where
  'P -1 = ((N2:Nat * N1:Nat) = 0)
  'P -2 = (N1:Nat = 0)
  'P -3 = (N2:Nat = 0)
```

2.4 bguessコマンド

- **bguess** コマンドは **binspect** コマンドによって指定された Bool 項に対して 経験から得られている幾つかの有用な推測方法を適用し結果を調べる.

- 構文

```
bguessコマンド ::= bguess { imply  | and | or }
```

- 実行例-1

  引数 **imply** を指定した場合，抽象化した Bool 項に含まれる 擬似定数 p, q について**p implies q = true** が成立すると仮定した時に 結果がどうなるかを調べ，true となる場合その仮定を公理の形式で印字して示す．

```
CafeOB J> bg imply

** (1) each of the following equations makes the inspected term 'true'
[1] eq [:bimply ]: ('P -1:B ool and 'P -3:B ool) = 'P -1 .
where
  'P -1 = ((N2:Nat * N1:Nat) = 0)
  'P -3 = (N2:Nat = 0)

[2] eq [:bimply ]: ('P -1:B ool and 'P -2:B ool) = 'P -1 .
where
  'P -1 = ((N2:Nat * N1:Nat) = 0)
  'P -2 = (N1:Nat = 0)
```

- 上の例のように **bguess** は **bg** と略称することができる．
- **p implies q = true** は XOR 標準形を用いた **p and q = p** に論理的に等価である． 結果の印字では XOR 標準形を用いた形式で印字する．
- and

  bguess の引数として **and** を指定した場合は，**p and q = false** という形の 仮定を導入した場合に結果がどうなるかを調べる．
- or

  bguess の引数として **or** を指定した場合は，**p or q = true** という形の 仮定を導入した場合に結果がどうなるかを調べる．
- 現在までのところ，引数 and や or により有用な補題の発見ができた例は存在しない．