A design for predicate calculus prover based on completion.

A talk in RICAM, Linz, April 2006.

S.D.Mechveliani
Program Systems Institute, Pereslavl-Zalessky, Russia,
e-mail: mechvel@botik.ru

1 Introduction

1.1 Acknowledgments

This talk has been prepared in March – April 2006, during the Special Semester on Gröbner Bases, organized by RICAM, Austrian Academy of Sciences, Johannes Kepler University.

The author is grateful to the supporters of his travel to Linz and of his work in Linz:
Russian Foundation for Basic Research (grant 06-01-10519),
organizers and sponsors of the Special Gröbner Semester.

This work is also a part of the `Dumatel` project. And this project is partially supported in 2006 by the Program of Fundamental Research of the Russian Academy of Sciences Presidium ("Razrabotka fundamentalnykh osnov sozdaniya nauchnoj raspredelennoj informatcionno-vychislitelnoj sredy na osnove tekhnologij GRID").

1.2 Our general problem and goals

(1) To develop an automatic reasoning tool for processing knowledge expressed as equations, as it is known in the *term rewriting* area.

(2) To extend the equation processing tool to enable proofs in predicate calculus.

(3) To make this equation processing tool also an adequate functional programming language for mathematical computation. This must constitute a system in which there is no gap between direct computation and reasoning.

For example, the direction (1) and (3) were developed, to a certain extent, in the `Maude` project, with its order-sorted term rewriting, reflection, functorial tools, and such (see `Maude` in Internet).

1.3 The problem described in this particular talk:

extending the unfailing completion technique for many-sorted term rewriting to Boolean terms, so that this would enable proofs in predicate calculus.

1.4 History

Our design bases on the following works.

- The completion method (KBC) by Knuth and Bendix of 1971 [KB].
- The refinement of this method to "unfailing completion" by [Hsi:Ru] (1987) and by [B:D:P] (1989).
- Certain advanced strategy design for unfailing completion described in [Lo:Hi] (2002).
- The extention of KBC to Boolean terms, with application to proofs in predicate calculus [Hsi] (1985).

1.5 The particular points of our design

- It accomodates the Boolean term extension by [Hsi] of KBC to unfailing completion, with applying a certain variant of the strategy by [Lo:Hi].
- It adds to the completion strategy the parameters for the `Superposability` and `Preference` flags. This provides means for narrowing the proof search.
- It allows a partial completion bounded by the *resource bound*.
  This serves for cautious use of completion as of one of the search steps in the inductive proof search.
- We consider different methods for the Boolean term (BT).
- It includes a strategy for the proof via completion and considering 'cases' for the terms with a finitely enumerated value domain.

Let us illustrate our approach by considering examples.

2 Proof by "positive" ordinary unfailing completion

Example 1. Prove that the operation of inverse in a group is unique.

Define a calculus `group_inv2a` for `Group`, with the two inverse operators, with equality predicate `"~"` (with a single trivial law for it), and with additional constant `"a"`, which meaning is a 'manually' introduced Skolem constant.

```
-------------------------------------------------------------------------------
module Main
where
import Dumatel

group_inv2a =
  parseCalculus
  0 defaultVariableWeight defaultInitialOpId defaultParenthesisTable
  SkolemizeNot
  $
  CalculusDeclaration
  {sortOpDecls =
     [Sorts ["G", "Bool"],

      Op " true   []  []  Bool "   [],
      Op " false  []  []  Bool "   [],
      Op " ~      [G] [G] Bool "   [ParsePreceds 1 1],

      Op " 1  []  []  G "   [],                   -- 1 : G
      Op " *  [G] [G] G "   [ParsePreceds 5 5],   -- * : G G -> G
      Op " i  []  [G] G "   [ParsePreceds 0 7],   -- i : G -> G

      Op " f  []  [G] G "   [ParsePreceds 0 7],
      Op " a  []  []  G "   []
     ],
   opPrecedDecls = [OpPrecedDescentName
                                ["a", "~", "f", "i", "*", "1", "false", "true"]],
   termCompDecl  = Just lpo,
   opWeightDecls = [],
   formulaDecls  = [],
   ruleDecls     = [],
   varDecls      = [" x y z u v w :  G "],

   eqDecls = [ " (x*y)*z = x*(y*z) ",   " 1*x     = x ",
               " (i x)*x = 1       ",   " (f x)*x = 1 ",

               " x ~ x = true "   -- equality predicate  ]
  }
```

```
main =                  -- the client program -----------------------------------
  putStr $ concat
  [
   "\n\n Applying completion ...\n",
   "\n Remaining goals =  ",          show $ ukbGoalRem res,
   "\n Resource remainder =  ",       show $ ukbRcRem $ ukbStat res,
   "\n Produced equations =\n ",      show $ ukbCompl res,
   "\n\nProof trace =\n ",            shows (ukbTrace res) ""
  ]
  where
  calc         = group_inv2a
  sortNameToId = fst $ sortIdTabs calc
  otab         = opTab calc
  preced       = opPrecedence otab

  goal = eqnSides $ parseSingleOrBreak calc ""   " (f a) ~ (i a) = false "
                                                      -- formula to refute
  jump = 0
  res  = ukb Infinity jump [] calc [goal]
```

This aims at proving `f a = i a` for an arbitrary constant `a`.
It applies `ukb` to the calculus.
And this procedure stops when `(f a)` and `(i a)` connect by the current set `eE'` *or* the set `eE'` is complete.

For completeness, we need: a ground goal.

This prints out the completion result, and then, the trace of the proof.
In this output we use the following abbreviations.

`Sup`	means	`Superposable`.
`Od`	means	`Ordinary`.
`Pref`	means	`Preferable`.
`Par, P`	mean	`Parents`.
`"Chosen"`	means	`"Chosen equations after reduction"`.

```
 Applying completion ...

 Remaining goals    =  [(true, false)]     -- true = false  derived
 Resource remainder =  Infinity
 Produced equations =
 [
  [10] x * 1 -> x                  Sup Od (Parents [[6], [2]]),
  [3]  1 * x -> x                  Sup Od (Parents []),
  [20] i i x -> x                  Sup Od (Parents [[16], [6]]),
  [22] x * ((i x) * z) -> z        Sup Od (Parents [[20], [6]]),
  [6]  (i x) * (x * z) -> z        Sup Od (Parents [[4], [2]]),
  [5]  x ~ x -> true               Sup Od (Parents []),
  [23] x * (i x) -> 1              Sup Od (Parents [[20], [2]]),
  [2]  (i x) * x -> 1              Sup Od (Parents []),
  [15] i 1 -> 1                    Sup Od (Parents [[13], [2]]),
  [4]  (x * y) * z -> x * (y * z)  Sup Od (Parents []),
  [57] i (y * x) -> (i x) * (i y)  Sup Od (Parents [[46], [20]]),

  [25] f x -> i x                  Sup Od (Parents [[11], [20]])
  --
  -- f x = i x  is proved
 ]

Proof trace =
 [
(Chosen  [3] 1 * x -> x          Sup Od (Parents [])),
(Chosen  [5] x ~ x -> true       Sup Od (Parents [])),
(Chosen  [1] (f x) * x -> 1      Sup Od (Parents [])),
(Chosen  [2] (i x) * x -> 1      Sup Od (Parents [])),
(Chosen  [4] (x*y)*z -> x*(y*z)  Sup Od (Parents [])),

(Chosen  [6] (i x)*(x*z) -> z    Sup Od (Parents [[4], [2]])
),
-- because by superposing  [2]  (i u)*u         =  1
-- with                              |
--                         [4]  ( x  * y ) * z  =  x * (y * z)
-- by
-- [u := y, x := i y],          (i y) * y       =  1
--
--                              ((i y)* y ) * z =  (i y) * (y * z),
-- it obtains
--                          1*z = (i y)*(y*z)   --reduction-->   (i y)*(y*z) = z


(Chosen  [8] (i 1)*x -> x        Sup Od (Parents [[6], [3]])  ),
(Chosen  [13] (i i 1)*x -> x     Sup Od (Parents [[8], [6]])  ),
```

```
(Chosen  [15] i 1 -> 1           Sup Od (Parents [[13], [2]]) ),
(The following facts have been newly reduced, or subsumed,
and moved to delayed set:                                 [[8], [13]]
),
(Chosen  [10] (i i x)*1 -> x    Sup Od (Parents [[6], [2]])  ),
(Chosen  [11] (i f x)*1 -> x    Sup Od (Parents [[6], [1]])  ),
(Chosen  [16] (i i i x)*x -> 1  Sup Od (Parents [[10], [6]]) ),
(Chosen  [20] i i x -> x        Sup Od (Parents [[16], [6]]) ),
(The following facts have been newly reduced,
or subsumed, and moved to delayed set:         [[10], [16]]
),
(Chosen  [10] x*1 -> x          Sup Od (Parents [[6], [2]]) ),
(The following facts have been newly reduced,
or subsumed, and moved to delayed set:         [[11]]
),
(Chosen  [24] f 1 -> 1          Sup Od (Parents [[10], [1]])  ),
(Chosen  [11] i f x -> x        Sup Od (Parents [[6], [1]])   ),

(Chosen  [25] f x -> i x        Sup Od (Parents [[11], [20]]) ),      -- proved !

(The following facts have been newly reduced,
or subsumed, and moved to delayed set:         [[11], [1], [24]]
),
(Chosen  [23] x*(i x) -> 1           Sup Od (P [[20], [2]])  ),
(Chosen  [22] x*((i x)*z) -> z       Sup Od (P [[20], [6]])  ),
(Chosen  [29] x*(y*(i (x*y))) -> 1   Sup Od (P [[23], [4]])  ),
(Chosen  [31] y*(i ((i x)*y)) -> x   Sup Od (P [[29], [22]]) ),
(Chosen  [39] y*(i (x*y)) -> i x     Sup Od (P [[31], [20]]) ),
(The following facts have been newly reduced,
or subsumed, and moved to delayed set:        [[31], [29]]
),
(Chosen  [52] (i (x*y))*x -> i y      Sup Od (P [[39], [39]]) ),
(Chosen  [46] i (x*(i y)) -> y*(i x)  Sup Od (P [[39], [22]]) ),
(Chosen  [57] i (y*x) -> (i x)*(i y)  Sup Od (P [[46], [20]]) ),
(The following facts have been newly reduced,
or subsumed, and moved to delayed set:        [[46], [39], [52]]
)]
```

2.1 Ordering on terms

This must be a *simplification ordering total on ground terms* ([Hsi:Ru, B:D:P]). ("SOTG ordering").
Its laws are as follows.

(1) `">>"` is a *terminating* partial ordering.

(2) `c[t] >> t` for any proper subterm `t`.

(3) Monotononicity: `s >> t ==> c[s] >> c[t]`
(for example, `u >> v ==> u+a >> v+a`).

(4) `s >> t ==> sb s >> sb t`
for any substitution `sb`.

(5) It is a total ordering on the set of ground terms.

'Terminating' maens 'Noetherian'.

Example: lexicographic path ordering (`lpo`):

```
lpo tab s t =  (s /= t) && lpo' s t
  where
  lpo' s             (VarT x)        =  varOccurs x [s]
  lpo' (VarT _)      (App _ _ _)     =  False
  lpo' (App f ls rs) (App g ls' rs') =
    let
      lsrs                 = ls  ++ rs
      ls'rs'               = ls' ++ rs'
      s_greaterAllSubOf_t = and [lpo tab s s' | s' <- ls'rs']
    in
    or [or [geqTerm (lpo tab) si t | si <- lsrs]
        ,
        greaterOp tab f g  &&  s_greaterAllSubOf_t
        ,
        f == g  &&  s_greaterAllSubOf_t
        &&
        (case  dropWhile (\ (s, t) -> s == t) $ zip lsrs ls'rs'
         of
         (s, t): _ -> lpo tab s t
         []        -> False
        )
       ]
```

Problem: how to optimize this implementation?

3 Proof in predicate calculus by Boolean terms

Example 2. (taken from [Hsi])

Let `G` be a group with the operations `e, *, i`, and `P` a non-empty subset in `G` such that if `X` and `Y` belong to `P` then `X*(i Y)` belongs to `P`.

Prove that it follows from these axioms that

(1) `e` $\in$ `P`, **(2)** for all `X (X` $\in$ `P ==> i X` $\in$ `P)`

To solve this, denote `x` $\in$ `P` as `(P x)`, and create a calculus `groupPCalculus`, with the

sort `G`, operators `e, i, *, P`, variables `X, Y, Z`,
the three equations for the group laws,
equations for the Boolean connectives (like `x & true = true` and the such — we do not print them here),
operator precedence `P >> i >> * >> ...`

Set the term ordering as `lpo` — for example.
The problem is set as

```
groupPCalculus |-
            ( (exist [X] (P X))  &  (forall [X,Y] (P X & P Y ==> P (X*(i Y))))
              ==>
              P e  &  (forall [X] (P X ==> P (i X)))
            )
```

We denote:
`"not"` — negation, `"&"` — conjunction, `"|"` — disjunction,
`"xor"` — exclusive disjunction, `"==>"` — implication.

This converts to four *disjuncts*:

```
[ P XSk,             not P X | not P Y | P (X*(i Y)),

  not P e | P XSk0,   not P e | not P i XSk0           ]
```

The disjuncts convert to BT-s, with

`not x` converting to `x xor 1`, `true -> 1`, `false -> 0`,

and so on, the BT simplification rules applied ([Hsi] Sections 2, 3).
In our example, there appear four bt-disjuncts:

```
[(P XSk)  xor  1,                                   -- converts to equation
 (P (X*(i Y))) & (P Y) & (P X)  xor  (P Y) & (P X),
 (P XSk0) & (P e)  xor  (P e),
 (P i XSk0) & (P e)
]
```

`"&"` stands for multiplication, `xor` is addition `"+"`.

A disjunct `d` has the meaning of the statement `d = 1 (= true)` — with the presumed universal quantifier.

This statement is equivalent to `r(d) = 0 (= false)`, where the method `r` converts from disjunct to BT as follows:

```
r [] = 1,   r (l | d') = (l+1) & (r d').
```

Here `l` is a literal, and for negative `l = not l'`, `l+1` is put `l'`.

3.1 Meaning of a Boolean term

BT is a commutative sum of several different monomials.
A monomial is a commutative product of several different *atoms*.
The coefficients of monomials are integers modulo 2.
It holds

`A & A = A` for any atom `A`.

'Atom' means an entity which is not structured further by the connectives `&, xor`.

Examples

`(P e)` is a ground atom ("predicate term").
`(P (X*(i X))` is a non-ground atom.

`(P (X*(i Y))) & (P Y) & (P X)` is a monomial of three atoms.

```
(P (X*(i Y))) & (P Y) & (P X)  xor  (P Y) & (P X)
```

consists of two monomials, and represents the statement

```
(P Y) & (P X)  ==>  P (X*(i Y))
```

(the quantifier is presumed).

The meaning of a BT `f` is the equation

`f = false`, or `f = 0`,

(compare with a polynomial equation).

In our example, `(P i XSk0) & (P e)`
means the negation for

```
(P i XSk0) & (P e) =  1,
```

and

```
(P (X*(i Y))) & (P Y) & (P X)  xor  (P Y) & (P X)
```

represents the negation for

```
(P (X*(i Y)))&(P Y)&(P X)  xor  (P Y)&(P X)  xor  1  = 0,
```

and thus, it represents the statement

```
((P Y) & (P X)  ==>  P (X*(i Y)))  = true.
```

The refutation procedure adds the obtained BT-s to the calculus and applies *completion*.

Our completion procedure processes 'facts': equations and BT.

BT are being *reduced* by equations and by BT by `reduceBT`.

Equations try to superpose with equations and with BT;
and a monomial BT tries to superpose with any BT at various monomials.

<u>A particular point about Boolean rules</u>

(like `x & true = true` and such):
they apply in reduction but do not superpose.

3.2 Superpositions

are of kind "ee", "be", "bb".

"bb" superposes a monomial `m` with a BT `f`.
by 'mul-unifying' `m` with some monomial in `f`.

A monomial-to-monomial (mul-)unification is by unifying an atom subset and by applying the complementary factors
(compare to LCM for polynomial power products).

This is the AC+idempotence unification for `"&"`.
It finds a complete set of mul-unifiers.

<u>Example</u>

For our example with `groupPCalculus`, suppose that the completion deals with the BT (1) and (2), as below.
Then, it produces

```
(1)  (P Y) & (P i Y) & (P e)  xor  (P Y) & (P e)  |
(2)  (P i XSk0) & (P e)                           |   -->  (P XSk0) & (P e).
```

Because `[Y := XSk0]` unifies (2) with a submonomial of the head monomial of the BT instance

```
(P XSk0) & (P i XSk0)&(P e)  xor  (P XSk0) & (P e).
```

Now, similar as with polynomials, the submonomial after this substitution is replaced with `0`,
and there remains the smaller substituted monomial `(P XSk0) & (P e)`.
Then the BT

```
(P XSk0) & (P e)  xor  (P e)
```

from the initial basis *reduces* the result to `(P e)`.
This reduction is due to the assumed multiset monomial ordering
explained below
(compare this reduction with polynomial reduction).

In the general case, there may be needed the complementary factors.
(compare to s-polynomial of Gröbner bases [Bu]).

<u>What is a refutation procedure:</u>
by superposing 'facts', and by reducing them by each other, try to obtain the unity BT.
This is equivalent to deriving `true = false`
(compare to polynomial completion).

For completeness, it is sufficient only BN-superposition for "bb" superposition.
Also this restriction helps efficiency.

The method scheme we have sketched, is a certain version of the `RN+` strategy by [Hsi].

3.3 Reduction

The ‘facts’ are being reduced by each other.
The more is reduced the current fact the smaller search space remains.

The ”ee” reduction is as in the classic unfailing completion.
The ”be” reduction is through reduction of each atom.

For reduction, we need a partial ordering `">m>"` on monomials, with extension to BT.
`">m>"` needs to agree with the partial TRW ordering `">>"`.

The reduction of a BT `f` aims at decreasing `f` in the `">m>"` ordering by reducing by equations and by subtracting BT of kind

`q & (sb g),`

where `g` is some active BT, `sb` a selected substitution, `q` a selected monomial factor.

[Hsi] suggests the reduction only by monomials,
and its presumed ordering is, evidently, ”`f` is a subset in `g`”.

Currently, we think that the most sensible reduction is by putting

`(>m>) = multiset ordering extention of (>>).`

Because it agrees with the reduction by equations and seems to have certain good properties.
Its rules are

```
mon >mls> mon' =  gtm (mon \\ mon') (mon' \\ mon)    -- cancel intersection
  where
  gtm _  [] =  True    -- case of a proper subset
  gtm [] _  =  False   --
  gtm m  n  =  exist [a <- m, b <- n] such that  a >> b  and  m >mls> (n \\ [b])
```

An interesting method to study for reduction can be converting to ground BT-s, applying the Gröbner basis method for reduction, and converting back.

So far, we use the reduction of BT by the multiset ordering.

3.4 Performing the example

```
-----------------------------------------------------------------------------
module Main
where
import Dumatel

bool_lpo = bool defaultVariableWeight defaultInitialOpId weightDecls lpo
                                                         where
                                                         weightDecls = []
groupHsi43 =
  addDeclarations 0 CheckOn SkolemizeNot Ordinary (\ _ _ _ -> Superposable)
                (superposabilityToCalculus Unsuperposable bool_lpo)
  $
  CalculusDeclaration
  {sortOpDecls =
         [Sorts ["G"],
          Op " e  []  []  G    "  [],
          Op " i  []  [G] G    "  [ParsePreceds 0 7],
          Op " *  [G] [G] G    "  [ParsePreceds 5 5,
                                    OpAlgAttrs [Associative, LeftIdentity "e"]
                                  ],
          Op " P  []  [G] Bool "  [ParsePreceds 0 6]
         ],
   opPrecedDecls = [OpPrecedDescentName ["P", "i", "*", "e", "false"]],
   opWeightDecls = [],
   formulaDecls  = [],
   ruleDecls     = [],
   termCompDecl  = Nothing,   -- taken from  bool_lpo
   varDecls      = [" X Y Z :  G "],    -- to differ from x.. from bool_lpo

   eqDecls = [ "(X*Y)*Z = X*(Y*Z)",  "e*X = X ",   "(i X)*X = e" ]
  }

--------------------------------------------------------------------
main =
  putStr $ concat
  [
   let parse = parse1 groupHsi43 "calculus = groupHsi43"  :: String -> Formula

       axiomStr =  " (exist [X] (P X))  &                      "++
                   " (forall [X,Y] (P X & P Y ==> P (X*(i Y)))) "

       fF = parse (axiomStr ++ " ==>  P e  &  (forall [X] (P X ==> P (i X))) ")

       jump         = 0
```

```
    rF            = reduce jump groupHsi43 fF
    (nF, negCost) = negateFormula jump groupHsi43 rF

    (calcOfNegatedGoal, newEqs, _, _) =
                              addFormulae jump groupHsi43 skolemParams Preferable
                                         (\ _ _ _ -> Superposable ) [nF]

    skolemParams = Skolemize defaultSkolemParameters
    preced'      = opPrecedence calcOfNegatedGoal

    rcInitCompl  = Infinity
    rc           = rcInitCompl
    rcPerSubcase = 0
    --
    -- that is prove the goal by negation and completion, with
    -- spending the infinite resource only to completion,
    -- without considering 'cases'

    res = proveByNegationAndCompletion
          TraceNot jump rc rcInitCompl rcPerSubcase skolemParams
          Preferable (\ _ _ _ -> Superposable) groupHsi43 rF

    complRes = cpRefInitialCompletion res
 in
 concat
 ["\n\n fF =\n",                show fF,
  "\n\n reduced fF = rF  =\n",  show rF,
  "\n\n negFormulaLaws   =\n",  show $ formulaLaws $ laws calcOfNegatedGoal,
  "\n\n neg_btLaws       =\n",  show $ btLaws $ laws calcOfNegatedGoal,
  "\n\n newEqs           =\n",  show newEqs,
  "\n\n ukbbGoalRem      =  ",  show $ ukbbGoalRem complRes,
  "\n\n Completion trace =\n",  shows (ukbbTrace complRes) ""
 ]
 ,
 "\n"
]
where
parse1 calc           = parseSingleOrBreak calc
sg_bool_lpo           = signature bool_lpo
otab_bool_lpo         = opTab sg_bool_lpo
fromSortName_bool_lpo = fst $ sortIdTabs sg_bool_lpo
nameToOp_bool_lpo     = nameToOpTab  otab_bool_lpo
preced_bool_lpo       = opPrecedence otab_bool_lpo
aA_bool_lpo           = equationList bool_lpo

sg_hsi43           = signature        groupHsi43
```

```
  sotab_hsi43        = sOpTab           sg_hsi43
  otab_hsi43         = opTab            sg_hsi43
  preced_hsi43       = opPrecedence     otab_hsi43
  nameToOp_hsi43     = nameToOpTab      sg_hsi43
  fromSortName_hsi43 = fst $ sortIdTabs sg_hsi43
  aA_hsi43           = equationList     groupHsi43
```

Below we use the abbreviations:
"Chosen e" means "Chosen equations after reduction",
"Chosen b" means "Chosen b-terms after reduction".

We mark with * the facts most essential for the inference.

```
-----------------------------------------------------------------------------
fF = ( (exist [X] (P X))  &  (forall [X,Y] (P X & P Y ==>  P (X*(i Y)))) )
     ==>
     ( (P e) & (forall [X] (P X ==> P i X)) )

reduced fF = rF =

 (not (exist [X] (P X)))                                      |
 (not (forall [X,Y] (not P X | (not P Y | (P (X*(i Y)))))) )  |
 (P e & (forall [X] (not P X | (P i X))))

negFormulaLaws =
 [exist  [X]   (P X),                                             -- 1
  forall [X,Y] (not P X | (not P Y | P (X*(i Y)))),               -- 2
  not P e  |  (exist [X] (P X & not P i X))                       -- 3
 ]
neg_btLaws = [ (P (X*(i Y))) & (P Y) & (P X)  xor  (P Y) & (P X),    -- 2'

               (P XSk) & (P e)  xor  (P e),                       -- 3.1'

               (P i XSk) & (P e)                                  -- 3.2'
             ]
newEqs =     [P XSk0 -> true]                          -- converted from (1)


ukbbGoalRem =  ([], [])     -- proved!

Completion trace =
[
(Pre-reduction in completion. The number of rules is  26.
The number of equations and BT
in the initial calculus are respectively  31, 5;
after reduction:                          31, 10.
```

```
The reduced goals are                     ([(true, false)], [])
),
(Chosen e  [29] x ~ x -> true           Unsup Od (P [])),
(Chosen e  [30] true ~ false -> false   Unsup Od (P [])),
(Chosen e  [31] false ~ true -> false   Unsup Od (P [])),

(Chosen b  [42] (P i XSk)&(P e)         Pref (P [])),           -- *

(Chosen e  [27] P XSk012 -> true        Sup Pref (P [])),       -- parasitic

--  XSk, XSk0  are necessary.  Other  *Sk*  are parasitic.

(Chosen e  [28] P XSk0 -> true          Sup Pref (P [])),       -- *

(Chosen b  [44] (P i XSk01)&(P e)       Pref (P [])),           -- parasitic

(Chosen e  [25] e*X -> X                Sup Od (P [])),
(Chosen e  [24] (i X)*X -> e            Sup Od (P [])),
(Chosen e  [26] (X*Y)*Z -> X*(Y*Z)      Sup Od (P [])),

(Chosen e  [50] (i X)*(X*Z) -> Z        Sup Od (P [[26], [24]])),   --
(Chosen e  [51] (i e)*X -> X            Sup Od (P [[50], [25]])),
(Chosen e  [55] (i i e)*X -> X          Sup Od (P [[51], [50]])),

(Chosen e  [57] i e -> e                Sup Od (P [[55], [24]])     --
),
(The following facts have been newly reduced,
or subsumed, and moved to delayed set:         [[51], [55]]
),
(Chosen e  [53] (i i X)*e -> X            Sup Od (P [[50], [24]])
),
(Chosen b  [41] (P XSk)&(P e) xor (P e)   Pref (P [])               -- *
),
(Chosen e  [58] (i i i X)*X -> e          Sup Od (P [[53], [50]])
),
(Chosen e  [60] i i X -> X                Sup Od (P [[58], [50]])   --
),
(The following facts have been newly reduced,
or subsumed, and moved to delayed set:        [[53], [58]]
),
(Chosen e  [53] X*e -> X                Sup Od (P [[50], [24]])     --
),
(Chosen e  [63] X*(i X) -> e            Sup Od (P [[60], [24]])     -- *
),
(Chosen b  [43] (P XSk01)&(P e) xor (P e)   Pref (P [])             -- parasitic
),
```

```
(Chosen e  [62] X*((i X)*Z) -> Z        Sup Od (P [[60], [50]])
),
(Chosen e  [65] X*(Y*(i (X*Y))) -> e    Sup Od (P [[63], [26]])
),
(Chosen e  [67] Y*(i ((i X)*Y)) -> X    Sup Od (P [[65], [62]])
),
(Chosen e  [74] Y*(i (X*Y)) -> i X      Sup Od (P [[67], [60]])
),
(The following facts have been newly reduced,
or subsumed, and moved to delayed set:         [[67], [65]]
),
(Chosen e [84] (i (X*Y))*X -> i Y      Sup Od (P [[74], [74]])
),
(Chosen e [82] i (X*(i Y)) -> Y*(i X)  Sup Od (P [[74], [62]])
),
(Chosen e [83] i (X*Y) -> (i Y)*(i X)  Sup Od (P [[74], [50]])             --
),
(The following facts have been newly reduced,
or subsumed, and moved to delayed set:         [[74], [84], [82]]
),

(Chosen b  [40] (P (X*(i Y)))&(P Y)&(P X) xor (P Y)&(P X)   Pref (P [])    -- *
),
-- recall  [63] X*(i X) -> e,  apply  Y := X,  and derive
--
(Chosen b  [106] (P X0)&(P e) xor (P X0)          Pref (P [[63], [40]])    -- *
),

(The following facts have been newly reduced,
or subsumed, and moved to delayed set:        [[42], [44], [41], [43]]
),
(Chosen e  [44] P i XSk01 -> false      Sup Pref (P [])
),
(Chosen e  [42] P i XSk -> false        Sup Pref (P [])                    -- *
),

-- recall given  [28]  P XSk0 -> true,
-- recall        [106] (P X0)&(P e)  xor  (P X0),   apply  X0 := XSk0  and derive
--
(Chosen e  [111] P e -> true            Sup Pref (P [[28], [106]])       -- *  goal 1
),
(The following facts have been newly reduced,
or subsumed, and moved to delayed set:        [[106]]
),
(Chosen e  [43] P XSk01 -> true            Sup Pref (P [])
),
```

```
(Chosen e  [41] P XSk -> true             Sup Pref (P [])
),

-- recall  [111] P e -> true,                                 put X := e  in the given
--         [40] (P (X*(i Y)))&(P Y)&(P X)  xor  (P Y)&(P X)             and derive
--
(Chosen b  [117] (P i Y)&(P Y) xor (P Y)   Pref (P [[111], [40]])
),


-- recall given  [41] P XSk -> true,  put  Y := XSk  in [117],  derive
--                                                               (P i XSk) xor 1,
-- and reduce by [42] to
--
(Chosen b  [128] 1                    Pref (P [[41], [117]])        -- contradiction!
),

(The following facts have been newly reduced,
or subsumed, and moved to delayed set:        [[117], [40]]
),
(Chosen e  [128] false -> true               Sup Pref (P [[41], [117]])
),
The number of goals has reduced to  0
,
(The following facts have been newly reduced,
or subsumed, and moved to delayed set:
[[22],[21],[15],[16],[10],[9],[4],[5],[14],[30],[31],[42],[44]]
)]
--------------------------------------------------------------------------------
```

4 Proof by completion plus “cases”

Example 3. A simple problem for ordered lists.

The following specification defines the calculus for lists (of the domain `List`) over any domain `Elt`.

Then, there follows the function `main`, which presents the proof search for the formula

```
  forall [X,Y,Z] (isOrdered insert Z insert Y (X : nil))
```

```
------------------------------------------------------ proofs for ordering lists
module Main
where
import Dumatel

bool_lpo = bool defaultVariableWeight defaultInitialOpId weightDecls lpo
                                                         where
                                                         weightDecls = []
(list, eqsFromFormulae, _, _) =

  addFormulae 0 preList SkolemizeNot Ordinary (\ _ _ _ -> Sup) formulae
  where
  formulae =
     map (parseSingleOrBreak preList "list calculus")
     [
      "forall [X,Y] (X eq_Elt Y | X > Y | Y > X)",
      "forall [X,Y] (X > Y ==> not (Y > X))",       -- antisymmetry
      "forall [X,Y] (X > Y ==> not (X eq_Elt Y))"

      -- But skip transitivity!  (X > Y & Y > Z ==> X > Z)
      --
      -- A problem:  transitivity complicates completion. How to handle it?
     ]

  preList =
    (\ calc -> addEquations calc $ rulesFromCalculusToEquations calc)
    $
    addDeclarations 0 CheckOn SkolemizeNot Ordinary (\ _ _ _ -> Sup)
                    (superposabilityToCalculus Unsuperposable bool_lpo)
    $
    CalculusDeclaration
    {sortOpDecls =
     [
      Sorts ["Elt",      -- sort for the list elements
             "List"
            ],
```

```
 SortGen "List" [["nil", ":"]],

 Op " nil  []     []     List "  [],
 Op " :    [Elt]  [List] List "  [ParsePreceds 5 5],   -- prepend
 Op " +    [List] [List] List "  [ParsePreceds 5 5],   -- concatenation
 Op " and  [Bool] [Bool] Bool "
                       [ParsePreceds 5 5, OpAlgAttrs [Associative, Commutative,
                                                      LeftIdentity  "true"]
                       ],
 Op " >  [Elt] [Elt] Bool "  [eqPPreceds],

 Op " a  [] [] Elt "  [],  -- some element constants
 Op " b  [] [] Elt "  [],  -- to make the list examples
 Op " c  [] [] Elt "  [],  --

 Op " eq_Elt [Elt] [Elt] Bool "    [eqPPreceds, OpAlgAttrs [Commutative]],
 Op " eq_List [List] [List] Bool " [eqPPreceds, OpAlgAttrs [Commutative]],

 Op " insert []  [Elt List]          List " [ParsePreceds 0 7],
 Op " ins    []  [Elt Elt List Bool] List " [ParsePreceds 0 7],
 Op " isOrdered [] [List]       Bool "      [ParsePreceds 0 7],
 Op " isOrd     [] [Bool, Bool] Bool "      [ParsePreceds 0 7]
],
opPrecedDecls = [OpPrecedDescentName
                 ["insert", "ins", "isOrdered", "isOrd", "eq_List", "+",
                  ":", "nil", ">", "eq_Elt", "and",
                  "a", "b", "c",
                  "false"
                ]],
opWeightDecls = [],
termCompDecl  = Just lpo,
varDecls      = ["X Y Z X' Y' :  Elt",
                 "Xs Ys Zs :     List",    "bo : Bool"
                ],
formulaDecls = [],
eqDecls      = [],  -- equations are made later, from rules
ruleDecls    =
  [
   " X eq_Elt X -> true  ",
   " a eq_Elt b -> false ",  " a eq_Elt c -> false ",
   " b eq_Elt a -> false ",  " b eq_Elt c -> false ",
   " c eq_Elt a -> false ",  " c eq_Elt b -> false ",

   " Xs eq_List Xs -> true ",

   " false and x     -> false ",
```

```
        " x     and false -> false ",
        " true  and x     -> x     ",
        " x     and true  -> x     ",

        " nil    + Ys = Ys            ",
        " (X:Xs) + Ys = X : (Xs + Ys) ",
        -----------------------------------------
        " X > X -> false ",
        " a > b -> false ",  " a > c -> false ",
        " b > a -> true  ",  " b > c -> false ",
        " c > a -> true  ",  " c > b -> true  ",
        -----------------------------------------------------------
        " isOrdered nil      -> true ",
        " isOrdered (X:nil)  -> true ",
        " isOrdered (X:Y:Ys) -> isOrd (X > Y) (isOrdered (Y:Ys)) ",

          " isOrd true  bo -> false ",
          " isOrd false bo -> bo    ",

        " insert X nil      -> X : nil            ",
        " insert X (Y : Xs) -> ins X Y Xs (X > Y) ",

          " ins X Y Xs true  ->  Y : (insert X Xs) ",
          " ins X Y Xs false ->  X : Y : Xs        "
       ]
    }
    where
    eqPPreceds = ParsePreceds l r  where
                                (l, r) = defaultEqualityOperatorParsePrecedences

main =
  putStr $ concat
  [
   "Goal formula =  ",    show fF,
   "\n\n",
   "is proved = ",        show $ isProved res,
   "\n",
   "rcRem = ",            show $ cpRefRcRem res,
   "\n",
   "refuted cases =\n",   showsList ",\n" (cpRefRefutedCases res)
   "\n\n",
   "remaining cases =\n", showsList ",\n" (cpRefRemCases res)
   "\n"
  ]
  where
  parse1 = parseSingleOrBreak th "list calculus"
```

```
th     = list

superpty _ _ _ = Superposable
jump           = 2

fF = parse1 " forall [X,Y,Z] (isOrdered insert Z insert Y (X : nil)) "

rc              = 4*10^5
rcInitialCompl  = Fin (10^3)
rcPerSubcase rc = quot rc 64

res = proveByNegationAndCompletion
      Trace jump (Fin rc) rcInitialCompl (rcPerSubcase rc)
      (Skolemize defaultSkolemParameters) Preferable superpty th fF
```

4.1 Domain enumeration and its subvariety of models

The above theorem is tried to prove `InVariety`: for all models.

If this example is done only by completion, as this was with **Example 2**, then the prover fails to prove this formula.

Indeed, this formula is not true *in variety*!
— if nothing is specified about finiteness of the domains involved.

Because, for example, `(YSk > XSk)` has values in `Bool`, and nowhere is specified that `Bool` has only two values in it!

More example: consider a domain `D` with the only values of `A1, A2, A3`.
The enumeration axiom could be

```
X ~ A1  or  X ~ A2  or  X ~ A3  = true
```

For technical reasons, we choose another way:

introduce the *finite enumeration* denotation to the data construct of `Calculus`.

For example, `bool_lpo` includes the declaration like

```
FiniteEnumeration Bool [true, false].
```

Meaning: the prover needs consider for this calculus only the corresponding finite models.

This provides the correctness for the proof by splitting to the *cases*.

Let us call the chosen terms `e` *case-terms*.

The proof must consider a logically complete set of cases.

It makes sense to select only a small set of case-terms.

Our prover extracts the case-terms out of the previous failed proof attempt.

In a refutational proof, these are certain ground terms made from the Skolem constants,
the subsignatures to form the case-terms are taken by certain heuristics of the relationship to equations.

4.2 Performing the example with 'cases'

Put the enumeration for `Bool` and perform the example.

```
----------------------------------------------------------------------
Pre-reduction done. Initial numbers of equations and BTs are
57 and 3,  after reduction:  57 and 6.

Completion stops by resource exhaustion.

Starting final mutual reduction for UKBBResult.
The number of rules is  56.
Numbers of equations and BTs before reduction are  23 and 0,
after reduction:                                   23 and 0.
Non-trivial jump reduxes are  [].

Initial refutation by completion has failed.
But there are some 'cases' to consider for ground terms.
Starting to refute by cases.

---------------------------------------------
refuteByCompletionAndCasesForGroundTerms Trace 2 superpty 6250 calc state,
rc = (Fin 396465)

enumPairs =  [(YSk > XSk,  [true, false])]

refuteCases:  refuting the case  [(YSk > XSk, true)].
Resource per subcase is  6250,
the total remaining resource is  Fin 396465.

Failed to refute the subcase. Additional support =  {(ZSk > XSk)}.
Failed to refute the subcase. Additional support =  {(ZSk > YSk)}.

The case of   [(YSk > XSk, true), (ZSk > XSk, true), (ZSk > YSk, true)]
is refuted!
refuteCases:  refuting the case
              [(YSk > XSk, true), (ZSk > XSk, true), (ZSk > YSk, false)].

Resource per subcase is  6250,
the total remaining resource is  Fin 338485.
```

```
Failed to refute the subcase. Additional support =  {}.
Failed to refute the subcase. Additional support =  {}.

The case of  [(YSk > XSk, true),(ZSk > XSk, true),(ZSk > YSk, false)]
is refuted!
refuteCases:  refuting the case
              [(YSk > XSk, true),(ZSk > XSk, false),(ZSk > YSk, true)].

Resource per subcase is  6250,
the total remaining resource is  Fin 289508.

Failed to refute the subcase. Additional support =  {}.

The case of  [(YSk > XSk, true), (ZSk > XSk, false)]
is refuted!
refuteCases:  refuting the case
              [(YSk > XSk, false), (ZSk > XSk, true), (ZSk > YSk, true)].

Resource per subcase is  6250,
the total remaining resource is  Fin 261757.

Failed to refute the subcase. Additional support =  {}.
Failed to refute the subcase. Additional support =  {}.

The case of  [(YSk > XSk, false),(ZSk > XSk, true),(ZSk > YSk, true)]
is refuted!
refuteCases:  refuting the case
              [(YSk > XSk, false), (ZSk > XSk, false), (ZSk > YSk, true)].

Resource per subcase is  6250,
the total remaining resource is  Fin 213664.

Failed to refute the subcase. Additional support =  {}.
Failed to refute the subcase. Additional support =  {}.

The case of  [(YSk > XSk, false), (ZSk > XSk, false), (ZSk > YSk, true)]
is refuted!
refuteCases:  refuting the case
              [(YSk > XSk, false), (ZSk > XSk, true), (ZSk > YSk, false)].

Resource per subcase is  6250,  t
he total remaining resource is  Fin 168281.

Failed to refute the subcase. Additional support = {}.
Failed to refute the subcase. Additional support = {}.
```

```
The case of  [(YSk > XSk, false), (ZSk > XSk, true), (ZSk > YSk, false)]
is refuted!
refuteCases:  refuting the case
             [(YSk > XSk, false), (ZSk > XSk, false), (ZSk > YSk, false)].

Resource per subcase is  6250,
the total remaining resource is  Fin 120300.

Failed to refute the subcase. Additional support = {}.
Failed to refute the subcase. Additional support = {}.

The case of  [(YSk > XSk, false), (ZSk > XSk, false), (ZSk > YSk, false)]
is refuted!


Goal formula =  forall [X,Y,Z] (isOrdered (insert Z (insert Y (X:nil))))

is proved     = True
rcRem         = Fin 73439
refuted cases =
[
[(YSk > XSk, false), (ZSk > XSk, false), (ZSk > YSk, false)],
[(YSk > XSk, false), (ZSk > XSk, true ), (ZSk > YSk, false)],
[(YSk > XSk, false), (ZSk > XSk, false), (ZSk > YSk, true )],
[(YSk > XSk, false), (ZSk > XSk, true ), (ZSk > YSk, true )],
[(YSk > XSk, true ), (ZSk > XSk, false)],
[(YSk > XSk, true ), (ZSk > XSk, true ), (ZSk > YSk, false)],
[(YSk > XSk, true ), (ZSk > XSk, true ), (ZSk > YSk, true )]
]

remaining cases = []
```

References

[B:D:P] L. Bachmair, N. Dershovitz, D. Plaisted.: *Completion without failure.* In Ait-Kaci (Editor): Resolution of Equations in Algebraic Structures, Volume 2, pages 1-30, Academic Press, 1989.

[Bu] B. Buchberger. *Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory.* CAMP. Publ. No.83–29.0 November 1983.

[Hsi] J. Hsiang. *Refutational theorem proving using term-rewriting systems.* Artificial Intelligence, 1985, v.25, p.255-300.

[Hsi:Ru] J. Hsiang, M. Rusinowitch. *On word problems in equational theories.* In Th. Ottman (ed.), Proceedings of the Fourteenth International Conference on Automata, Languages and Programming, Karlsruhe, West Germany, July 1987, Springer Verlag, Lecture Notes in Computer Science **267**, (54–71), 1987.

[KB] D. Knuth, P. Bendix: *Simple word problems in universal algebras.* In John Leech, editor, "Computational Problems in Abstract Algebra", (263–297), Pergamos Press, 1970.

[Lo:Hi] B. Löchner, T. Hillenbrand: *A Phytography of Waldmeister.* AC Communications **(15)** (2,3) (2002) (127–133).