# Provers

September 7, 2011

## 1 Minlog

Formalization by Helmut Schwichtenberg <schwicht@mathematik.uni-muenchen.de>.
Questions answered by Helmut Schwichtenberg and Ulrich Berger <U.Berger@swansea.ac.uk>.

### 1.1 Statement

```
all x,p,q.2===x*x -> x===p#q -> F
```

### 1.2 Definition

```
(add-ids
 (list (list "RealEq" (make-arity (py "real") (py "real"))))
 '("all x,y.all k abs(x seq(x mod(k+1))-y seq(y mod(k+1)))<=(1#exp 2 k) ->
            RealEq x y"))

add-token
 "==="
 'pred-infix
 (lambda (x y)
   (make-predicate-formula (make-idpredconst "RealEq" '() '()) x y)))

(add-idpredconst-display "RealEq" 'pred-infix "===")
```

### 1.3 Proof

```
(load "~/minlog/init.scm")
(mload "../lib/numbers.scm")

(set-goal (pf "all p,q.SZero(q*q)=p*p -> F"))
(ind)
(auto)
(assume "p" "IHp")
(cases)
```

```
(auto)
(save "LemmaOne")

(set-goal (pf "all p,q.2==(p#q)*(p#q) -> F"))
(use "LemmaOne")
(save "LemmaOneRat")

(aga ;for add-global-assumption
  "RealRatTimesComp"
  (pf "all x1,x2,a1,a2.x1===a1 -> x2===a2 -> x1*x2===a1*a2"))
(aga "RatRealEq" (pf "all a,b.a===b -> a==b"))
(aga "RealEqTrans" (pf "all x,y,z.x===y -> y===z -> x===z"))

(set-goal (pf "all x,p,q.2===x*x -> x===p#q -> F"))
(strip)
(use "LemmaOneRat" (pt "p") (pt "q"))
(use "RatRealEq")
(use "RealEqTrans" (pt "x*x"))
(prop)
(use "RealRatTimesComp")
(auto)
(save "Corollary")
```

### 1.3.1 Proof Terms

Lemma 1:

```
((((|Ind| (lambda (q) (lambda (u55) u55)))
   (lambda (p)
     (lambda (|IHp58|)
       (((((|Cases| p) (lambda (u62) u62))
         (lambda (n290) (lambda (u63) ((|IHp58| n290) u63))))
        (lambda (n291) (lambda (u64) u64))))))
  (lambda (n286)
    (lambda (u65) (lambda (q) (lambda (u66) u66)))))
```

Corollary:

```
(lambda (x)
   (lambda (p)
     (lambda (q)
       (lambda (u69)
         (lambda (u70)
           ((((|LemmaOneRat| p) q)
            ((((|RatRealEq| 2)
```

```
 (* ((|RatConstr| p) q) ((|RatConstr| p) q)))
((((((|RealEqTrans|
      ((|RealConstr| (lambda (n300) 2))
       (lambda (n300) 1)))
    ((|RealTimes| x) x))
   ((|RealConstr|
      (lambda (n299)
        (* ((|RatConstr| p) q)
           ((|RatConstr| p) q))))
    (lambda (n299) 1)))
  u69)
 ((((((|RealRatTimesComp| x) x)
     ((|RatConstr| p) q))
    ((|RatConstr| p) q))
   u70)
  u70)))))))))
```

### 1.4 Proof of Lemma 1, using unary numbers

```
(load "~/minlog/init.scm")
(mload "../lib/nat.scm")

; "Even" and "Odd"
(add-program-constant "Even" (py "nat=>boole") 1)
(add-program-constant "Odd" (py "nat=>boole") 1)

(add-computation-rule (pt "Even 0") (pt "True"))
(add-computation-rule (pt "Odd 0") (pt "False"))
(add-computation-rule (pt "Even(Succ n)") (pt "Odd n"))
(add-computation-rule (pt "Odd(Succ n)") (pt "Even n"))

; "Double"
(add-program-constant "D" (py "nat=>nat") 1)

(add-computation-rule (pt "D 0") (pt "0"))
(add-computation-rule
  (pt "D(Succ n)") (pt "Succ(Succ(D n))"))

; "Half"
(add-program-constant "H" (py "nat=>nat") 1)

(add-computation-rule (pt "H 0") (pt "0"))
(add-computation-rule (pt "H 1") (pt "0"))
(add-computation-rule
```

```
  (pt "H(Succ(Succ n))") (pt "Succ(H n)"))

; "CvInd"
(set-goal
  (pf "(all n.(all m.m<n -> Q m) -> Q n) -> all n Q n"))
(assume "Prog")
(cut (pf "all n,m.m<n -> Q m"))
(assume "QHyp")
(assume "n")
(use "QHyp" (pt "Succ n"))
(use "Truth-Axiom")
(ind)
(assume "m" "Absurd")
(use "Efq")
(use "Absurd")

(assume "n" "IHn")
(assume "m" "m<Succ n")
(use "LtSuccCases" (pt "n") (pt "m"))
(use "m<Succ n")
(use "IHn")
(assume "m=n")
(simp "m=n")
(use "Prog")
(use "IHn")
(save "CVInd")

; "LemmaOneAux"
(set-goal (pf "all n,m.n*n=D(m*m) -> m*m=D(H n*H n)"))
(assume "n" "m" "n*n=D(m*m)")
(simp "TimesDouble1")
(use "DoubleInj")
(simp "<-" "n*n=D(m*m)")
(simp "TimesDouble2")
(simp (pf "D(H n)=n"))
(use "Truth-Axiom")
(use "EvenOddDoubleHalf")
(use "EvenOddSquareRev")
(simp "n*n=D(m*m)")
(use "EvenDouble")
(save "LemmaOneAux")

; "LemmaOne"
(set-goal (pf "all n,m.n*n=D(m*m) -> n=0"))
```

```
(use-with
  "CVInd"
  (make-cterm (pv "m") (pf "all n.m*m=D(n*n) -> m=0")) "?")
(assume "n" "IHn" "m"  "n*n=D(m*m)")
(cases (pt "0<n"))
(assume "0<n")
(use "ZeroSquare")
(simp "n*n=D(m*m)")
(cut (pf "m=0"))
(assume "m=0")
(simp "m=0")
(use "Truth-Axiom")
(use "IHn" (pt "H n"))
(use "LtSquareRev")
(simp "n*n=D(m*m)")
(use "LtDouble")
(use "DoublePos")
(simp "<-" "n*n=D(m*m)")
(use "SquarePos")
(use "0<n")
(use "LemmaOneAux")
(use "n*n=D(m*m)")
(use "NotPosImpZero")
(save "LemmaOne")
```

### 1.5 System

*What is the home page of the system?*

<http://www.minlog-system.de>

*What are the books about the system?* There is no book on Minlog at present. However, the paper "Minimal Logic for Computable Functions"

<http://www.mathematik.uni-muenchen.de/~minlog/minlog/mlcf.ps>

describes the logical basis of the system. There is a reference manual

<http://www.mathematik.uni-muenchen.de/~minlog/minlog/ref.ps>

and a tutorial (by Laura Crosilla)

<http://www.mathematik.uni-muenchen.de/~minlog/minlog/tutor.ps>

*What is the logic of the system?* Minimal logic, hence (via ex-falso-quodlibet and stability axioms) also intuitionistic and classical logic are

included. Quantification over higher order functionals (of any finite type) is possible. There are also (parametric) type variables and predicate variables, but they are seen as place-holders for concrete types and comprehension terms, i.e., they are (implicitly) universally quantified (this is sometimes called ML-polymorphism). To keep the system predicative and moreover proof-theoretically weak (i.e., conservative over Heyting arithmetic), quantification over type and predicate variables is not allowed. Inductive data types and inductive definitions with their usual introduction and elimination rules are present. Of course, the latter in general increases the proof-theoretic strength.

*What is the implementation architecture of the system?* The system is written in Scheme (a Lisp dialect). Types, terms, formulas and proofs are separate entities. Terms and (following the Curry-Howard correspondence) also proofs are seen as $\lambda$-terms. There is a simple proof checking algorithm, which guarantees the correctness of a generated proof object.

*What does working with the system look like?* The standard LCF tactic style is used. David Aspinall's "Proof General" interface has also been adapted (by Stefan Schimanski).

*What is special about the system compared to other systems?* The intended model consists of the partial continuous functionals (in the Scott-Ersov sense), so partiality is built in from the outset. Program extraction can be done from constructive as well as from classical proofs; the latter uses a refined form of Harvey Friedman's $A$-translation. A proof that an extracted program realizes its specification can be machine generated.

Function (or better: program) constants (again of any finite type) have user-defined computation and rewrite rules associated with them. Using Scheme's evaluation, terms are normalized ("normalization by evaluation") w.r.t. all standard conversion rules ($\beta$, $\eta$, recursion, user-defined computation and rewrite rules), and terms with the same normal form are identified. This feature can shorten proofs drastically, as seen in the example above. Rewrite rules are to be viewed as part of the proof object, but, in the current version of Minlog, are not kept as part of the proof term. The system supports working in proof-theoretically weak theories, in order to have control over the complexity of extracted programs (this is work in progress).

*What are other versions of the system?* There is only one supported version. The current one is 4.0.

*Who are the people behind the system?* The Munich (LMU) logic group, and the Swansea logic group.

*What are the main user communities of the system?* The main user communities are in Munich and in Swansea.

*What large mathematical formalizations have been done in the system?* The intermediate value theorem in constructive analysis; the usage of concrete representations of the reals (as above) allowed extraction of a usable program to compute $\sqrt{2}$ (20 binary digits in 10 ms). Higman's lemma

(Monika Seisenberger). Correctness of Dijkstra's and Warshall's algorithms. Program extraction from a classical proof of Dickson's lemma (Ulrich Berger).

*What representation of the formalization has been put in this paper?* In the proof of Lemma 1 and the statement about $\mathbb{R}$ using binary numbers, what is presented is the complete tactic file producing the proofs terms shown. In the proof of Lemma 1 using unary numbers, for brevity the (easy) proofs of the auxiliary lemmas have been left out. They should be part of the standard library, and can be found in the file `examples/arith/sqrttwo.scm` of the `minlog` directory.

*What needs to be explained about this specific proof?* The short proof of Lemma 1 (only five commands) is due to the fact that arguing with even and odd numbers in the context of a binary numbers is particularly simple (see below for more details). On the other hand, binary numbers are a must (for efficiency reasons) when working with rationals.

To provide a clearer picture of how working with the system is like in less fortunate circumstances, a proof of Lemma 1 for unary numbers is included as well.

The file `numbers.scm` contains definitions of the (binary) positive numbers `n`, `m`, `p`, `q` ... with constructors `One`, `SZero` and `SOne`, the integers `i`, `j` ... with constructors `IntPos`, `IntZero` and `IntNeg`, the rationals `a`, `b` ... seen as pairs `i#n` of an integer `i` and a positive natural number `n`, and the reals `x`, `y` ... seen as pairs of a Cauchy sequence of rationals and a modulus. Equality deserves special attention:

- `=` is the structurally defined equality for positives and integers,
- `==` is the decidable (equivalence) relation on the rationals, and
- `===` is the (undecidable) equivalence relation on (the chosen representation of) the reals.

Here are some more details on the proofs.

*Lemma 1 (binary)*: $\forall p, q.\mathsf{S}_0(q * q) = p * p \to \mathsf{F}$. Since the proof is by induction on binary numbers the following sub-goals are created (writing $1, \mathsf{S}_0, \mathsf{S}_1$ for `One`, `SZero`, `SOne` respectively):

?2 $\forall q.\mathsf{S}_0(q * q) = 1 * 1 \to \mathsf{F}$

?3 $\forall p.\mathsf{IH}(p) \to \forall q.\mathsf{S}_0(q * q) = \mathsf{S}_0(p) * \mathsf{S}_0(p) \to \mathsf{F}$

?4 $\forall p.\mathsf{IH}(p) \to \forall q.\mathsf{S}_0(q * q) = \mathsf{S}_1(p) * \mathsf{S}_1(p) \to \mathsf{F}$

where $\mathsf{IH}(p) \equiv \forall q.\mathsf{S}_0(q * q) = p * p \to \mathsf{F}$.

Goal ?2 is solved by normalizing $1 * 1$ to 1 and hence $\mathsf{S}_0(q * q) = 1 * 1$ to $\mathsf{F}$. Similarly, in goal ?4 the premise $\mathsf{S}_0(q * q) = \mathsf{S}_1(p) * \mathsf{S}_1(p)$ normalizes to $\mathsf{F}$ since $\mathsf{S}_1(p) * \mathsf{S}_1(p)$ normalizes to a term of the form $\mathsf{S}_1(\_)$. The premise $\mathsf{S}_0(q * q) = \mathsf{S}_0(p) * \mathsf{S}_0(p)$ of goal ?3 normalizes to $q * q = \mathsf{S}_0(p * p)$. Consider

the possible forms of $q$: 1, $\mathsf{S}_0(r)$, or $\mathsf{S}_1(r)$. In the first and last case $q * q$ normalizes to a term of the form $\mathsf{S}_1(\_)$ hence the equation normalizes to $\mathsf{F}$. In the second case the equation normalizes to $\mathsf{S}_0(q * q) = p * p$ which implies $\mathsf{F}$ by the induction hypothesis. The system does all this fully automatically, except for the case analysis on $q$.

*Lemma 1 Rat*: $\forall p, q.2 == \frac{p}{q} * \frac{p}{q} \to \mathsf{F}$. Up to normalization this is identical to Lemma 1, since $2 == \frac{p}{q} * \frac{p}{q}$ normalizes to $\mathsf{S}_0(q * q) = p * p$.

*Global assumption on equalities on the rationals and the reals.* Here the fact is used that the rationals are coerced into the reals.

*Lemma 1 (unary).* The 12th command, `(use "IHn" (pt "H n"))`, reduces the goal $m = 0$ with the help of the induction hypothesis $\forall m.m < n \to \forall k.m * m = D(k * k) \to m = 0$. The argument `(pt "H n")` is *not* –as one might think– used to instantiate the first universal quantifier, $\forall m$, but the second one, $\forall k$, which is the only quantifier left uninstantiated after matching the goal with the head of the induction hypothesis.