Goedel's Proof Programmed with C Macros

Jens Doll
University of Hamburg
jens.doll@ studium.uni−hamburg.de

Coding of Logic

Ground Set $\mathbb{N} = \{1,2\ldots\infty\}$

Functions, multivariate	$x_1 + x_2$	(*)
Formulas	$x_2 = 2\,x_1$	(*)
Propositions	$(x_1 = 2 * x_2) \vee \neg\,(2 = 3 * x_2)$	
Quantified Propositions	$\forall x_1.(x_1+1)*(x_1-1) = x_1 * x_1 - 1$	
	$\exists x_1.x_2 + 1 = x_1$	

(*) *expressions like this are not fully defined in the original paper, but feasable*

Alphabet

Alphabet	$\mathbb{A} = \{0\ 1\ \mathit{succ}\ (\ )\ x_1^i \ldots x_n^i = \exists\ \forall\ \}$	
	$(0 = \mathit{false}\ 1 = \mathit{true})\ \ (\mathit{also}\ 1 \in \mathbb{N})$	
Terms	$\subseteq\ \mathbb{N}^m \rightarrow \mathbb{N}$	$m \in [1,4]$
	$\subseteq\ \mathbb{N}^m \rightarrow \mathbb{B}$	"
	$\subseteq\ \mathbb{B}^m \rightarrow \mathbb{B}$	"
Syntax	*regular?*	*Yes, it's a term language with a finite number of functions!*
Axioms	*coded in own language, see above* *predicate x is an axiom*	

Coding of Characters

0	$succ$	$\neg$	v
1	3	5	7
$\forall$	$($	$)$	
9	11	13	
x_1	x_2	…	
17	19		
class x_1	class x_2	…	
17^2	19^2		
…			

Some propositions

$Sequence$	$\approx$ *indexed set*
$\mid$	*concatenation of sequences*
$FV(x_1,x_2)$	x_1 *is a free variable in sequence* x_2
$BV(x_1,x_2)$	x_1 *is a bound variable in sequence* x_2
$Sub(a,b,e,r)$	*substitute a by b in e yielding r*
$Ax(x_1)$	*sequence* x_1 *is an axiom*
$\ldots$	

Programming

$\exists$ *bounded existence,* $x_i \in [low, high]$

$\forall$ *bounded for all,* $x_i \in [low, high]$

Therefore and because of ground set $\mathbb{N}$

all propositions are finitely evaluable!

$\Rightarrow$ *propositions 1 ... 45 can be enumerated in for loops with a stack (for Pr and n!)*

Proposition Macros

NAT   *natural numbers arithmetic*

```
#define EXISTS(v,low,high,cond,res) \
 {NAT v; for(v=low;Le(v,high);v=Add(v,ONE)) { \
   if (cond) RESULT(res)}}
```

```
#define FORALL(v,low,high) \
 {NAT v; for (v=low;Le(v,high);v++) {
```

… <proposition>

```
#define ENDFOR }}
```

Prover Systems

Code for the Proof Assistants (like Isabelle) can be generated from the 46 macros.

These macros

```
BOOLFUN1(Prim,x,"x is a prime number")
  CASE(Eq(x,ONE),TRUE)
  EXISTS(z,TWO,x,And(Ne(z,x),divisible(x,z)),FALSE);
  RESULT(TRUE)
FIN
```

can generate

```
Lemma “Prim n” “ALL n::nat.NOTEX m::nat.((m<n)
                               AND (NOT divisible(n,m)))”;
```

Function Macros

```
#define BOOLFUN1(f,p1,man) \
 void MAN##f(void) {manpage(#p1,man);}   \
 BOOL f(NAT p1) { \
 BOOL erg=FALSE; long funlevel=level; \
 startout(#f);startout(p1); {
   ...
 FIN
```

```
#define NATFUN1(f,p1,man) \
 void MAN##f(void) {manpage(#p1,man);}   \
 NAT f(NAT p1) { \
 NAT erg=ZERO;  long funlevel=level; \
 startout(#f);argout(p1); {
  ...
 FIN
```

Sketch of Proof

1...42	prim. recursive functions (NAT, BOOL, for loops)	
3,4	recursive functions, all others are enumerations	
43	$Fl(x_1,x_2,x_3)$	$x_1 \wedge x_2 \rightarrow x_3$
44	$Bw(x_1)$	x_1 is a formal proof
45	$Bew(x_1,x_2)$	x_1 proves x_2
46	unknown if prim. recursive	

$Bew(x_1)$ <=> $\exists$ proof $x_2 \in \mathbb{A}^*$ for sequence x_1

x_1: proposition, sequence of chars $\in \mathbb{A}^*$

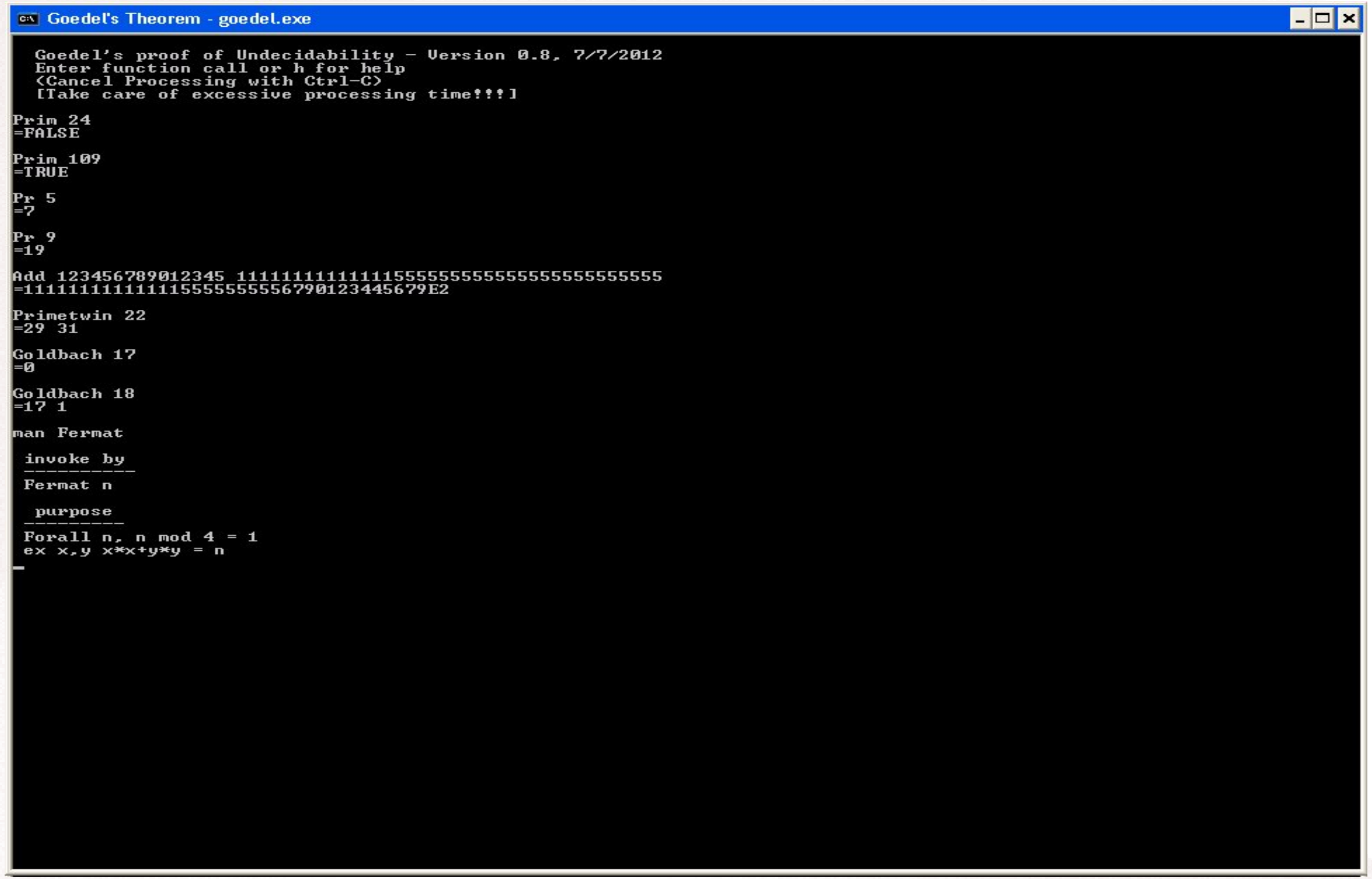
Live - Help Function

Goedel's Theorem - goedel.exe

```
 Goedel's proof of Undecidability - Version 0.8, 7/7/2012
 Enter function call or h for help
 <Cancel Processing with Ctrl-C>
 [Take care of excessive processing time!!!]
h

        ------------------------------------------------------------
         Functions of Goedel's Incompleteness Proof
        ------------------------------------------------------------
         parameters are naturals or res = <result>
         tr or /tr in command sets trace on/off
         spec displays C functions
         man <fun> displays help on a function
         q or . finishes
        ------------------------------------------------------------
         Possible function calls are:
          divisible x y Prim x          Pr n x
          Fac n         Pr n            Gl n x
          l x           Join x  y       R x
          E x           Var n x         Var x
          Neg  x        Dis x y         Gen x y
          N n x         Z n             Typ x
          Typ n x       Elf x           Op x y z
          Fr x          Form x          Geb v n x
          Fr v n x      Fr v x          Su x n y
          St k v x      A v x           Sb k x v y
          Sb k x v y    Imp x y         Con x y
          Aeq x y       Ex v y          Th n x
          Z_Ax x        A1_Ax n x       A_Ax x
          Q z y v       L1_Ax x         L2_Ax x
          R_Ax x        M_Ax x          Ax x
          Fl x y z      Bw x            B x y
          Bew x
        ------------------------------------------------------------
         hint: from Gl on processing lasts almost infinite
        ------------------------------------------------------------
         basic natural arithmetic functions are:
          Add x y       Sub x y         Mul x y
          Div x y       Pow x y         Mod x y
        ------------------------------------------------------------
        relations can be built by:
          Eq a b        Ne a b  Lt a b  Le a b  Ge a b  Gt a b
        ------------------------------------------------------------
         furthermore the following functions are available:
          Goldbach n    Primetwin n     Collatz n       Fermat n
        ------------------------------------------------------------
         questions may be sent to goedels@cococo.de
        ------------------------------------------------------------
```

Live – Simple Functions



Live – Specification

Goedel's Theorem - goedel.exe

```
spec
//-------------------------------------------
//       The following functions
//       have been derived from
//       Goedels 46+ propositions.
//
//   "On Formally Undecidable Propositions of
//    Principa Mathematics and Related Systems"
//       by Kurt Goedel, Vienna
//
//       Translated by B. Meltzer
//       Dover Publications 1962, 1992
//
//-------------------------------------------
//    EXISTS(v,low,high,cond,res)
//    := find first v>=low and v<=high
//       which satisfies cond andreturn res
//
//    FORALL(v,low,high,cond)
//    := do next check for all v>=low and v<=high
//
//-------------------------------------------
//  (c) Context IT GmbH, Ahrensburg, Germany
//      Web: http://www.cococo.de
//
//    Version: 0.8 July, 7th 2012
//    Modified: NN
//
//    Please send bug reports and feature
//    requests to goedels@cococo.de
//-------------------------------------------
#include "defs.nat.h"
#include "propositions.h"
//--------------------------------------
//   I.1 Axiom
//--------------------------------------
NATFUN1(AxiomI1,n,"there is no predecessor for ZERO")
  EXISTS(z,ONE,Add(n,ONE),Eq(n,Add(z,ONE)),FALSE);
  RESULT(FALSE)
FIN

//--------------------------------------
//   I.2 Axiom
//--------------------------------------
NATFUN2(AxiomI2,n,m,"if fx = fy then x = y")
  EXISTS(z,ONE,Add(n,ONE),Eq(n,Add(z,ONE)),FALSE);
  RESULT(FALSE)
FIN

//--------------------------------------
//   I.3 Axiom
//--------------------------------------
NATFUN1(AxiomI3,n,"Ex x2(0) && Forall(x1,x2(x1)>x2(f x1)) \n"
                  " => Forall(x1,x2(x1))")
  EXISTS(z,ONE,Add(n,ONE),Eq(n,Add(z,ONE)),FALSE);
  RESULT(FALSE)
FIN

//--------------------------------------
//   II.1 Axiom
//--------------------------------------
NATFUN2(AxiomII1,p,q,"p or q => p")
  CASE(Dis(p,q),Neg(p))
```

What has been done? What's planned?

Achieved

Implementation of functions 1..46
Execution of functions 1..7 for small n
Others partially with small n

Planned

- *Extension to 64bit (memory > 4GB)*
- *porting to fast machines (functions > 7)*
- *Optimize Loops, local and interprocedural*
- *Improve quality (CMMI 4)*

Recursive Functions

```
//--------------------------------------------
//  3. Function (2 parameters)
//--------------------------------------------

NATFUN2(Pr2,n,x,"the n-th prime number contained in x")
  CASE(Eq(n,ZERO),ZERO)
  EXISTS(y,Add(Pr2(Sub(n,ONE),x),ONE),x,And(Prim(y),divisible(x,y)),y);
  RESULT(ZERO)
FIN
```

The only functions, which need a memory (stack)
The other functions are mere enumerations.

```
//--------------------------------------------
//  4. Function
//--------------------------------------------

NATFUN1(Fac,n,"faculty of n")
 CASE(Eq(n,ZERO),ZERO)
 CASE(Eq(n,ONE),ONE)
 RESULT(Mul(n,Fac(Sub(n,ONE))))
FIN
```

...and now

thanks for listening!