Proof Translation Study
March 2000

Thank you for taking part in the latest Nuprl Proof Translation Study. This study is part of a research project to build a system that translates formal, computer-generated proofs into natural language proofs. Specifically, we are transforming proofs generated by the Nuprl system into English. As part of this project, it is important to look at how people perform this translation.

As a participant, you will be asked to read a Nuprl proof and then write a corresponding English version of the proof. You wil also be asked to answer a small number of biographical questions. You are not required to answer every question, but the more information you can supply, the better. There are 6 proofs total to be translated; you should translate them all if possible, though a subset of the proofs can still be used.

For participants who are unfamiliar with the Nuprl system and how to read its proofs, I suggest that you read the **Introduction to Nuprl** and **How To Read Nuprl Proofs** sections of this document. There is also a **Definitions of Nuprl Tactics** sheet provided which defines the tactics used in the following proofs. Feel free to consult any of this material while performing the translation tasks.

Introduction to Nuprl

The Nuprl system is a proof assistant which helps users create formal proofs of theorems. Within the Nuprl language, one defines mathematical objects and then states theorems to prove about these objects. The language which Nuprl uses is typed; that is, every mathematical object is explicitly required to have a type.

In order to prove a theorem which has been entered into Nuprl, the user specifies a sequence of tactics to apply. A tactic is a strategy for proving the current goal given the current hypotheses. The user specifies a tactic to apply and the tactic runs on the current goal and hypotheses to create a set of new subgoals. The subgoals created are labeled as "main", "aux" for auxiliary or "wf" for well-formedness (this is a subset of the "aux" subgoals). Intuitively, auxiliary subgoals prove supplementary facts which are needed for the main branch of the proof to hold and well-formedness subgoals prove that the statements made respect the type information given for the theorem (this is necessary because Nuprl uses a typed language). The user then specifies tactics to use to prove the new subgoals. This process is repeated until every subgoal is proved, in which case the entire theorem is proved.

Nuprl also has objects called conversions which can be used to rewrite clauses or parts of clauses. Conversions are rewrite rules that take a term and rewrite it to an equivalent term. These may be used to manipulate hypotheses to facilitate matching. Once a conversion has been defined, tactics are used to run these conversions and perform the actual rewriting.

The user often knows a sequence of tactics that they wish to apply, such as knowing that once they apply tactic T they want to apply tactic "Auto" to the auxiliary subgoals in order to prove them automatically as they will be simple enough to prove in this manner (this occurs often). Nuprl allows the user to combine the application of tactics, for example by using the "THEN" tactic which takes two tactics as arguments, one to apply first to the current goal and the second to apply to all of the generated subgoals. The only subgoals which are shown are those that remain to be proved after all of the listed tactics are run one the specified subgoals.

Nuprl allows the user to make reference to previously proved theorems, or, as Nuprl refers to them, lemmas. A lemma is invoked by using the name under which the theorem was proved as an argument to a tactic that acts on lemmas. the formal statement of the lemma does not need to be given again. Similarly, mathematical objects such as functions do not need to be defined within a proof but rather are defined outside of proofs and given names. In order to use the definition of such an object, one uses the name under which it was defined as an argument to a relevant tactic.

How To Read Nuprl Proofs

There are multiple ways to present Nuprl proofs. You are being asked to read proofs that are presented in a fairly linear, symbolic manner, though the tree structure of the proof should still be evident.

Each step of a Nuprl proof consists of four pieces of information: the hypotheses, the conclusion, the applied tactic and the generated subgoals. The proof begins with the goal statement to be proved, and at each step in the proof a tactic is applied that either discharges the proof obligation or produces one or more new subgoals to prove.

In general, a proof tree is made up of proof nodes with hypotheses, a conclusion, and an applied tactic. The top node of a proof tree shows the theorem to be proved as the conclusion and there are no hypotheses. A leaf of the proof tree is reached when no subgoals are generated after a tactic is applied. The hypothses are the set of valid premises to use for proving the current goal (or conclusion), which is given after the ⊢symbol. Hypotheses can be referred to by number; the conclusion can be referred to by the number 0, as if it were a hypothesis. The applied tactic is the tactic which the user has chosen to apply to the current goal and hypothesis in order to make progress in the proof. The generated subgoals are the new subgoals which need to be proved after the tactic is applied to the current goal.

A simple proof can be seen here:

```
*T add_mono_wrt_eq

⊢ ∀a,b,n:ℤ.  a = b ⟺ a + n  = b + n
|
BY GenUnivCD THENA Auto
|\
| 1. a: ℤ
| 2. b: ℤ
| 3. n: ℤ
| 4. a = b
| ⊢ a + n = b + n
| |
| BY Auto
| |
| DONE
 \
  1. a: ℤ
  2. b: ℤ
  3. n: ℤ
  4. a + n = b + n
  ⊢ a = b
  |
  BY Using [‘n’,⌜n⌝] (BackThruLemma ‘add_cancel_in_eq‘) THEN Auto
  |
  DONE
```

This proof has three nodes (or steps) in it. The first step takes the initial goal, with no hypotheses, and creates two new subgoals, each with four, numbered hypotheses. The tactic applied to create the two new subgoals is “GenUnivCD THENA Auto”. You can follow the path from the current node to any child nodes by following the vertical lines draw into the proof. When multiple subgoals are created, all of new nodes are indented from the previous goal. For longer proofs, a subgoal and its children may be presented separately from the rest of the tree, with a label given to match the sub-proof with its intended location in the proof tree.

Definitions of Nuprl Tactics

The following is not a complete list of Nuprl tactics; only those tactics which are used in the proofs in this study are listed here. Tactics are listed alphabetically. If you are do not know what a tactic or a conversion is, you should consult the Introduction to Nuprl page before proceeding. Feel free to refer to this page while completing this study. Recall that hypotheses can be referred to by number, with hypothesis 0 being the conclusion.

... an abbreviation for "THEN Auto"; see the entries for **THEN** and **Auto** for further explanation

...a an abbreviation for "THENA Auto"; see the entries for **THENA** and **Auto** for further explanation

AbReduce used in the form "AbReduce *i*", simplifies hypothesis *i* using a defined set of equivalences

ArithSimp used in the form "ArithSimp *i*", creates a main subgoal where hypothesis i is rewritten to be in arithmetical canonical form and an auxiliary subgoal to show that the replacement statement is equivalent to the original statement of hypothesis *i*

Assert used in the form "Assert *t*", generates two subgoals, one which adds *t* as a new hypothesis and the other with *t* as the goal to prove from the current hypotheses

Auto or Auto' repeatedly applies a set of elementary automatic proving techniques (including Arith, GenExRepD, and others) until no more progress is made

BLemma an abbreviation for "BackThruLemma", used in the form "BLemma *n*" where *n* is the name of a lemma whose statement is a universally quantified inference, matches the current current goal to the consequent of the inference and takes the antecedents of the inference as subgoals

CompNatInd used in the form "CompNatInd *i*" where *i* is the number of a hypothesis, does complete natural number induction on hypothesis *i*, generating upcase and basecase subgoals

D used in the form "D *i*", decomposes the outermost connective of hypothesis *i*, performing as many Unfold operations as necessary on externally defined mathematical objects until a connective that it can decompose is outermost; see **Unfold** for further information

Decide used in the form "Decide (*p*)", performs a case split on whether the proposition *p* is true or false; two subgoals are created, one with *p* added as a new hypothesis and the other with "not *p*" added as a new hypothesis

DTerm used in the form "DTerm *t i*" where *t* is a term and *i* is the number of a hypothesis whose outermost connective is an existential quantifier, decomposes the outermost existential of the hypothesis using *t* as the witness

ExistHD used in the form "ExistHD *i*" where *i* is the number of a hypothesis that is an existential formula, replaces hypothesis *i* with hypotheses for the witness(es) of the existential quanitifier and the asserted proposition in terms of these witnesses

FLemma an abbreviation for "FwdThruLemma"; see the entry for **FwdThruLemma** for more information

FwdThruLemma used in the form "FwdThruLemma n" where n is the name of a lemma whose statement is a universally quantified inference, matches the antecedents to current hypotheses and adds the consequent of the inference as a new hypothesis

HypSubst used in the form "HypSubst *i c*" where *i* is the number of a hypothesis of the form "$t_1 = t_2$" and *c* is the number of a hypothesis or the goal, a new subgoal is created where all occurrences of t_1 in clause number *c* are replaced with occurrences of t_2, a subgoal with "$t_1 = t_2$" as the goal and a wf subgoal

InstConcl used in the form "InstConcl [t_1,...,t_n]", instantiates the existential quantifiers in the conclusion with the terms t_1 through t_n

InstHyp used in the form "InstHyp [t_1,...,t_n] *i*", instantiates a universal formula in hypothesis *i* with the terms t_1 through t_n

InstLemma used in the form "InstLemma n $[t_1,..,t_n]$" where n is the name of a lemma, instantiates the lemma n with the terms t_1 through t_n

LemmaC used in the form "LemmaC n" where n is the name of a lemma whose statement is a universally quantified inference with the consequent "$a\ r\ b$" where r is an equivalence relation, creates a conversion to replace instances of a with instances of b

NatInd used in the form "NatInd i" where i is the number of a hypothesis, does natural number induction on hypothesis i, generating upcase and basecase subgoals

NSubsetInd used in the form "NSubstInd i" where i is the number of a hypothesis that indicates a subrange of the natural numbers; does induction on that subrange, generating upcase and basecase subgoals

NthC used in the form "NthC i c", specifies that the conversion c should be applied in the ith place where it is applicable

OnMCls an abbreviation for "OnMClauses"; used in the form "OnMCls $[i_1,...,i_n]$ T" where $i_1,...,i_n$ are hypothesis numbers and T is a tactic, runs T on each hypothesis in the list, in the order listed, considering only main subgoals

OnVar used in the form "OnVar v T" where v is a variable existing in the proof and T is a tactic, runs T using the hypothesis declaring v as its argument

RecCaseSplitNth used in the form "RecCaseSplitNth i n" where i is a hypothesis number and n is the name of an externally defined mathematical object, replace n with its definition in the ith position where it is applicable and repeat recursively, decompositing "if-then-else" statements as necessary

RecUnfold used in the form "RecUnfold n i" where n is the name of an externally defined mathematical object and i is a hypothesis number, recursively replaces n with its definition in the hypothesis i

Rewrite used in the form "Rewrite c i" where c is a conversion and i is the number of a hypothesis or the goal, applies the conversion to the clause i

RWN an abbreviation for "Rewrite (NthC n c)"; see the entries for **Rewrite** and **NthC** for further information

SeqOnM used in the form "SeqOnM $[T_1,...T_n]$" where $T_1,...,T_n$ are tactics, runs T_1 through T_n on successive main suboals

SIAuto runs the same as Auto but with **SupInf** added to the set of automatic proving techniques used

SupInf used in the form "SupInf", solves integer inequalities

THEN used in the form "T_1 THEN T_2" where T_1 and T_2 are tactics, runs T_1 and then runs T_2 on all of the generated subgoals

THENA used in the form "T_1 THENA T_2" where T_1 and T_2 are tactics, runs T_1 and then runs T_2 on the auxiliary subgoals generated

THENM used in the form "T_1 THENM T_2" where T_1 and T_2 are tactics, runs T_1 and then runs T_2 on the main subgoal generated

TryOnAllClauses used in the form "TryOnAllClauses T" where T is a tactic, run T on all hypotheses and the conclusion, doing nothing if it fails

Unfold used in the form "Unfold n i" where n is the name of an externally defined mathematical object, replaces n with its definition anywhere that it appears in hypothesis i

UnivCD decomposes universal quantifiers and implications in the conclusion until the outermost connective is not one of these two connectives; see **D** for further information on decomposition

Using used in the form "Using $[s_1,...,s_n]$ T" where $[s_1,...,s_n]$ is a list of substitutions, runs tactic T using the list of substitutions to help it instantiate universally quantified variables as called for

Proof Translation Study - Instructions

Fill out the following information.

Name ____________________

E-mail Address ____________________

Cornell Status (e.g. undergrad, grad student, researcher, etc.) ____________________

What experience to do you have in writing math proofs? ____________________

How proficient would you say you are in reading formal logical? ____________________

How proficient would you say you are in reading Nuprl proofs? ____________________

How proficient would you say you are in using the Nuprl system? ____________________

For each of the following six proofs, please follow the following procedure:

1. Read the proof carefully.
2. Describe in a single sentence what the proof is trying to show.
3. Describe in 1-2 sentences what approach the proof uses; i.e., give a high-level description of the proof approach such that an experienced mathematician could produce a similar proof.
4. Write out an English language version of the proof given. Think about how you would want the proof to look if you were handing it out to a student who is just learning this material and wants to see a "perfect" example of this proof.

You may use symbols and formulas where it seems natural.

Please make sure that all of your responses are legible as we are interested in exactly what word and symbol choices you make.

When you are finished, please turn in this sheet along with your work on each of the proofs. You are encouraged but not required to complete the entire study; if you choose to stop before working on all of the proofs, please turn in as much of the study as you have completed.

(blank page)

Translation Study: Proof One

Proof Concepts

```
*T int_entire              ⊢ ∀a,b:ℤ.  a * b = 0 ⇒ a = 0 ∨ b = 0
```

Proof

```
*T int_entire_a

⊢ ∀a,b:ℤ.  a ≠ 0 ⇒ b ≠ 0  ⇒ a * b ≠ 0
|
BY UnivCD THENA Auto
|
1. a: ℤ
2. b: ℤ
3. a ≠ 0
4. b ≠ 0
⊢ a * b ≠ 0
|
BY Decide ⌜a * b = 0⌝ THENA Auto
|\
| 1. a: ℤ
| 2. b: ℤ
| 3. a ≠ 0
| 4. b ≠ 0
| 5. a * b = 0
| ⊢ a * b ≠ 0
| |
| BY FwdThruLemma ‘int_entire‘ [5] THEN Auto
| |
| 1. a: ℤ
| 2. b: ℤ
| 3. a ≠ 0
| 4. b ≠ 0
| 5. a * b = 0
| 6. a = 0 ∨ b = 0
| ⊢ a * b ≠ 0
| |
| BY D 6 THEN Auto
| |
| DONE
 \
  1. a: ℤ
  2. b: ℤ
  3. a ≠ 0
  4. b ≠ 0
  5. ¬(a * b = 0)
  ⊢ a * b ≠ 0
  |
  BY Auto
  |
  DONE
```

(blank page)

Translation Study: Proof Two

Proof Concepts

ℕb   ℕb = { x ∈ ℕ | x < b }

Proof

```
*T quot_rem_exists_n

⊢ ∀a:ℕ. ∀b:ℕ+.  ∃q:ℕ. ∃r:ℕb. a = q * b + r
|
BY (UnivCD ...a)
|
1. a: ℕ
2. b: ℕ+
⊢ ∃q:ℕ. ∃r:ℕb. a = q * b + r
|
BY % Generalize: introduce induction var and invariant %
|  Assert ⌈∀c:ℕ. (∃q:ℕ. a = q * b + c) ⇒ (∃q:ℕ. ∃r:ℕb. a = q * b + r)⌉
|  THENA (D 0 ...a)
|\
| 1. a: ℕ
| 2. b: ℕ+
| 3. c: ℕ
| ⊢ (∃q:ℕ. a = q * b + c) ⇒ (∃q:ℕ. ∃r:ℕb. a = q * b + r)
| |
| BY (OnVar ‘c’ CompNatInd ...a) THENM (D 0 ...a)
| |
| 1. a: ℕ
| 2. b: ℕ+
| 3. c: ℕ
| 4. ∀c1:ℕ. c1 < c ⇒ (∃q:ℕ. a = q * b + c1) ⇒ (∃q:ℕ. ∃r:ℕb. a = q * b + r)
| 5. ∃q:ℕ. a = q * b + c
| ⊢ ∃q:ℕ. ∃r:ℕb. a = q * b + r
| |
| BY % Case split on whether c is good enough yet %
| |  (Decide ⌈c < b⌉ ...a)
| |\
| | 1. a: ℕ
| | 2. b: ℕ+
| | 3. c: ℕ
| | 4. ∀c1:ℕ. c1 < c ⇒ (∃q:ℕ. a = q * b + c1) ⇒ (∃q:ℕ. ∃r:ℕb. a = q * b + r)
| | 5. ∃q:ℕ. a = q * b + c
| | 6. c < b
| | ⊢ ∃q:ℕ. ∃r:ℕb. a = q * b + r
| | |
| | BY % Invariant and condition true implies original goal%
| | |  (D 5 THEN InstConcl [⌈q⌉;⌈c⌉] ...)
| | |
| | DONE
|  \
|   1. a: ℕ
|   2. b: ℕ+
|   3. c: ℕ
|   4. ∀c1:ℕ. c1 < c ⇒ (∃q:ℕ. a = q * b + c1) ⇒ (∃q:ℕ. ∃r:ℕb. a = q * b + r)
|   5. ∃q:ℕ. a = q * b + c
|   6. ¬(c < b)
|   ⊢ ∃q:ℕ. ∃r:ℕb. a = q * b + r
|   |
|   BY % Inductive Case%
|   |  InstHyp [⌈c - b⌉] 4 THEN SIAuto
|   |
|   1. a: ℕ
|   2. b: ℕ+
|   3. c: ℕ
|   4. ∀c1:ℕ. c1 < c ⇒ (∃q:ℕ. a = q * b + c1) ⇒ (∃q:ℕ. ∃r:ℕb. a = q * b + r)
|   5. ∃q:ℕ. a = q * b + c
|   6. ¬(c < b)
|   ⊢ ∃q:ℕ. a = q * b + (c - b)
|   |
|   BY % Check invariant maintained %
|   |  D 5 THEN (InstConcl [⌈q + 1⌉] ...)
|   |
|   DONE
 \
  1. a: ℕ
  2. b: ℕ+
  3. ∀c:ℕ. (∃q:ℕ. a = q * b + c) ⇒ (∃q:ℕ. ∃r:ℕb. a = q * b + r)
  ⊢ ∃q:ℕ. ∃r:ℕb. a = q * b + r
  |
  BY (InstHyp [⌈a⌉] 3 ...)
  |
  1. a: ℕ
  2. b: ℕ+
  3. ∀c:ℕ. (∃q:ℕ. a = q * b + c) ⇒ (∃q:ℕ. ∃r:ℕb. a = q * b + r)
  ⊢ ∃q:ℕ. a = q * b + a
  |
  BY % Check invariant holds initially %
  |  (InstConcl [⌈0⌉] ...)
  |
  DONE
```

Translation Study: Proof Three

Proof Concepts

```
CoPrime                    CoPrime(a,b) ==  GCD(a;b;1)
fib                        fib(n) == if (n = 0) ∨ (n = 1) then 1 else fib(n - 1) + fib(n - 2) fi
*T gcd_p_one             ⊢ ∀a:ℤ. GCD(a;1;1)
*T gcd_p_sym             ⊢ ∀a,b,y:ℤ.  GCD(a;b;y) ⇒ GCD(b;a;y)
*T gcd_p_shift           ⊢ ∀a,b,y,k:ℤ.  GCD(a;b;y) ⇒ GCD(a;b + k * a;y)
```

Proof

```
*T fib_coprime

⊢ ∀n:ℕ. CoPrime(fib(n),fib(n + 1))
|
BY (D 0 THENM OnVar ‘n’ NatInd  ...a)
|\
| ⊢ CoPrime(fib(0),fib(0 + 1))
| |
| BY RecUnfold ‘fib‘ 0 THEN AbReduce 0
| |
| ⊢ CoPrime(1,1)
| |
| BY Unfold ‘coprime‘ 0 THEN (BLemma ‘gcd_p_one‘ ...)
| |
| DONE
 \
  1. n: ℤ
  2. 0 < n
  3. CoPrime(fib(n - 1),fib((n - 1) + 1))
  ⊢ CoPrime(fib(n),fib(n + 1))
  |
  BY (OnMCls [0;3] ArithSimp  ...a)
  |
  1. n: ℤ
  2. 0 < n
  3. CoPrime(fib(-1 + n),fib(n))
  ⊢ CoPrime(fib(n),fib(1 + n))
  |
  BY (RecCaseSplitNth 2 ‘fib‘ ...a)
  |\
  | 1. n: ℤ
  | 2. 0 < n
  | 3. CoPrime(fib(-1 + n),fib(n))
  | 4. 1 + n = 0 ∨ 1 + n = 1
  | ⊢ CoPrime(fib(n),1)
  | |
  | BY SIAuto
  | |
  | DONE
   \
    1. n: ℤ
    2. 0 < n
    3. CoPrime(fib(-1 + n),fib(n))
    4. ¬(1 + n = 0) ∧ ¬(1 + n = 1)
    ⊢ CoPrime(fib(n),fib((1 + n) - 1) + fib((1 + n) - 2))
```

```
|
BY (ArithSimp 0 ...a)
|
1. n: ℤ
2. 0 < n
3. CoPrime(fib(-1 + n),fib(n))
4. ¬(1 + n = 0) ∧ ¬(1 + n = 1)
⊢ CoPrime(fib(n),fib(-1 + n) + fib(n))
|
BY TryOnAllClauses (Unfold ‘coprime‘)
|  THEN SeqOnM
|  [FLemma ‘gcd_p_sym‘ [3]
|  ;Using [‘k’,⌈1⌉] (FLemma ‘gcd_p_shift‘ [-1])
|  ;ArithSimp (-1)]
|  THEN SIAuto
|
DONE
```

Translation Study: Proof Four

Proof Concepts

```
GCD                       GCD(a;b;y) ==  y | a ∧ y | b ∧ (∀z:ℤ. z | a ∧ z | b ⇒ z | y)
*T gcd_p_zero            ⊢ ∀a:ℤ. GCD(a;0;a)
*T quot_rem_exists       ⊢ ∀a:ℤ. ∀b:ℕ+.  ∃q: ℤ. ∃r:ℕb. a = q * b + r
*T gcd_p_sym             ⊢ ∀a,b,y:ℤ.  GCD(a;b;y) ⇒ GCD(b;a;y)
*T mul_com               ⊢ ∀a,b:ℤ.  a * b = b * a
*T gcd_p_shift           ⊢ ∀a,b,y,k:ℤ.  GCD(a;b;y) ⇒ GCD(a;b + k * a;y)
```

Proof

```
*T bezout_ident_n

⊢ ∀b:ℕ. ∀a:ℤ.  ∃u,v:ℤ. GCD(a;b;u * a + v * b)
|
BY (D 0 THENM OnVar ‘b’ CompNatInd THENM D 0 ...a)
|
1. b: ℕ
2. ∀b1:ℕ. b1 < b ⇒ (∀a:ℤ. ∃u,v:ℤ. GCD(a;b1;u * a + v * b1))
3. a: ℤ
⊢ ∃u,v:ℤ. GCD(a;b;u * a + v * b)
|
BY (Decide ⌈b = 0⌉ ...a)
|\
| 1. b: ℕ
| 2. ∀b1:ℕ. b1 < b ⇒ (∀a:ℤ. ∃u,v:ℤ. GCD(a;b1;u * a + v * b1))
| 3. a: ℤ
| 4. b = 0
| ⊢ ∃u,v:ℤ. GCD(a;b;u * a + v * b)
| |
| BY (InstConcl [⌈1⌉;⌈0⌉] THENM HypSubst 4 0 THENM ArithSimp 0 ...a)
| |
| 1. b: ℕ
| 2. ∀b1:ℕ. b1 < b ⇒ (∀a:ℤ. ∃u,v:ℤ. GCD(a;b1;u * a + v * b1))
| 3. a: ℤ
| 4. b = 0
| ⊢ GCD(a;0;a)
| |
| BY (BLemma ‘gcd_p_zero‘ ...)
| |
| DONE
 \
  1. b: ℕ
  2. ∀b1:ℕ. b1 < b ⇒ (∀a:ℤ. ∃u,v:ℤ. GCD(a;b1;u * a + v * b1))
  3. a: ℤ
  4. ¬(b = 0)
  ⊢ ∃u,v:ℤ. GCD(a;b;u * a + v * b)
  |
  BY (InstLemma ‘quot_rem_exists‘ [⌈a⌉;⌈b⌉] ...a) THEN ExistHD (-1)
  |
  1. b: ℕ
  2. ∀b1:ℕ. b1 < b ⇒ (∀a:ℤ. ∃u,v:ℤ. GCD(a;b1;u * a + v * b1))
  3. a: ℤ
  4. ¬(b = 0)
  5. q: ℤ
```

```
6. r: ℕb
7. a = q * b + r
⊢ ∃u,v:ℤ. GCD(a;b;u * a + v * b)
|
BY (InstHyp [⌈r⌉;⌈b⌉] 2 ...a) THEN ExistHD (-1)
|
1. b: ℕ
2. ∀b1:ℕ. b1 < b ⇒ (∀a:ℤ. ∃u,v:ℤ. GCD(a;b1;u * a + v * b1))
3. a: ℤ
4. ¬(b = 0)
5. q: ℤ
6. r: ℕb
7. a = q * b + r
8. u: ℤ
9. v: ℤ
10. GCD(b;r;u * b + v * r)
⊢ ∃u,v:ℤ. GCD(a;b;u * a + v * b)
|
BY (InstConcl [⌈v⌉;⌈u - q * v⌉] THENM HypSubst 7 0 THENM OnMCls [0;10] ArithSimp ...a)
|
1. b: ℕ
2. ∀b1:ℕ. b1 < b ⇒ (∀a:ℤ. ∃u,v:ℤ. GCD(a;b1;u * a + v * b1))
3. a: ℤ
4. ¬(b = 0)
5. q: ℤ
6. r: ℕb
7. a = q * b + r
8. u: ℤ
9. v: ℤ
10. GCD(b;r;b * u + r * v)
⊢ GCD(r + b * q;b;b * u + r * v)
|
BY (BLemma ‘gcd_p_sym‘ ...a)
|
1. b: ℕ
2. ∀b1:ℕ. b1 < b ⇒ (∀a:ℤ. ∃u,v:ℤ. GCD(a;b1;u * a + v * b1))
3. a: ℤ
4. ¬(b = 0)
5. q: ℤ
6. r: ℕb
7. a = q * b + r
8. u: ℤ
9. v: ℤ
10. GCD(b;r;b * u + r * v)
⊢ GCD(b;r + b * q;b * u + r * v)
|
BY (RWN 1 (LemmaC ‘mul_com‘) 0 THENM BLemma ‘gcd_p_shift‘ ...)
|
DONE
```

Translation Study: Proof Five

Proof

```
*T stamps

⊢ ∀i:{8...}. ∃m,n:ℕ. 3 * m + 5 * n = i
|
BY D 0 THENA Auto
|
1. i:{8...}
⊢ ∃m,n:ℕ. 3 * m + 5 * n = i
|
BY NSubsetInd 1 THEN Auto
|\
| 1. i:ℤ
| 2. 0 < i
| 3. 8 = i
| ⊢ ∃m,n:ℕ. 3 * m + 5 * n = i
| |
| BY DTerm ⌈1⌉ 0 THENM DTerm ⌈1⌉ 0 THEN Auto
| |
| DONE
 \
  1. i:ℤ
  2. 8 < i
  3. ∃m,n:ℕ. 3 * m + 5 * n = i - 1
  ⊢ ∃m,n:ℕ. 3 * m + 5 * n = i
  |
  BY D 3 THEN D 4
  |
  1. i:ℤ
  2. 8 < i
  3. m:ℕ
  4. n:ℕ
  5. 3 * m + 5 * n = i - 1
  ⊢ ∃m,n:ℕ. 3 * m + 5 * n = i
  |
  BY Decide ⌈n > 0⌉ THENA Auto
  |\
  | 1. i:ℤ
  | 2. 8 < i
  | 3. m:ℕ
  | 4. n:ℕ
  | 5. 3 * m + 5 * n = i - 1
  | 6. n > 0
  | ⊢ ∃m,n:ℕ. 3 * m + 5 * n = i
  | |
  | BY DTerm ⌈m + 2⌉ 0 THENM DTerm ⌈n - 1⌉ 0 THEN Auto
   \
    1. i:ℤ
    2. 8 < i
    3. m:ℕ
    4. n:ℕ
    5. 3 * m + 5 * n = i - 1
    6. ¬(n > 0)
    ⊢ ∃m,n:ℕ. 3 * m + 5 * n = i
    |
BY DTerm ⌈m - 3⌉ 0 THENM DTerm ⌈n + 2⌉ 0 THEN Auto
|
1. i:ℤ
2. 8 < i
3. m:ℕ
4. n:ℕ
5. 3 * m + 5 * n = i - 1
6. ¬(n > 0)
⊢ 0 ≤ m - 3
|
BY SupInf THEN Auto
|
DONE
```