ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ
ΚΡΗΤΗΣ

Σχολή Τεχνολογικών Εφαρμογών

Τμήμα Ηλεκτρολογίας

Πτυχιακή Εργασία

Ευκλείδης: Ένα Διαλογικό Σύστημα Απόδειξης Θεωρημάτων με τη Μέθοδο της Επαγωγής

Σγουράκη Μαριάννα

Ηράκλειο, Οκτώβριος 2006

Εργασία που υποβλήθηκε από την Σγουράκη Μαριάννα
ως μερική εκπλήρωση των απαιτήσεων για την απόκτηση
πτυχίου στο τμήμα Ηλεκτρολογίας

Συγγραφέας:_____
Σγουράκη Μαριάννα
Τμήμα Ηλεκτρολογίας
ΤΕΙ Κρήτης

Εισηγητής:_____
Δρ. Μαρακάκης Εμμανουήλ
Αναπλ. Καθηγητής
ΤΕΙ Κρήτης

Ηράκλειο, Οκτώβριος 2006

Περιεχόμενα

Κατάλογος Αλγορίθμων

Κατάλογος Εικόνων

Κατάλογος Προγραμμάτων

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

1 Εισαγωγή

Ο στόχος αυτής της πτυχιακής εργασίας είναι η κατασκευή ενός συστήματος, το οποίο θα αποδεικνύει μαθηματικά θεωρήματα, χρησιμοποιώντας τη μέθοδο της μαθηματικής επαγωγής. Εφαρμόζουμε στο θεώρημα, που θέλουμε να αποδείξουμε, αξιώματα, λήμματα, καθώς και νόμους της λογικής πρώτης τάξης επιδιώκοντας το μετασχηματισμό του σε άλλη μορφή. Οι μετασχηματισμοί αυτοί γίνονται με μοναδικό σκοπό την απόδειξη αυτού του θεωρήματος. Το σύστημα που κατασκευάσαμε ονομάστηκε **Ευκλείδης,** προς τιμήν του αρχαίου Έλληνα μαθηματικού, ο οποίος ήταν ο άνθρωπος, που εφάρμοσε πρώτος την απόδειξη θεωρημάτων δια της επαγωγικής μεθόδου.

Η εφαρμογή αυτού του συστήματος είναι σημαντική κυρίως στον εκπαιδευτικό τομέα. Μπορεί να χρησιμοποιηθεί σαν εκπαιδευτικό εργαλείο για την διδασκαλία της μαθηματικής επαγωγής και παράλληλα ως βοηθητικό εργαλείο για την εξάσκηση μαθητευομένων σ' αυτήν. Το σύστημα απόδειξης θεωρημάτων **Ευκλείδης**, μπορεί επίσης να χρησιμοποιηθεί ως βοηθητικό εργαλείο στήριξης, για την επίλυση σημαντικών προβλημάτων με τη χρήση μαθηματικής επαγωγής.

Το Σύστημα **Ευκλείδης** υλοποιήθηκε σε γλώσσα προγραμματισμού Prolog. Η διεπικοινωνία (Interface) του συστήματος υλοποιήθηκε στη γλώσσα προγραμματισμού Visual Basic. Ο χρήστης εισάγει τα δεδομένα μέσω παραθυρικού και γραφικού περιβάλλοντος. Το σύστημα δίνει τα αποτελέσματα είτε μέσω του ίδιου παραθύρου ή σε μορφή αρχείου κειμένου. Η διεπικοινωνία έχει υλοποιηθεί σε Visual Basic για να επιτευχθούν τα σχεδιαστικά κριτήρια που αναφέρθηκαν παραπάνω.

Τα υπόλοιπα κεφάλαια αυτής της πτυχιακής έχουν ως εξής: Στο δεύτερο κεφάλαιο αναφέρονται βασικές εισαγωγικές έννοιες για το Λογικό Προγραμματισμό, την γλώσσα Prolog, την αναπαράσταση σε μη-βασική μορφή, την μαθηματική επαγωγή, καθώς και για τη μηχανική υποστήριξη απόδειξης θεωρημάτων. Στο τρίτο κεφάλαιο περιγράφεται η σύνταξη των θεωρημάτων, αξιωμάτων, λημμάτων, και νόμων του προτασιακού και κατηγορηματικού λογισμού, τα οποία γίνονται αποδεκτά από το σύστημα το οποίο κατασκευάστηκε. Επιπλέον σ' αυτό το κεφάλαιο αναλύονται οι αναπαραστάσεις των θεωρημάτων, αξιωμάτων και των άλλων στοιχείων που χρησιμοποιούνται από το σύστημα. Επίσης αναλύεται η αναπαράσταση της ισότητας, των μετασχηματισμών, αλλά και της απόδειξης. Στο τέταρτο κεφάλαιο παρουσιάζεται η αρχιτεκτονική του συστήματός μας. Περιγράφονται τα τμήματα (modules) που το απαρτίζουν και πως αλληλεπιδρούν μεταξύ τους. Επίσης σ' αυτό το κεφάλαιο περιγράφεται ο σχεδιασμός της διεπικοινωνίας του συστήματος. Τέλος, δίνεται ένα πλήρες σενάριο χρήσης του συστήματος. Στο πέμπτο κεφαλαίο αναφέρονται τα συμπεράσματα, και οι τυχόν οι δυνατές επεκτάσεις. Τέλος, υπάρχουν δύο παραρτήματα. Στο πρώτο παράρτημα παρουσιάζονται παραδείγματα αποδείξεως θεωρημάτων δια της μαθηματικής επαγωγής σε διεπικοινωνία, η οποία έχει υλοποιηθεί σε Prolog αλλά και σε παραθυρικό περιβάλλον. Στο δεύτερο παράρτημα παρουσιάζονται επιπλέον παραδείγματα μαθηματικής επαγωγής.

2 Βασικές Γνώσεις Υπόβαθρου

2.1 Εισαγωγή στην Prolog

Η Prolog είναι μια γλώσσα προγραμματισμού για εφαρμογές Τεχνητής Νοημοσύνης. Η λέξη Prolog σημαίνει PROgramming in LOGic. Η Prolog, μπορεί επίσης να θεωρηθεί σαν ένα τυπικό σύστημα αποδείξεων. Η Prolog έχει τις ρίζες της στη μαθηματική λογική και συγκεκριμένα στην λογική πρώτης τάξεως. Βασίζεται σε ένα σύνολο μηχανισμών οι οποίοι την κάνουν αρκετά δυνατή και ευέλικτη γλώσσα προγραμματισμού. Οι μηχανισμοί στους οποίους βασίζεται είναι οι εξής:

1. Η ταυτοποίηση (unification).
2. Οι δενδροειδείς δομές δεδομένων.
3. Η οπισθοδρόμηση (backtracking).

Η Prolog είναι κατάλληλη για προβλήματα Τεχνητής Νοημοσύνης και ιδιαίτερα για εκείνα που έχουν να κάνουν με δομημένα αντικείμενα (objects) και τις μεταξύ τους σχέσεις. Στις συνήθεις γλώσσες προγραμματισμού (Pascal, C, Fortran, κτλ.) ένα πρόβλημα περιγράφεται διαδικαστικά (procedural). Δηλαδή, ο προγραμματιστής πρέπει να δώσει όλα τα βήματα που θα ακολουθήσει ο υπολογιστής για να λύσει το πρόβλημα. Ενώ η Prolog εισάγει την έννοια του δηλωτικού (declarative) τρόπου περιγραφής προβλημάτων. Σ' αυτήν την περίπτωση ο προγραμματιστής περιγράφει το πρόβλημα υπό μορφή γεγονότων και κανόνων, η Prolog χρησιμοποιώντας συμπερασματική συλλογιστική (deductive reasoning) βρίσκει όλες τις πιθανές λύσεις του προβλήματος. Αυτός ο τρόπος περιγραφής του προβλήματος και εύρεσης των υπαρχόντων λύσεων είναι που κάνει διαφορετικό τον προγραμματισμό σε Prolog.

Σ' ένα Prolog πρόγραμμα διακρίνουμε δύο επίπεδα εννοιών, την δηλωτική έννοια και την διαδικαστική.

1. Η ***δηλωτική έννοια*** ενδιαφέρεται μόνο για τις σχέσεις οι οποίες ορίζονται σ' ένα λογικό πρόγραμμα. Δηλαδή, προσδιορίζει τι κάνει το πρόγραμμα αλλά όχι πως το κάνει.
2. Η ***διαδικαστική έννοια*** ενδιαφέρεται για τον υπολογισμό των σχέσεων, καθορίζει πως υπολογίζονται οι σχέσεις των αντικειμένων. Δηλαδή προσδιορίζει πως θα υπολογιστεί η έξοδος του προγράμματος.

Η Prolog είναι μια γλώσσα με πολλές δυνατότητες γι' αυτό και συνεχώς γίνεται πιο δημοφιλής. Χρησιμοποιείται κυρίως για υλοποίηση εφαρμογών Τεχνητής Νοημοσύνης αλλά όχι μόνο, έχει πιο ευρεία χρήση στην ανάπτυξη λογισμικού. Τα προτερήματα της Prolog για την ανάπτυξη λογισμικού είναι τα εξής:

- Η Prolog είναι μια γλώσσα υψηλού επιπέδου βασισμένη στην λογική η οποία υποστηρίζει τυπική συλλογιστική (formal reasoning).
- Η Prolog είναι κατάλληλη για την κατασκευή γρήγορων πρωτοτύπων (rapid prototyping).
- Λόγω της απλότητας της σύνταξης της Prolog προγράμματα συντηρούνται και επαναχρησιμοποιούνται εύκολα.
- Η Prolog μπορεί να χρησιμοποιηθεί σαν γλώσσα υλοποίησης εκτελέσιμων προδιαγραφών εφόσον φυσικά οι προδιαγραφές έχουν εκφραστεί σε λογική.
- Η Prolog θεωρείται κατάλληλη για κατασκευή μετα-προγραμμάτων.

2.1.1 Προτάσεις Γεγονότα

Η πιο απλή πρόταση σε Prolog είναι το γεγονός. Τα γεγονότα είναι ένας τρόπος για να εκφραστούν οι σχέσεις που ισχύουν μεταξύ αντικειμένων. Ας πάρουμε για παράδειγμα τις σχέσεις που υπάρχουν σε μια οικογένεια. Οι σχέσεις οικογένειας είναι ένα κλασικό παράδειγμα για εισαγωγή στις βασικές έννοιες της Prolog.

Σε αυτό το παράδειγμα τα αντικείμενα είναι οι άνθρωποι (yannis, maria, soula, kostas κλπ.) και οι μεταξύ τους σχέσεις είναι (sizigos, pateras, mitera κλπ.).

Το γεγονός ότι ο «Γιάννης είναι σύζυγος της Μαρίας» εκφράζεται ως εξής σε Prolog: sizigos (yiannis, maria).

Πρώτα γράφεται η σχέση sizigos, μέσα σε παρένθεση γράφονται τα αντικείμενα και η τελεία δεικνύει το τέλος του γεγονότος. Η σχέση σε ένα γεγονός λέγεται *κατηγόρημα (predicate)* και τα αντικείμενα *ορίσματα (arguments)*. Στο παραπάνω γεγονός η σχέση sizigos είναι το κατηγόρημα, τα αντικείμενα yannis και maria είναι τα ορίσματα. Τα ονόματα των κατηγορημάτων (σχέσεων) και των ορισμάτων (αντικειμένων) μπορούν να γραφούν με πεζά ή κεφαλαία γράμματα του λατινικού αλφάβητου, τους αριθμούς 0–9 και το σύμβολο της υπογράμμισης (_). Ο πρώτος χαρακτήρας του ονόματος θα πρέπει να είναι πεζό γράμμα του λατινικού αλφάβητου.

Όταν ορίζουμε ένα γεγονός πρέπει να είμαστε συμβατοί με την σειρά που έχουμε ορίσει τα ορίσματα στα γεγονότα. Τα παρακάτω γεγονότα αντιπροσωπεύουν διαφορετικές σχέσεις.

```
mitera (anna, soula).

mitera (soula, anna).
```

Το πρώτο γεγονός εκφράζει την σχέση ότι «η Άννα είναι μητέρα της Σούλας», ενώ το δεύτερο εκφράζει τη σχέση ότι «η Σούλα είναι μητέρα της Άννας». Κάθε κατηγόρημα πρέπει να έχει τον ίδιο αριθμό ορισμάτων. Κατηγορήματα με ίδιο όνομα αλλά διαφορετικό αριθμό ορισμάτων θεωρούνται σαν διαφορετικά κατηγορήματα.

Τα ονόματα που χρησιμοποιούνται για ονόματα κατηγορημάτων και για τα ορίσματά τους είναι συμβολικά. Αυτό σημαίνει ότι η σχέση sizigos(yannis, maria) θα μπορούσε να είχε γραφτεί ως εξής a(b, c) όπου το a θα συμβόλιζε το κατηγόρημα sizigos, το b το όρισμα yannis και το c το όρισμα maria.

Μια συλλογή από γεγονότα ονομάζεται ***βάση δεδομένων*** ή ***βάση γνώσεων***. Το Πρόγραμμα 2.1 είναι μία βάση γνώσεων από γεγονότα.

```
sizigos(yannis, maria).
pateras(yannis, kostas).
mitera(maria, kostas).
pateras(kostas, anna).
pateras(kostas, rania).
mitera(soula, anna).
mitera(soula, rania).
sizigos(kostas, soula).
adelfi(anna, rania).
```

Πρόγραμμα 2.1: *Βάση γνώσεων σχέσεων οικογένειας.*

2.1.2 Ερωτήσεις

Στη βάση γνώσεων του παραδείγματός μας, (Πρόγραμμα 2.1) η οποία μέχρι τώρα περιέχει μόνο γεγονότα, μπορούμε να κάνουμε ερωτήσεις για την αλήθεια (ισχύ) ή μη ενός γεγονότος. Οι ερωτήσεις σχηματίζονται από κατηγορήματα με τη διαφορά ότι το ειδικό σύμβολο «?-» προηγείται στην ερώτηση. Η ερώτηση, «είναι ο Κώστας πατέρας της Άννας;» έχει την εξής μορφή σε Prolog:

```
?- pateras(kostas, anna).
```

Η Prolog ερευνά την βάση γνώσεων ώστε να βρεί κάποιο γεγονός στη βάση που να ταυτοποιείται (ταιριάζει) με το κατηγόρημα της ερώτησης. Εάν υπάρχει κατηγόρημα που να ταυτοποιείται με αυτό της ερώτησης τότε η Prolog θα δώσει σαν απάντηση **yes**.

Η απάντηση της Prolog στην ερώτηση, «είναι η Μαρία μητέρα του Κώστα;»

```
?- mitera(maria, kostas).
```

θα είναι yes. Ενώ η απάντηση στην ερώτηση, «είναι ο Κώστας παιδί της Μαρίας;» είναι **no**.

```
?- paidi(kostas, maria).
```

Η απάντηση **no** δεν σημαίνει ότι ο ισχυρισμός, «ο Κώστας είναι παιδί της Μαρίας» είναι ψευδής, άλλωστε οι σχέσεις συγγενείας που εκφράζονται με το παράδειγμα του Προγράμματος 2.1 δεικνύουν ότι αυτός ο ισχυρισμός ισχύει. Η απάντηση **no** της Prolog σημαίνει ότι αυτός ο ισχυρισμός δεν μπορεί να αποδειχθεί ότι είναι αληθής από το υπάρχον πρόγραμμα, δηλαδή από το Πρόγραμμα 2.1.

2.1.3 Προτάσεις Κανόνες

Ορισμός 2.1

Πλειάδα (***tuple*** ή ***ν-tuple*** όπου ν>0 και ν πεπερασμένος αριθμός), είναι μια ακολουθία από ν όρους. Η τιμή και η θέση κάθε όρου είναι σημαντική.

Για παράδειγμα, οι πλειάδες <b, Y, Z> και <Y, b, Z> είναι διαφορετικές παρόλο που οι τιμές των στοιχείων είναι ίδιες. Δεν συμβαίνει όμως το ίδιο με τη θέση των στοιχείων. Έτσι, το στοιχείο b βρίσκεται στην $1^{η}$ θέση της πρώτης πλειάδας και στην $2^{η}$ θέση της δεύτερης πλειάδας. Ενώ το στοιχείο Y βρίσκεται στην $2^{η}$ θέση της πρώτης πλειάδας και στην $1^{η}$ θέση της δεύτερης πλειάδας.

Ορισμός 2.2

Ένας ***όρος (term)***, είναι είτε μια σταθερά, ή μια μεταβλητή, ή μια συνάρτηση εφαρμοζόμενη σε μια πλειάδα (tuple) όρων.

Παράδειγμα

Σταθερά	Μεταβλητή	Συνάρτηση
a	X	f(b,f(b, X, Y), Y)

Ορισμός 2.3

Ένας ***ατομικός τύπος (atomic formula)***, είναι ένα κατηγόρημα εφαρμοζόμενο σε μια πλειάδα όρων.

Παραδείγματα

1. has(yannis, book): το κατηγόρημα has/2 εφαρμόζεται στους όρους "yannis"και "book".
2. is(X, mother(maria)): το κατηγόρημα is/2 εφαρμόζεται στους όρους "X"και "mother(maria)".

Ορισμός 2.4

Ένας ***στοιχειώδης τύπος (literal)***, είναι είτε ένας ατομικός τύπος p ή ένας αρνητικός ατομικός τύπος ¬p. Για παράδειγμα τα p(X,Y), ¬p(X,Y), q(a,Z), ¬q(a,Z) είναι στοιχειώδεις τύποι.

Ορισμός 2.5

Μία ***πρόταση Horn (Horn clause)*** έχει την μορφή:

$h :- l_1, l_2, \ldots, l_n$

όπου το h είναι ένας ατομικός τύπος και τα $l_1, l_2, \ldots, l_n$ είναι στοιχειώδεις τύποι. Επίσης, το h ονομάζεται *κεφαλή (head)* της πρότασης και το τμήμα $l_1, l_2, \ldots, l_n$ *σώμα (body)*.

Αν n≥0 η πρόταση ονομάζεται *κανόνας (rule),* ενώ αν n=0 ονομάζεται *γεγονός (fact)* (ή μοναδιαία πρόταση -unit clause) και γράφεται, απλά, h. Ένας στόχος (ή ερώτημα - query) σ' ένα λογικό πρόγραμμα, συντάσσεται ως «?- $l_1,\ldots, l_i, \ldots, l_n$.», όπου l_i στοιχειώδης τύπος και $1 \leq i \leq n$. Κάθε πρόταση ή στόχος τερματίζεται από μια τελεία. Όπως φαίνεται στις παραπάνω προτάσεις, στα λογικά προγράμματα το σύμβολο της συνεπαγωγής «←» μιας πρότασης Horn γράφεται «:-».

Μια συλλογή προτάσεων που έχουν το ίδιο όνομα κατηγορήματος για κεφαλή ορίζει μια ***σχέση (relation)*** ή μια ***διαδικασία (procedure)***.

Οι λογικοί τελεστές ***and***, ***or*** και ***not*** αντικαθίστανται από τα «,», «;» και «\+» αντίστοιχα (όπως στην Prolog) για ομοιομορφία στον συμβολισμό. Αυτό αποσκοπεί στην απλοποίηση της πτυχιακής εργασίας.

Προτάσεις σε μορφή κανόνων χρησιμοποιούνται στην Prolog όταν ένας ατομικός τύπος εξαρτάται από κάποιους άλλους στοιχειώδεις τύπους. Η γενική μορφή ενός κανόνα είναι η εξής:

$A :- B_1, B_2,\ldots,B_n$ όπου n≥0

το κόμμα «,» παριστά το λογικό ***and***. Α είναι ατομικός τύπος και ονομάζεται η κεφαλή του κανόνα. B_i όπου i = 1,…,n είναι στοιχειώδεις τύποι (αρνητικοί ή θετικοί ατομικοί τύποι) και αποτελούν το σώμα του κανόνα. Όταν ένας κανόνας εκτελείται τα Α και B_i (i = 1,…,n) είναι στόχοι. Η αλήθεια του στόχου Α προϋποθέτει την αλήθεια όλων των στόχων B_i (i = 1,…,n). Η δηλωτική ερμηνεία του παραπάνω κανόνα είναι ότι «η αλήθεια του ατομικού τύπου της κεφαλής του κανόνα, δηλαδή του Α, είναι λογική συνέπεια της αλήθειας της σύζευξης των στοιχειωδών τύπων του σώματος του, δηλαδή των $B_1, B_2,\ldots,B_n$». Ενώ η διαδικαστική ερμηνεία του παραπάνω κανόνα είναι ότι «για ν' αποδειχθεί ότι ο ατομικός τύπος Α είναι αληθής πρέπει να αποδειχθεί ότι η σύζευξη των στοιχειωδών τύπων B_1, B_2, B_n να είναι αληθής».Το σύμβολο «**:-**» της Prolog αντιστοιχεί στο σύμβολο «←» ή ***if*** της λογικής.

Οι μεταβλητές που υπάρχουν στη κεφαλή κανόνων πιθανόν να επαναλαμβάνονται στο σώμα μιας πρότασης έχουν καθολική δέσμευση, ενώ οι μεταβλητές που εμφανίζονται ***μόνο*** στο σώμα μιας πρότασης έχουν υπαρξιακή δέσμευση. Επιπλέον, η εμβέλειά μιας

μεταβλητής είναι όλος ο κανόνας. Αυτό σημαίνει ότι ίδιο όνομα μεταβλητής σε διαφορετικούς κανόνες αντιστοιχεί σε διαφορετικές μεταβλητές. Συνεπώς, η μετονομασία μιας μεταβλητής σ' ένα κανόνα δεν αλλάζει την σημασιολογία του προγράμματος.

Για να ήταν πληρέστερη η βάση γνώσεων του παραδείγματος στο Πρόγραμμα 2.1 θα έπρεπε να υπάρχουν επιπλέον τα γεγονότα γ_4 μέχρι γ_9 για να εκφράσουν τη σχέση ότι «ο Χ είναι παιδί του Υ».

γ_4: paidi(kostas, yannis).
γ_5: paidi(kostas, maria).
γ_6: paidi(anna, kostas).
γ_7: paidi(anna, soula).
γ_8: paidi(rania, kostas).
γ_9: paidi(rania, soula).

Η σχέση paidi(X, Y) είναι αληθής εάν «*ο/η Χ είναι παιδί του/της Υ*». Αυτή η σχέση εξαρτάται από τις σχέσεις pateras/2 και mitera/2. Δηλαδή, «*ο/η Χ είναι παιδί του/της Υ εάν ο Υ είναι πατέρας του/της Χ ή η Υ είναι μητέρα του/της Χ*». Δεν χρειάζεται να προσθέσουμε τα γεγονότα γ_4, γ_5, γ_6, γ_7, γ_8 και γ_9 στο Πρόγραμμα 2.1, αλλά να προσθέσουμε τους κανόνες κ_1 και κ_2, οι οποίοι εκφράζουν την εξάρτηση της σχέσης paidi/2 από τις σχέσεις pateras/2 και mitera/2. Οι κανόνες κ_1 και κ_2 δημιουργούν τα γεγονότα γ_4 μέχρι γ_9 χρησιμοποιώντας τα ήδη υπάρχοντα γεγονότα στο Πρόγραμμα 2.1.

κ_1: paidi(X, Y) :- pateras(Y, X).

κ_2: paidi(X, Y) :- mitera(Y, X).

Οι κανόνες κ_1 και κ_2 μπορούν να αντικατασταθούν από τον λογικά ισοδύναμο κανόνα κ.

κ: paidi(X, Y) :- pateras(Y, X); mitera(Y, X).

Το σύμβολο «;» είναι το λογικό ***or***. Θεωρήσατε την ερώτηση «ποιά είναι τα παιδιά του Κώστα;» ή σε Prolog «?- paidi(X, kostas)». Για ν' απαντήσει αυτή την ερώτηση η Prolog πρέπει να ταυτοποιήσει τον στοιχειώδη τύπο της ερώτησης με την κεφαλή του κανόνα κ. Η μεταβλητή Χ του κανόνα δεσμεύεται με την τιμή kostas. Έστω η ερώτηση «?- paidi(yannis, Y)» και ο κανόνας κ, η ταυτοποίηση της ερώτησης με τον κανόνα κ σημαίνει ότι για να είναι αληθής η ερώτηση «?- paidi(yannis, Y).», θα πρέπει να είναι αληθής η διάζευξη των στοιχειωδών τύπων του σώματος «pateras(Y, yannis); mitera(Y, yannis)». Δηλαδή, η ερώτηση «?- paidi(yannis, Y).» αντικαθίσταται από την ερώτηση «?- pateras(Y, yannis); mitera(Y, yannis).», η αλήθεια της οποίας συνεπάγεται την αλήθεια της αρχικής ερώτησης.

2.2 Βασικές Έννοιες Λογικής και Λογικού Προγραμματισμού

Η διαφορά του λογικού προγραμματισμού και της γλώσσας Prolog από τον παραδοσιακό προγραμματισμό και γλώσσες όπως Fortran, Basic, Cobol, Pascal κτλ. βρίσκεται στις θεμελιώδεις αρχές του λογικού προγραμματισμού, τόσο στο σχεδιασμό, όσο και στην εκτέλεση ενός λογικού προγράμματος.

Ένα πρόγραμμα αποτελείται από δυο δομικά στοιχεία, τους ***αλγόριθμους*** και τις ***δομές δεδομένων***. Ο αλγόριθμος αποτελείται από την ***λογική*** και τον ***έλεγχο***. Με τον όρο λογική χαρακτηρίζουμε όλες εκείνες τις συντακτικές έννοιες που προσδιορίζουν το ***τι***

κάνει ένα πρόγραμμα, ενώ με τον όρο έλεγχο όλες εκείνες τις συντακτικές έννοιες, που προσδιορίζουν το ***πως*** το κάνει. Εκφράζοντας όλα τα παραπάνω με δυο εξισώσεις έχουμε:

Αλγόριθμος = Λογική + Έλεγχος

Πρόγραμμα = Αλγόριθμος + Δομές Δεδομένων

Ένα πρόγραμμα γραμμένο στη Basic ή σε κάποια άλλη γλώσσα του παραδοσιακού προγραμματισμού αποτελείται από εντολές οι οποίες περιγράφουν ενέργειες που πρέπει να εκτελεστούν βήμα προς βήμα από τον υπολογιστή για να έχει το πρόγραμμα το επιθυμητό αποτέλεσμα. Οι γλώσσες προγραμματισμού όπως η Basic χαρακτηρίζονται από *προστακτικές (imperative)* εντολές που περιγράφουν βήμα προς βήμα την συμπεριφορά του προγράμματος, έτσι ώστε, μετά από μια πεπερασμένη ακολουθία τέτοιων εντολών, να επιτυγχάνεται το σωστό και αναμενόμενο αποτέλεσμα.

Γενικά, ένα πρόγραμμα σε μια παραδοσιακή γλώσσα προγραμματισμού, εκφράζει μια *απεικόνιση (function)* από τα δεδομένα (input) στο αποτέλεσμα (output) του προγράμματος, ενώ ένα πρόγραμμα σε μια γλώσσα Λογικού Προγραμματισμού, εκφράζει μια *σχέση (relation)* μεταξύ των δεδομένων. Επειδή οι σχέσεις είναι γενικότερες από τις απεικονίσεις (οι σχέσεις ***δεν*** είναι απαραίτητα μονοσήμαντες, ενώ οι απεικονίσεις είναι), οι δυνατότητες του Λογικού Προγραμματισμού είναι μεγαλύτερες από τις δυνατότητες του κλασικού προγραμματισμού.

Η επιλογή των κατηγορημάτων και των σχέσεων που εκφράζονται με αυτά τα κατηγορήματα αποτελούν στην Prolog τη λογική. Ο έλεγχος είναι αφενός η σειρά με την οποία ερευνάται η βάση γνώσεων και η σειρά με την οποία εφαρμόζεται η επίλυση (revolution) και αφετέρου μερικά δομικά στοιχεία ελέγχου, όπως η αποκοπή (!), τα οποία προσφέρονται σαν συντακτικά αντικείμενα της Prolog. Η Prolog δηλαδή είναι μια περιγραφική γλώσσα, η οποία διαθέτει κάποιο μηχανισμό ελέγχου.

Ας δούμε για παράδειγμα ένα πρόγραμμα που διαβάζει δύο αριθμούς και τυπώνει τον μεγαλύτερο. Θα δώσουμε πρώτα το πρόγραμμα σε Basic και μετά το ίδιο ακριβώς πρόγραμμα σε Prolog:

Πρόγραμμα Basic

```
INPUT  "NUMBER1", X
INPUT  "NUMBER2", Y
IF X > Y THEN
PRINT X
ELSE
PRINT Y
ENDIF
```

Πρόγραμμα Prolog

```
program :- write("NUMBER1"), read(X), real(X), nl,
          write("NUMBER2"),read(Y),real(Y),nl,
          greater(X, Y, Z), write(Z).
greater(X, X, X).
greater(X, Y, Y):- X < Y.
greater(X, Y, X):- X > Y.
```

Το «real» είναι ένα ειδικό κατηγόρημα της Prolog που ελέγχει αν ένας αριθμός είναι πραγματικός και το κατηγόρημα "nl" συνεχίζει την έξοδο στην επόμενη γραμμή εξόδου. Δηλαδή κάθε αριθμός γράφεται σε διαφορετική σειρά κατά την εκτύπωση.

Το πρόγραμμα σε Basic είναι μόνο μια ακολουθία εντολών. Αυτές οι εντολές που εκτελούνται με την σειρά που υποδεικνύει το πρόγραμμα, αποτελούν τον έλεγχο, δηλαδή το σχεδιασμό και τη ροή του προγράμματος. Το στοιχείο της λογικής σε αυτό το πρόγραμμα βρίσκεται στη σχέση ">". Αντίθετα, το πρόγραμμα Prolog είναι ένα σύνολο από προτάσεις (clauses), που περιγράφουν πλήρως τη σχέση που καθορίζει τη διάταξη δύο αριθμών, δηλαδή το κατηγόρημα "greater". Αυτό το σύνολο των τύπων εκφράζει τη λογική, η οποία έχει και τον κυρίαρχο ρόλο σε ένα πρόγραμμα Prolog, ενώ ο έλεγχος

βρίσκεται στη σειρά που διατάσσουμε ή και ορίζουμε τα κατηγορήματα, καθώς και ο τρόπος με τον οποίο η Prolog εκτελεί το στόχο μας, για παράδειγμα " ?- program".

Ένα λογικό πρόγραμμα αποτελείται από προτάσεις γεγονότα ή γεγονότα (facts), από προτάσεις κανόνες ή κανόνες (rules) και από ερωτήσεις (queries). Τα γεγονότα και οι κανόνες καθορίζουν τις σχέσεις μεταξύ των αντικειμένων. Ουσιαστικά, τα γεγονότα και οι κανόνες είναι τα αξιώματα του συστήματος. Οι ερωτήσεις πρέπει να απαντηθούν από τα γεγονότα και τους κανόνες. Ουσιαστικά οι ερωτήσεις είναι αρνητικά θεωρήματα τα οποία πρέπει να αποδειχθούν.

Συνεπώς ο προγραμματισμός σε Prolog περιλαμβάνει τα εξής στοιχεία:

1. Δήλωση γεγονότων για αντικείμενα και τις μεταξύ τους σχέσεις.
2. Ορισμός κανόνων για τα αντικείμενα και τις μεταξύ τους σχέσεις.
3. Ερωτήσεις για τα αντικείμενα και τις μεταξύ τους σχέσεις που υπάρχουν στη βάση γνώσεων.

2.2.1 Ορολογία και Συμβολισμός στο Λογικό Προγραμματισμό

Οι ***Μεταβλητές*** σε λογικά προγράμματα αντιπροσωπεύουν μια οντότητα η οποία δεν έχει καθοριστεί, ενώ στις συμβατικές γλώσσες προγραμματισμού οι μεταβλητές αντιπροσωπεύουν θέσεις μνήμης. Όταν μια μεταβλητή Χ αντιπροσωπεύει κάποιο αντικείμενο λέμε ότι είναι ***δεσμευμένη***. Όταν μια μεταβλητή Χ δεν αντιπροσωπεύει κάποιο αντικείμενο λέμε ότι δεν είναι δεσμευμένη. Το όνομα μιας μεταβλητής σε Prolog μπορεί να σχηματιστεί από τα κεφαλαία και τα πεζά γράμματα του λατινικού αλφάβητου και από την υπογράμμιση (_). Ο πρώτος χαρακτήρας του ονόματος μιας μεταβλητής πρέπει να είναι είτε κεφαλαίο γράμμα ή η υπογράμμιση (_).

Θεωρούμε ως σύμβολα ***σταθερές*** ονόματα που έχουν το πρώτο γράμμα τους πεζό και ακολουθεί είτε ψηφίο ή υπογράμμιση ή πεζό ή κεφαλαίο γράμμα π.χ. a,s1,const,…

Σύμβολα ***συναρτήσεων*** είναι ονόματα που έχουν το πρώτο γράμμα τους πεζό και ακολουθεί είτε ψηφίο ή υπογράμμιση ή πεζό ή κεφαλαίο γράμμα, π.χ. a,s1,const,… Επιπλέον ο συμβολισμός f/ν χρησιμοποιείται για τις συναρτήσεις, όπου f είναι το όνομα της συνάρτησης, και ν η πληθυκότητά της, δηλαδή το πλήθος των ορισμάτων της.

Σύμβολα ***μεταβλητών*** είναι ονόματα που έχουν το πρώτο γράμμα τους κεφαλαίο και ακολουθεί είτε ψηφίο ή υπογράμμιση ή πεζό ή κεφαλαίο γράμμα,: X,Y,Z,X1,…

Σύμβολα ***κατηγορημάτων*** είναι ονόματα με το πρώτο γράμμα πεζό και ακολουθεί είτε ψηφίο ή υπογράμμιση ή πεζό ή κεφαλαίο γράμμα, όπως: αρέσει, είναι, έχει, γονιός, πατέρας, κτλ… Ο συμβολισμός p/ν χρησιμοποιείται και για τα κατηγορήματα όπως και για τις συναρτήσεις. Δηλαδή, όπου p είναι το όνομα του κατηγορήματος, και ν το πλήθος των ορισμάτων του.

2.2.2 Βασικές Έννοιες Προτασιακού και Κατηγορηματικού Λογισμού

Ένας από τους πρώτους τρόπους αναπαράστασης γνώσεων ήταν η λογική. Η πιο θεμελιώδης έννοια στη λογική είναι η αλήθεια. Μία πρόταση μπορεί να έχει δύο πιθανές τιμές είτε αληθής ή ψευδής.

Ο προτασιακός λογισμός είναι η πιο απλή μορφή λογικής, ασχολείται με την αναπαράσταση πληροφοριών σαν προτάσεις καθώς και με την εξαγωγή συμπερασμάτων από προτάσεις. Ο προτασιακός λογισμός είναι μια συμβολική λογική, η οποία ασχολείται με τις λογικές ιδιότητες συνθέτων προτάσεων. Μία πρόταση μπορεί να έχει μια τιμή από τις τιμές αληθείας, *αληθής* και *ψευδής*.

Παράδειγμα

Πρόταση	*Τιμή αληθείας*
Η Κρήτη είναι νησί	αληθής
Η Πελοπόννησος είναι νησί	ψευδής

Ενώ οι εκφράσεις, α) δυο συν τρία και β) ο πατέρας του Κώστα, δεν είναι προτάσεις κατά συνέπεια δεν μπορούμε να τους δώσουμε κάποια τιμή αληθείας.

Απλές προτάσεις μπορούν να συνδέονται με λογικούς συνδέσμους για σχηματισμό πιο σύνθετων προτάσεων. Τα πιο συνηθισμένα λογικά σύμβολα είναι τα εξής:

Σύμβολο λογικού συνδέσμου	**Έννοια**	**Αγγλικά σύμβολα και έννοια**
$\wedge$ (και)	σύζευξη	and
$\vee$ (ή)	διάζευξη	or
$\neg$ (όχι)	άρνηση	not
$\rightarrow$ (εάν.. τότε..)	συνεπαγωγή	implies, if..then, only if
$\leftrightarrow$ (εάν και μόνο εάν)	ισοδυναμία	equivalent, if and only if

Η χρήση λογικών συνδέσμων σε προτάσεις δημιουργεί την πιο απλή μορφή λογικής, τον προτασιακό λογισμό. Με τον προτασιακό λογισμό μπορούμε να εκφράσουμε σύνθετες προτάσεις.

Παράδειγμα

Ο Γιάννης σπουδάζει Πληροφορική και βρίσκεται στο τρίτο έτος. Εάν ο Γιάννης είναι τριτοετής σπουδαστής Πληροφορικής τότε έχει εγγραφεί στο μάθημα Τεχνητή Νοημοσύνη.

Το κύριο πρόβλημα του προτασιακού λογισμού είναι ότι μελετά μόνο πλήρεις προτάσεις και δεν μπορεί να εξετάσει την εσωτερική δομή μιας πρότασης. Για παράδειγμα ο προτασιακός λογισμός δεν μπορεί να αποδείξει την ορθότητα του εξής συλλογισμού :

Όλοι οι άνθρωποι είναι θνητοί

Όλοι οι Έλληνες είναι άνθρωποι

Συνεπώς, όλοι οι Έλληνες είναι θνητοί

Για να μπορεί ένας φορμαλισμός να εκφράσει επαρκώς την γνώση του χώρου δεν αρκεί μόνο να μπορεί να εκφράζει την αλήθεια ή μη προτάσεων αλλά θα πρέπει επιπλέον, α) να μιλάει για τα αντικείμενα ή οντότητες του χώρου, β) να εκφράζει τις σχέσεις μεταξύ αυτών των αντικειμένων και γ) να γενικεύει αυτές τις σχέσεις σε κλάσεις αντικειμένων.

Ο *κατηγορηματικός λογισμός* ή κατηγορηματική λογική πρώτης-τάξεως περιέχει, εκτός από τα συστατικά του προτασιακού λογισμού, επιπλέον όρους (terms), κατηγορήματα (predicates) και ποσοδείκτες (quantifiers).

Πεδίο ή πεδίο ορισμού είναι ένα σύνολο από αντικείμενα ή οντότητες ή ιδέες κτλ, όπως Γιάννης, πέντε, κτλ. Εάν το πεδίο ενός προβλήματος είναι $Π=\{α_1, ..., α_κ\}$ τότε κάθε $α_i$, $1≤i≤κ$ ονομάζεται σταθερά ή αντικείμενο ή οντότητα. Κάθε οντότητα είναι ένας *όρος*.

Παράδειγμα

α) Το σύνολο Π={γιάννης, μαρία, ελένη, κώστας} παριστά το πεδίο ενός προβλήματος σχέσεων οικογένειας

β) Το σύνολο των ακέραιων αριθμών Z={..,-1,0,1,..} μπορεί να παριστά το πεδίο σ' ένα αριθμητικό πρόβλημα.

Κατηγορήματα είναι ισχυρισμοί για αντικείμενα (σταθερές). Κάθε κατηγόρημα εκφράζεται σαν μια διατεταγμένη Ν-άδα αντικειμένων τα οποία λέγονται *ορίσματα*. Η τιμή του κατηγορήματος είναι είτε αληθής ή ψευδής. Το πλήθος των ορισμάτων ενός κατηγορήματος λέγεται *πληθυκότητα* ή *βαθμός*. Κατηγορήματα με πληθυκότητα 1 ονομάζονται *ιδιότητες*. Ένα κατηγόρημα p με πληθυκότητα ν συμβολίζεται με p/ν.

Παραδείγματα

1. Ο ισχυρισμός, «ο Γιάννης είναι πατέρας της Μαρίας» παριστάνεται από το εξής κατηγόρημα: *πατέρας(γιάννης, μαρία)*

   Όπου *πατέρας* είναι το όνομα του κατηγορήματος με πληθυκότητα *2, γιάννης* και *μαρία* είναι σταθερές ή αντικείμενα από το πεδίο του προβλήματος.

2. Ο ισχυρισμός «το γινόμενο 3 επί 5 είναι 15» παριστάνεται από το εξής κατηγόρημα

   γινόμενο (3,5,15)

   Όπου *γινόμενο* είναι το όνομα του κατηγορήματος με πληθυκότητα 3 επιπλέον 3,5,15 είναι σταθερές από το πεδίο του προβλήματος που είναι οι ακέραιοι αριθμοί (το σύνολο *Z)*.

Μια μεταβλητή Χ είναι ένας όρος ο οποίος μπορεί να πάρει οποιαδήποτε τιμή από το πεδίο του προβλήματος. Εάν το πεδίο Π ενός προβλήματος είναι το σύνολο των ανθρώπων Π={γιάννης, νίκος, άννα, μαρία} τότε η μεταβλητή Χ μπορεί να πάρει σαν τιμή οποιοδήποτε στοιχείο του Π. Στο κατηγόρημα πατέρας(Χ,Υ) οι μεταβλητές Χ και Υ μπορούν να πάρουν οποιαδήποτε τιμή από το σύνολο Π.

Οι συναρτήσεις, όπως τα κατηγορήματα, εφαρμόζονται σε διατεταγμένες Ν-άδες αντικειμένων τα οποία λέγονται ορίσματα. Επιπλέον, οι συναρτήσεις επιστρέφουν μια τιμή από το πεδίο ορισμού τους. Το πλήθος των ορισμάτων μιας συνάρτησης ονομάζεται *πληθυκότητα ή βαθμός*. Εάν f είναι μια συνάρτηση με πληθυκότητα ν, αυτό συμβολίζεται με f/ν.

Θα χρησιμοποιήσουμε τα εξής σύμβολα για σταθερές, μεταβλητές, συναρτήσεις και ονόματα κατηγορημάτων.

Οι *σταθερές* θ' αρχίζουν με πεζό γράμμα του Ελληνικού ή του Αγγλικού αλφαβήτου και ακολουθεί είτε ψηφίο ή υπογράμμιση ή πεζό ή κεφαλαίο γράμμα του Ελληνικού ή του Λατινικού αλφάβητου. Επιπλέον σταθερές είναι και οι αριθμοί. Για παράδειγμα, γιάννης, κόκκινο, 4, 5.46 κτλ.

Οι *μεταβλητές* θ' αρχίζουν με κεφαλαίο γράμμα του Ελληνικού ή του Αγγλικού αλφαβήτου και ακολουθεί είτε ψηφίο ή υπογράμμιση ή πεζό ή κεφαλαίο γράμμα του Ελληνικού ή του Λατινικού αλφάβητου. Για παράδειγμα, Χ, Υ, Ηλικία, Αριθμός κτλ.

Σύμβολα συναρτήσεων. Σύμβολα συναρτήσεων αρχίζουν με μικρό γράμμα του Ελληνικού ή του Αγγλικού αλφαβήτου και ακολουθεί είτε ψηφίο ή υπογράμμιση ή πεζό ή κεφαλαίο γράμμα του Ελληνικού ή του Λατινικού αλφάβητου. Για παράδειγμα, f, g, συν, επί, πατέρας κτλ.

Σύμβολα κατηγορημάτων. Σύμβολα κατηγορημάτων αρχίζουν με μικρό γράμμα του Ελληνικού ή του Αγγλικού αλφαβήτου και ακολουθεί είτε ψηφίο ή υπογράμμιση ή πεζό ή κεφαλαίο γράμμα του Ελληνικού ή του Λατινικού αλφάβητου. Για παράδειγμα, p, q, r, μεγαλύτερο, αδέλφια, είναι_σπουδαστής κτλ.

Ορισμός 2.6
Ένας όρος ορίζεται αναδρομικά ως εξής.

1. Μία σταθερά είναι ένας όρος.
2. Μία μεταβλητή είναι ένας όρος.
3. Εάν f είναι ένα σύμβολο συνάρτησης με ν-ορίσματα, και t_1, ..., t_v είναι όροι τότε $f(t_1,..., t_v)$ είναι ένας όρος.

Ορισμός 2.7

Εάν p είναι ένα σύμβολο κατηγορήματος ν-ορισμάτων και t_1, ..., t_v είναι όροι, τότε $p(t_1, ..., t_v)$ είναι ένας *ατομικός τύπος*.Για παράδειγμα, αδέλφια(γιάννης, μαρία) και μεγαλύτερο(επί(3,7),12) είναι ατομικοί τύποι ή άτομα.

Ποσοδείκτες (Quantifiers)

Θεωρήσετε τις εξής προτάσεις, «όλοι οι άνθρωποι είναι θνητοί» και «μερικοί άνθρωποι είναι σπουδαστές». Για να εκφραστούν προτάσεις αυτής της μορφής έχουν εισαχθεί στον κατηγορηματικό λογισμό ο καθολικός ∀ και ο υπαρξιακός ∃ ποσοδείκτης. *Ένας ποσοδείκτης δεικνύει πόσο συχνά κάποιος ισχυρισμός είναι αληθής.*

Ο *καθολικός ποσοδείκτης* δεικνύει ότι μια πρόταση είναι αληθής για όλα τα αντικείμενα στα οποία εφαρμόζεται. Για παράδειγμα, ∀Χ θνητός(Χ) σημαίνει για όλες τις τιμές του Χ ο ισχυρισμός θνητός(Χ) είναι αληθής.

Ο *υπαρξιακός ποσοδείκτης* δεικνύει ότι ένας ισχυρισμός είναι αληθής για τουλάχιστον ένα αντικείμενο στο οποίο εφαρμόζεται. Για παράδειγμα, ∃Χ είναι_σπουδαστής(Χ) σημαίνει ότι για μία τουλάχιστον τιμή του Χ ο ισχυρισμός είναι_σπουδαστής(Χ) είναι αληθής.

Έτσι με τα κατηγορήματα και τους ποσοδείκτες μπορούμε να εκφράσουμε σε λογική περισσότερο πολύπλοκες προτάσεις. Για παράδειγμα, ο ισχυρισμός «όλοι οι σπουδαστές είναι μεγαλύτεροι των 18 χρονών» παριστάνεται σε λογική ως εξής:

∀Χ(είναι_σπουδαστής(Χ)→ μεγαλύτερος_από(Χ, 18)).

2.2.3 Ταυτοποίηση

Η πράξη της ταυτοποίησης εξετάζει εάν δύο όροι Τ1 και Τ2 ταιριάζουν δεσμεύοντας μερικές από τις μεταβλητές των δύο όρων Τ1 και Τ2. Κάθε μεταφραστής Prolog περιέχει μια διαδικασία ταυτοποίησης όρων της εξής μορφής: unify(T1, T2, Theta), όπου Τ1 και

Τ2 είναι οι όροι τους οποίους ταυτοποιεί και Theta είναι ένα σύνολο από δεσμεύσεις μεταβλητών των όρων Τ1 και Τ2, εφόσον οι όροι αυτοί ταυτοποιούνται. Σε διαφορετική περίπτωση επιστρέφει αποτυχία. Το Σχήμα 2.1 δίνει παραστατικά την πράξη της ταυτοποίησης.

Σχήμα 2.1: *Ταυτοποίηση όρων.*

Εάν εφαρμοστούν οι αντικαταστάσεις του Theta στους όρους Τ1 και Τ2, τότε οι όροι Τ1 και Τ2 θα γίνουν ίδιοι.

Ο πίνακας 2.1 δίνει τους κανόνες ενός απλού αλγορίθμου ταυτοποίησης για εισαγωγική κατανόηση του αντικειμένου [Cohen 1985]. Στον πίνακα 2.1 ο συμβολισμός Χ/Τ σημαίνει ότι η μεταβλητή Χ δεσμεύεται με την τιμή Τ, όπου το Τ μπορεί να είναι, είτε μια σταθερά ή μια μεταβλητή ή ένας σύνθετος όρος.

Όρος Τ2 / **Όρος Τ1**	σταθερά C2	μεταβλητή Χ2	σύνθετος όρος
Σταθερά C1	επιτυχία εάν C1 = C2	επιτυχία με Χ2 / C1	αποτυχία
Μεταβλητή Χ1	επιτυχία με δέσμευση Χ1/C2	επιτυχία με δέσμευση Χ1/Χ2	επιτυχία με δέσμευση Χ1/Τ2
Σύνθετος όρος	αποτυχία	επιτυχία με δέσμευση Χ2/Τ1	Επιτυχία εάν: 1) Οι όροι Τ1 και Τ2 έχουν ίδιο όνομα συνάρτησης και ίδια πλυθηκότητα. 2) Η ταυτοποίηση όλων των αντίστοιχων ορισμάτων είναι επιτυχής.

Πίνακας 2.1:*Κανόνες Αλγόριθμου Ταυτοποίησης.*

Ορισμός 2.8

Μια αντικατάσταση θ_1 θα είναι περισσότερο γενική από μια αντικατάσταση θ_2, συμβολίζεται $\theta_2 \leq \theta_1$ εάν υπάρχει αντικατάσταση θ_3 ώστε να ισχύει $\theta_2 = \theta_1 \circ \theta_3$.

Παράδειγμα

Η αντικατάσταση $\theta_1 = \{X/f(Z),Y/b\}$ είναι περισσότερο γενική από την αντικατάσταση $\theta_2 = \{X/f(a),Y/b\}$ γιατί εάν $\theta_3 = \{Z/a\}$ τότε $\theta_2 = \theta_1 \circ \theta_3$.

Ορισμός 2.9

Έστω E_1 και E_2 δυο απλές εκφράσεις. Ένας ταυτοποιητής των E_1 και E_2 είναι μια αντικατάσταση θ τέτοια ώστε $E_1\theta=E_2\theta$.

Παράδειγμα

Εάν $E_1 = p(X,Y)$ και $E_2 = p(Z,a)$ είναι δυο εκφράσεις τότε ένας ταυτοποιητής θ_1 των E_1 και E_2 είναι ο $\theta_1 = \{X/b, Y/a, Z/b\}$ έτσι ώστε $E_1\theta_1 = E_2\theta_1 = p(b,a)$. Ένας άλλος ταυτοποιητής θα ήταν ο $\theta_2 = \{X/a, Y/a, Z/a\}$ τέτοιος ώστε $E_1\theta_2 = E_2\theta_2 = p(a,a)$. Ομοίως, ταυτοποιητής είναι και ο $\theta_3 = \{X/Z, Y/a\}$ για τον οποίο έχουμε $E_1\theta_3 = E_2\theta_3 = p(Z,a)$.

Τέλος, ταυτοποιητής των E_1 και E_2 είναι και ο $\theta_4 = \{X/f(W), Y/a, Z/f(W)\}$ για τον οποίο προκύπτει $E_1\theta_4 = E_2\theta_4 = p(f(W), a)$.

Ορισμός 2.10

Ένας ταυτοποιητής θ_1 των απλών εκφράσεων E_1 και E_2 ονομάζεται ο πλέον γενικός ταυτοποιητής (πγτ) ή most general unifier (mgu) εάν για κάθε άλλο ταυτοποιητή θ_2 των E_1 και E_2 υπάρχει μια αντικατάσταση θ_3 τέτοια ώστε να ισχύει $\theta_2 = \theta_1 \circ \theta_3$.

Παράδειγμα

Από τους παραπάνω ταυτοποιητές, θ_1, θ_2, θ_3, θ_4 ο ταυτοποιητής θ_3 είναι ο πλέον γενικός ταυτοποιητής των εκφράσεων E_1 και E_2 καθώς ισχύει:

α) $\theta_3 \circ \{Z/b\} = \{X/b, Y/a, Z/b\} = \theta_1$.

β) $\theta_3 \circ \{Z/a\} = \{X/a, Y/a, Z/a\} = \theta_2$.

γ) $\theta_3 \circ \{Z/f(W)\} = \{X/f(W), Y/a, Z/f(W)\} = \theta_4$.

Υπάρχουν πολλά διαφορετικά είδη αλγορίθμων ταυτοποίησης διαθέσιμα για τον υπολογισμό του πλέον γενικού ταυτοποιητή. Ο περισσότερο διαδεδομένος, είναι ο αλγόριθμος ταυτοποίησης Robinson κατά τον οποίο εάν δοθούν σαν είσοδος δύο ατομικοί τύποι $A_1 = P(t_1,\ldots,t_v)$ και $A_2 = P(r_1,\ldots,r_v)$ ο αλγόριθμος βρίσκει εάν είναι ταυτοποιήσιμοι ή όχι. Εάν ναι επιστρέφει τον πγτ, διαφορετικά επιστρέφει αποτυχία. Ο αλγόριθμος χρησιμοποιεί μια στοιβάδα S στην οποία καταχωρούνται τα ζεύγη των αντίστοιχων όρων των A_1 και A_2, δηλαδή $S := [(t_1, r_1),\ldots,(t_v, r_v)]$.

Όταν σε ένα ζεύγος όρων (t, r) είτε το t είναι μεταβλητή και το r σύνθετος όρος ή αντίστροφα τότε ο αλγόριθμος κάνει τον έλεγχο-ύπαρξης γνωστό ως occur–check. Αυτός ο έλεγχος σκοπό έχει να μην επιτρέψει αυτοαναφερόμενες δεσμεύσεις όπως για παράδειγμα X/f(X). Μία τέτοια δέσμευση καταχωρεί στην μεταβλητή X ένα μη-πεπερασμένο όρο f(f(f(…))) ενώ όλες οι εκφράσεις πρέπει να είναι πεπερασμένες. Επειδή ο έλεγχος-ύπαρξης έχει υψηλό υπολογιστικό κόστος, για αυτό πολλές γλώσσες λογικού προγραμματισμού παραλείπουν τον έλεγχο-ύπαρξης από τον αλγόριθμο ταυτοποίησης που χρησιμοποιούν. Αυτή η παράλειψη έχει σαν συνέπεια ο αλγόριθμος ταυτοποίησης να χάσει την ορθότητά του.

Είσοδος: Οι δύο ατομικοί τύποι $A_1 = P(t_1,\ldots,t_v)$ και $A_2 = P(r_1,\ldots,r_v)$, οι οποίοι θα ταυτοποιηθούν.

Έξοδος: Η αντικατάσταση θ, ο πλέον γενικός ταυτοποιητής των E_1 και E_2, ή αποτυχία.

Αλγόριθμος:

Αρχική τιμή στην αντικατάσταση θ το κενό σύνολο, θ := { }.
Αρχική τιμή στη στοιβάδα S τα ζεύγη των όρων $(t_1, r_1),\ldots,(t_v,r_v)$,
S := [$(t_1, r_1),\ldots,(t_v, r_v)$].
Αρχική τιμή ψευδές στην μεταβλητή Αποτυχία, Αποτυχία := ψευδής.
Repeat loop
Πάρε την κορυφή (t, r) της στοιβάδας S;
if t και r είναι διαφορετικές μεταβλητές then
θ΄ = θ ◦ {t/r};
αντικατέστησε στην στοιβάδα S την μεταβλητή t με τον όρο r;
else if t μεταβλητή και r όρος (σύνθετος ή μη) στον οποίο δεν υπάρχει η μεταβλητή t
then
θ΄ = θ ◦ {t/r};
αντικατέστησε στην στοιβάδα S την μεταβλητή t με τον όρο r;
else if r μεταβλητή και t όρος (σύνθετος ή μη) στον οποίο δεν υπάρχει η μεταβλητή r
then
θ΄ = θ ◦ {r/t};
αντικατέστησε στην στοιβάδα S την μεταβλητή r με τον όρο t;
else if t και r είναι ίδιες σταθερές ή ίδιες μεταβλητές
then
συνέχισε;
else if t και r είναι σύνθετοι όροι με ίδιο όνομα συνάρτησης και ίδια πληθυκότητα κ,
$t = F(s_1,\ldots,s_K)$ και $r = F(u_1,\ldots,u_K)$
then
καταχώρησε στην στοιβάδα S τα ζεύγη των όρων $(s_1,u_1),\ldots,(s_K, u_K)$
else Αποτυχία : = αληθής
until (S = { }) or Αποτυχία;

if Αποτυχία then
έξοδος αποτυχία
else
έξοδος θ;

Αλγόριθμος 2.1:*Ταυτοποίηση.*

2.2.4 Αντικατάσταση

Ορισμός 2.11

Αντικατάσταση (substitution) είναι ένα σύνολο της μορφής : $\{X_1/t_1, \ldots, X_k/t_k\}$ όπου κάθε X_i $(1 \leq i \leq k)$ είναι μια μεταβλητή διαφορετική από τις υπόλοιπες και κάθε t_i είναι ένας όρος διαφορετικός από την μεταβλητή X_i. Κάθε στοιχείο της μορφής X_i/t_i ονομάζεται δέσμευση του X_i.

Ορισμός 2.12

Βασικός όρος είναι ένας όρος ο οποίος δεν περιέχει μεταβλητές.

Βασικός ατομικός τύπος είναι ένας ατομικός τύπος όλα τα ορίσματα του οποίου είναι βασικοί όροι.

Παράδειγμα

Έστω a, b σταθερές, X, Y μεταβλητές, f/1, g/2 συναρτήσεις και p/1, q/2 κατηγορήματα. Οι όροι b, f(a), g(a,b), f(f(b)) είναι βασικοί όροι. Οι όροι X, f(X), g(a,f(f(X))) δεν είναι βασικοί όροι. Οι ατομικοί τύποι p(a), p(f(b)), q(a,f(b)) είναι βασικοί τύποι ενώ οι p(X), p(f(X)), q(a,f(X)) δεν είναι βασικοί ατομικοί τύποι.

Ορισμός 2.13

Έστω η αντικατάσταση $\theta = \{X_1/t_1, \ldots, X_k/t_k\}$.

Η αντικατάσταση θ ονομάζεται ***βασική αντικατάσταση (ground substitution)*** εάν όλα τα t_i $(1 \leq i \leq k)$ είναι βασικοί όροι.

Η αντικατάσταση θ ονομάζεται ***αντικατάσταση μετονομασίας (renaming substitution)*** εάν όλα τα t_i $(1 \leq i \leq k)$ είναι μεταβλητές.

Η αντικατάσταση $\theta = \{\ \}$ ονομάζεται ***κενή ή ταυτοτική αντικατάσταση*** και συμβολίζεται με $\in$.

Παράδειγμα

Εάν a, b είναι σταθερές, X, Y, Z, W είναι μεταβλητές και f/1, g/2 είναι συναρτήσεις τότε η αντικατάσταση $\theta_1 = \{X/f(a), Y/g(a,f(b))\}$ είναι βασική αντικατάσταση ενώ η αντικατάσταση $\theta_2 = \{X/Z, Y/W\}$ είναι αντικατάσταση μετονομασίας.

Ορισμός 2.14

Μια έκφραση είναι είτε όρος, ή ένας στοιχειώδης τύπος, ή μια σύζευξη ή διάζευξη στοιχειωδών τύπων. Μια απλή έκφραση είναι είτε ένας όρος ή ένας ατομικός τύπος.

Ορισμός 2.15

Έστω E μια πρόταση και $\theta = \{X_1/t_1,\ldots,X_k/t_k\}$ μια αντικατάσταση. Ένα στιγμιότυπο (instance) Eθ του E λαμβάνεται εάν ταυτόχρονα αντικατασταθεί κάθε εμφάνιση του X_i στην E με t_i όπου $(1 \leq i \leq k)$. Η διαδικασία αυτή ονομάζεται εφαρμογή αντικατάστασης.

Παράδειγμα

Έστω η έκφραση $E = p(X) \vee \neg q(g(X, Y))$ και $\theta = \{X/a, Y/f(b)\}$. Τότε
$E\theta = p(a) \vee \neg q(g(a, f(b)))$.

2.2.5 Σύνθεση Αντικαταστάσεων

Ορισμός 2.16

Έστω ότι $\theta = \{X_1/t_1,\ldots,X_k/t_k\}$ και $\sigma = \{Y_1/s_1,\ldots,Y_\nu/s_\nu\}$ είναι δύο αντικαταστάσεις. Η σύνθεση $\theta\circ\sigma$ των θ και σ είναι η αντικατάσταση η οποία λαμβάνεται ως εξής:

$\{X_1/t_1\sigma,\ldots, X_k/t_k\sigma, Y_1/s_1,..., Y_\nu/s_\nu\}$

όπου διαγράφονται δεσμεύσεις $X_i/t_i\sigma$ για τις οποίες ισχύει $X_i=t_i\sigma$ $(1\leq i \leq k)$. Επίσης διαγράφονται δεσμεύσεις Y_j/s_j $(1\leq j \leq \nu)$ για τις οποίες ισχύει $Y_i \in \{X_1,\ldots,X_\kappa\}$.

Παραδείγματα

1. Έστω $\theta = \{X/a, Y/g(b,Z)\}$ και $\sigma = \{Z/b, W/f(a)\}$.
   Τότε $\theta\circ\sigma = \{X/a, Y/g(b,b), Z/b, W/f(a)\}$.
2. Έστω $\theta = \{X/f(Y), Y/a, Z/W\}$ και $\sigma = \{Y/f(a),W/Z, V/g(a,f(b))\}$.
   Τότε $\theta\circ\sigma = \{X/f(f(a)), Y/a, Y/f(a), W/Z, V/g(a,f(b))\}$.
3. Έστω $\theta = \{X/f(Y), Y/Z\}$ και $\sigma = \{X/john, Y/bill, Z/Y\}$.
   Τότε $\theta\circ\sigma = \{X/f(bill), Y/bill, Z/Y\}$.

Ορισμός 2.17

Μια αντικατάσταση θ είναι ***αυτοαπορροφητική (indempotent)*** εάν κατά την εφαρμογή της θ στον «εαυτό» της, προκύψει η ίδια θ $(\theta=\theta\circ\theta)$. Δηλαδή, για την αντικατάσταση $\theta = \{X_1/t_1,\ldots,X_k/t_k\}$ καμία μεταβλητή X_i $(1\leq i \leq k)$ δεν θα πρέπει να υπάρχει σε κάποιον από τους όρους $\{t_1,\ldots,t_k\}$. Κάθε νόμιμη αντικατάσταση θα πρέπει να ικανοποιεί την ιδιότητα της αυτοαπορρόφησης. Για παράδειγμα, η αντικατάσταση $\theta = \{X/Y, Y/a\}$ δεν είναι αυτοαπορροφητική, καθώς η εφαρμογή της θ στην θ είναι $\theta\circ\theta = \{X/a, Y/a\} \neq \theta$.

Το κριτήριο της αυτοαπορρόφησης δεν είναι το μοναδικό χαρακτηριστικό των αντικαταστάσεων. Θεωρούμε τις αντικαταστάσεις θ_1, θ_2, θ_3, την έκφραση Ε και τη κενή αντικατάσταση ϵ. Όλες οι αντικαταστάσεις πρέπει να ικανοποιούν τις ακόλουθες ιδιότητες:

$\theta_1\circ\epsilon = \epsilon\circ\theta_1=\theta_1$.

$(E\theta_1)\circ\theta_2=E(\theta_1\circ\theta_2)$.

$(\theta_1\circ\theta_2)\circ\theta_3=\theta_1\circ(\theta_2\circ\theta_3)$. Προσεταιριστική ιδιότητα της $\circ$

$E\circ\theta_1\circ\theta_2 \neq E\circ\theta_2\circ\theta_1$.

Η σύνθεση αντικαταστάσεων δεν ικανοποιεί την μεταθετική ιδιότητα.

2.3 Αναπαράσταση σε Μη-Βασική Μορφή

2.3.1 Εισαγωγή στον Μετα-Προγραμματισμό

Μετα-γλώσσα είναι μια γλώσσα που περιγράφει κάποια άλλη γλώσσα. Η περιγραφόμενη γλώσσα ονομάζεται γλώσσα-αντικείμενο. Για παράδειγμα, έστω ότι έχουμε την παρακάτω πρόταση «ο πληθυντικός των ουσιαστικών στην Αγγλική σχηματίζεται προσθέτοντας την κατάληξη –s πλην των ουσιαστικών που τελειώνουν σε –ch, -sh, -o, -x και –ss στα οποία προστίθεται η κατάληξη –es ». Στην πρόταση αυτή, η Ελληνική γλώσσα χρησιμοποιείται σαν μετα-γλώσσα για να περιγράψει την Αγγλική γλώσσα (γλώσσα-αντικείμενο).

Μετα-πρόγραμμα είναι ένα πρόγραμμα που δέχεται σαν δεδομένα ένα άλλο πρόγραμμα. Το πρόγραμμα που χρησιμοποιείται σαν δεδομένα ονομάζεται πρόγραμμα-αντικείμενο. Οι μεταφραστές (interpreters), οι μεταγλωττιστές (compilers), οι μετασχηματιστές προγραμμάτων, οι αναλυτές προγραμμάτων (analyzers), κτλ. είναι μετα-προγράμματα. Η γλώσσα στην οποία γράφεται ένας μεταφραστής ή ένας μεταγλωττιστής είναι η μετα-γλώσσα και η γλώσσα την οποία μεταφράζει ή μεταγλωττίζει είναι η γλώσσα αντικείμενο. Ο μετα-προγραμματισμός είναι εύκολος στην Prolog αλλά και γενικά στις γλώσσες του λογικού προγραμματισμού, επειδή προγράμματα και δεδομένα μπορούν να παρασταθούν με τον ίδιο τρόπο.

Δύο τρόποι υπάρχουν για να παραστήσουμε ένα πρόγραμμα-αντικείμενο.

1. Αναπαράσταση σε ***βασικούς όρους (ground representation).***
2. Αναπαράσταση σε ***μη-βασικούς όρους*** **(non-ground representation).**

Η σημαντικότερη διαφορά ανάμεσα στις δύο αναπαραστάσεις, είναι ότι στην αναπαράσταση σε *μη-βασικούς όρους,* η γλώσσα-αντικείμενο και η μετα-γλώσσα μπορούν να χρησιμοποιήσουν τα ίδια κατηγορήματα, τα ενσωματωμένα στη γλώσσα του λογικού προγραμματισμού, α) για *μετονομασία (renaming)* μεταβλητών, β) για *ταυτοποίηση (unification)* όρων και γ) για *εφαρμογή (apply) των αντικαταστάσεων* στους όρους και στους τύπους του προγράμματος-αντικείμενο. Επιπλέον, δεν υπάρχει ανάγκη για σαφή χειρισμό των αντικαταστάσεων *(σύνθεση αντικαταστάσεων)*. Αντίθετα, στην αναπαράσταση σε *βασικούς όρους,* τα μετα-προγράμματα δεν είναι τόσο αποτελεσματικά, επειδή οι μεταβλητές της γλώσσας-αντικείμενο πρέπει να παρασταθούν σαν σταθερές της μετα-γλώσσας. Αυτό συνεπάγεται ότι πρέπει να γραφούν πολύπλοκα προγράμματα, τα οποία θα μπορούν εκτελούν τις παραπάνω ενέργειες, δηλαδή α) την *μετονομασία* των μεταβλητών, β) την *ταυτοποίηση* των όρων, γ) την *εφαρμογή των αντικαταστάσεων* στους όρους και τους τύπους του προγράμματος-αντικείμενο και δ) τη *σύνθεση αντικαταστάσεων*.

2.3.2 Αναπαράσταση Προγράμματος-Αντικείμενου σε Μη-Βασικούς Όρους

Οι μεταβλητές της γλώσσας αντικείμενο, στην παράσταση σε μη-βασικούς όρους (non-ground representation), παριστάνονται από μεταβλητές της μετα-γλώσσας. Επιπλέον, οι προτάσεις της γλώσσας-αντικείμενο, όπως Η ← L1,…,Ln όπου L1,…,Ln στοιχειώδεις τύποι και n≥0, παριστάνονται σαν όροι της μετα-γλώσσας ως εξής:

H if L1 and…and Ln εάν n≥1
H if true εάν n=0

Ένας στόχος ← L1,…,Ln όπου n≥0, της γλώσσας αντικείμενο παριστάνεται στη μετα-γλώσσα από έναν όρο ως εξής:

← L1 and…and Ln εάν n≥1
← true εάν n=0

Το παρακάτω παράδειγμα δείχνει ένα μετα-πρόγραμμα solve/1 και ένα πρόγραμμα αντικείμενο το clause1/1 σε μη-βασική μορφή . Το μετα-πρόγραμμα είναι ένας μεταφραστής ,ο οποίος επεξεργάζεται στόχους για το πρόγραμμα αντικείμενο. Για παράδειγμα , "?- solve(p(b,X)). ".

Παράδειγμα

```
:-op(1200, xfx, [if]).
:-op(1000, xfy, [and]).
:-op(900, fy, [not]).
```

Μετα-πρόγραμμα

```
solve(true).
solve( (X and Y)) :- solve(X),
                     solve(Y).
solve( (not X)) :- \+ solve (X).
solve(Head) :- clause1( (Head if Body) ),
               solve(Body).
```

Πρόγραμμα Αντικείμενο

```
clause1( (p(X, Z) if q(X, Y) and p(Y, Z))).
clause1( (p(X, Y) if q(X, Y))).
clause1( (q(a, b) if true)).
clause1( (q(b, c) if true)).
```

Η *πρώτη πρόταση* του μετα-προγράμματος δηλώνει ότι ο άδειος στόχος ο οποίος παριστάνεται από το άτομο true λύνεται. Η *δεύτερη πρόταση* δηλώνει ότι για να λυθεί η σύζευξη των στόχων (A and B), πρώτα λύσε το στόχο A και μετά λύσε το στόχο B. Η *τρίτη πρόταση* δηλώνει ότι για να λυθεί η άρνηση ενός στόχου (not A) πρώτα λύσε το στόχο A και μετά αντίστρεψε την αλήθεια του. Η *τέταρτη πρόταση* δηλώνει ότι για να λυθεί ένας στόχος γίνεται επιλογή μιας πρότασης από το πρόγραμμα η κεφαλή της οποίας ταυτοποιείται με το στόχο, στη συνέχεια αναδρομικά, λύνει το σώμα της πρότασης.

Το πρόγραμμα είναι απλό και τρέχει σωστά, το πρόβλημα είναι ότι οι μεταβλητές του solve/1 και του clause1/1 παίρνουν τιμές από διαφορετικά πεδία. Οι τιμές των μεταβλητών του solve/1 είναι τύποι της γλώσσας-αντικείμενο π.χ. "p(X,Z)" if "q(X,Y) and p(Y,Z)", ενώ οι τιμές των μεταβλητών της clause1/1 είναι οντότητες/αντικείμενα από την ερμηνεία του προγράμματος-αντικείμενο δηλαδή οι σταθερές α, b και c. Αυτό το μετα-πρόγραμμα δεν είναι μια θεωρία που ανήκει στην λογική πρώτης-τάξεως. Δηλαδή, αυτό το μετα-πρόγραμμα δεν έχει δηλωτική σημασιολογία. Το ίδιο ισχύει και για όλα τα όμοια μετα-προγράμματα. Αυτός ο μεταφραστής μπορεί να παρασταθεί δηλωτικά (στη λογική πρώτης-τάξεως) σε γλώσσες λογικού προγραμματισμού που διαθέτουν τύπους. Αυτές οι γλώσσες υποστηρίζουν πολλά είδη μεταβλητών, καθένα από τα οποία παίρνει τιμές από διαφορετικά πεδία.

Το σύστημα Ευκλείδης χρησιμοποιεί αναπαράσταση σε μη βασική μορφή.

2.4 Μαθηματική Επαγωγή

Θεωρούμε ότι το $P(n)$ είναι μια *πρόταση* που ορίζεται στο σύνολο των φυσικών αριθμών ή μη αρνητικών, το οποίο συμβολίζεται με N. Με άλλα λόγια, *το P* είναι μια πρόταση *P:* $N \rightarrow$ {True,False}, όπου *το N* αντιπροσωπεύει το σύνολο των φυσικών αριθμών. Άτυπα αυτό σημαίνει ότι το $P(n)$, είναι κάποια πρόταση, της οποίας η αλήθεια εξαρτάται από την τιμή του ακέραιου αριθμού n. ***Η μαθηματική επαγωγή*** *(mathematical induction)* είναι μια τεχνική απόδειξης, που μπορεί να χρησιμοποιηθεί ώστε να αποδείξει προτάσεις της μορφής $\forall n \geq 0$: $P(n)$ (για όλους τους φυσικούς αριθμούς n, η πρόταση $P(n)$ είναι αληθής). Γενικότερα με τη μαθηματική επαγωγή επιδιώκουμε να αποδείξουμε ότι $\forall n \geq n_0 : P(n)$ είναι αληθής, όπου n και n_0 είναι φυσικοί αριθμοί. Αυτή η γενίκευση ισχύει και για τους ακέραιους αριθμούς.

Η μαθηματική επαγωγή εξηγείται μερικές φορές, αν κατ' αναλογία αναλύσουμε τον τρόπο λειτουργίας ενός ντόμινο. Εξετάζουμε ένα άπειρο σύνολο ντόμινο, που παρατάσσονται και είναι έτοιμα να πέσουν. Θεωρούμε ότι $P(n)$ είναι η δήλωση «το *n-στο* ντόμινο πέφτει». Αρχικά αποδεικνύουμε την πρόταση $P(1)$, η οποία κατ' αναλογία είναι «το πρώτο ντόμινο πέφτει». Κατόπιν επιδιώκουμε την απόδειξη της πρότασης $\forall n \geq 1: (P(n) \rightarrow P(n+1))$, το οποίο μεταφράζεται ως εξής: «εάν θεωρήσουμε ότι κάποιο ντόμινο πέφτει, κατά συνέπεια και το επόμενο ντόμινο πρέπει επίσης να πέσει». Όταν αυτό πραγματοποιηθεί μπορούμε να συμπεράνουμε ότι $\forall n \geq 1 : P(n)$, «όλα τα ντόμινο πέφτουν». Μια τέτοια απόδειξη αποτελείται από δύο βήματα.

ΑΡΧΗ ΤΗΣ ΑΠΛΗΣ ΜΑΘΗΜΑΤΙΚΗΣ ΕΠΑΓΩΓΗΣ

I. Βήμα βάσεως (Base step)**:** Απόδειξη $P(0)$.

Δηλαδή, αποδεικνύουμε ευθέως ότι η πρόταση $P(0)$ είναι αληθής.

II. Βήμα επαγωγής (Induction step)**:** Απόδειξη $\forall n \geq 0$: $(P(n) \rightarrow P(n+1))$

Επιλέγουμε έναν αυθαίρετο $n \geq 0$ και υποθέτουμε ότι για αυτό το n, η πρόταση $P(n)$ είναι αληθής. Κατόπιν αποδεικνύουμε, ως συνέπεια της παραπάνω υπόθεσης, ότι και η πρόταση $P(n+1)$ είναι αληθής. Η δήλωση $P(n)$ καλείται συχνά *υπόθεση επαγωγής*, δεδομένου ότι την λαμβάνουμε σαν υπόθεση στο βήμα επαγωγής.

Όταν το βήμα βάσεως (**I**) και το βήμα επαγωγής (**II**) έχουν αποδειχθεί πλήρως, έχουμε αποδείξει ότι η πρόταση $P(n)$ είναι αληθής για όλους τους φυσικούς αριθμούς $n \geq 0$.

<u>Παράδειγμα:</u>

Αποδείξτε ότι για όλα τα x,y,z∈N ισχύει η εξής πρόταση *P*:

***P*: x + (y + z) = (x + y) + z**

Απόδειξη:

Αρχίζουμε να εφαρμόζουμε την μέθοδο της μαθηματικής επαγωγής για x =0.

I. Βήμα βάσης

$$0 + (y + z) = (0 + y) + z$$
$$\Leftrightarrow \text{(αξίωμα πρόσθεσης: } 0 + x = x)$$
$$\mathbf{y + z} = (0 + y) + z$$
$$\Leftrightarrow \text{(αξίωμα πρόσθεσης: } 0 + x = x)$$
$$y + z = \mathbf{y + z}$$
$$\Leftrightarrow \text{true}$$

δείχνουμε ότι για x = 0 η πρόταση *P* είναι αληθής.

II. Βήμα επαγωγής

Θεωρούμε x ≥0 και υποθέτουμε ότι *P(x)* είναι αληθής. Προσπαθούμε ν' αποδείξουμε ότι η πρόταση *P* είναι αληθής και για x+1.

$$(x+1) + (y + z) = ((x+1) + y) + z$$
$$\Leftrightarrow \text{(αφαίρεση παρενθέσεων: } (x + y) = x + y)$$
$$\mathbf{x+1} + (y + z) = ((x+1) + y) + z$$
$$\Leftrightarrow \text{(αντιμεταθετική ιδιότητα: } x + y = y + x)$$
$$x + \mathbf{(y + z) +1} = ((x+1) + y) + z$$
$$\Leftrightarrow \text{(αφαίρεση παρενθέσεων: } (x + y) = x + y)$$
$$x + (y + z) +1 = \mathbf{(x +1 + y)} + z$$
$$\Leftrightarrow \text{(αντιμεταθετική ιδιότητα: } x + y = y + x)$$
$$x + (y + z) +1 = \mathbf{((x + y) +1)} + z$$
$$\Leftrightarrow \text{(αφαίρεση παρενθέσεων: } (x + y) = x + y)$$
$$x + (y + z) +1 = \mathbf{(x + y) +1} + z$$
$$\Leftrightarrow \text{(αντιμεταθετική ιδιότητα: } x + y = y + x)$$
$$x + (y + z) +1 = (x + y) + \mathbf{z +1}$$
$$\Leftrightarrow \text{(υπόθεση επαγωγής: } x + (y + z) = (x + y) + z \text{)}$$
$$\mathbf{(x + y) + z +1} = (x + y) + z +1$$
$$\Leftrightarrow \text{true}$$

Αποδεικνύοντας την ισχύ της πρότασης *P(x+1)*, προκύπτει από την αρχή της μαθηματικής επαγωγής, ότι η πρόταση *P(x)* ισχύει για όλα τα $x \in \mathrm{N}$.

ΓΕΝΙΚΕΥΜΕΝΗ ΑΡΧΗ ΤΗΣ ΑΠΛΗΣ ΜΑΘΗΜΑΤΙΚΗΣ ΕΠΑΓΩΓΗΣ

I. Βήμα βάσεως (Base step)**:** Απόδειξη $P(n_0)$.

Δηλαδή, αποδεικνύουμε ευθέως ότι η πρόταση $P(n_0)$ είναι αληθής.

II. Βήμα επαγωγής (Induction step)**:** Απόδειξη $\forall n \geq n_0 : (P(n) \rightarrow P(n+1))$

Επιλέγουμε έναν αυθαίρετο $n \geq n_0$ και υποθέτουμε ότι για αυτό το n, η πρόταση $P(n)$ είναι αληθής. Κατόπιν αποδεικνύουμε, ως συνέπεια της παραπάνω υπόθεσης, ότι και η πρόταση $P(n+1)$ είναι αληθής. Όταν το βήμα βάσεως (**I**) και το βήμα επαγωγής (**II**) έχουν αποδειχθεί πλήρως, έχουμε αποδείξει ότι η πρόταση $P(n)$ είναι αληθής για όλους τους φυσικούς αριθμούς $n \geq n_0$.

<u>Παράδειγμα:</u>

Αποδείξτε ότι για όλα τα n ≥1 ισχύει η εξής πρόταση *P*:

$$P\mathbf{: 1^2 + 2^2 + \ldots + n^2 = n\,(n+1)\,(2n+1) / 6}$$

Απόδειξη:

Αρχίζουμε να εφαρμόζουμε την μέθοδο της μαθηματικής επαγωγής για n =1.

I. Βήμα βάσης

$$1^2 = 1(1+1)(2+1) / 6$$
$$\Leftrightarrow \text{(αλγεβρικές πράξεις: πρόσθεση)}$$
$$1 = \mathbf{2*3} / 6$$
$$\Leftrightarrow \text{(αλγεβρικές πράξεις: πολλαπλασιασμός)}$$
$$1 = \mathbf{6} / 6$$
$$\Leftrightarrow \text{true}$$

δείχνουμε ότι για n = 1 η πρόταση *P* είναι αληθής.

II. Βήμα επαγωγής

Θεωρούμε n ≥1 και υποθέτουμε ότι *P(n)* είναι αληθής. Προσπαθούμε ν' αποδείξουμε ότι η πρόταση *P* είναι αληθής και για n+1.

$1^2 + 2^2 + \ldots + n^2 + (n+1)^2 = n(n+1)(2n+1) / 6 + (n+1)^2$

$\Leftrightarrow$ (αλγεβρικές πράξεις: πρόσθεση κλασμάτων)

$1^2 + 2^2 + \ldots + n^2 + (n+1)^2 = [n(n+1)(2n+1) + \mathbf{6(n+1)^2}] / 6$

$\Leftrightarrow$ (αλγεβρικές πράξεις: εξαγωγή κοινού παράγοντα)

$1^2 + 2^2 + \ldots + n^2 + (n+1)^2 = \mathbf{(n+1)}\,[n(2n+1) + 6(n+1)] / 6$

$\Leftrightarrow$ (αλγεβρικές πράξεις: πολλαπλασιασμός)

$1^2 + 2^2 + \ldots + n^2 + (n+1)^2 = (n+1)\,\mathbf{(2n^2 + n + 6n + 6)} / 6$

$\Leftrightarrow$ (αλγεβρικές πράξεις: πρόσθεση)

$1^2 + 2^2 + \ldots + n^2 + (n+1)^2 = (n+1)\,\mathbf{(2n^2 + 7n + 6)} / 6$

$\Leftrightarrow$ (αλγεβρικές πράξεις: παραγοντοποίηση)

$1^2 + 2^2 + \ldots + n^2 + (n+1)^2 = (n+1)\,\mathbf{(n+2)(2n+3)} / 6$

$\Leftrightarrow$ (αλγεβρικές πράξεις: ανάλυση σε συστατικά μέρη)

$1^2 + 2^2 + \ldots + n^2 + (n+1)^2 = (n+1)\,\mathbf{[(n+1)+1]\,[(2(n+1)+1)]} / 6$

$\Leftrightarrow$ true

Αποδεικνύοντας την ισχύ της πρότασης *P(n+1)*, προκύπτει από την αρχή της μαθηματικής επαγωγής, ότι η πρόταση *P(n)* ισχύει για όλα τα n≥1.

ΑΡΧΗ ΤΗΣ ΙΣΧΥΡΗΣ ΜΑΘΗΜΑΤΙΚΗΣ ΕΠΑΓΩΓΗΣ

I. Βήμα βάσεως (Base step)**:** Απόδειξη $P(n_0)$.

Δηλαδή, αποδεικνύουμε ευθέως ότι η πρόταση $P(n_0)$ είναι αληθής.

II. Βήμα επαγωγής (Induction step)**:** Απόδειξη $\forall n > n_0 : ((\forall k < n : P(k)) \rightarrow P(n))$

Θεωρούμε ένα $n \geq n_0$, και υποθέτουμε ότι για όλα *τα k,* που βρίσκονται στη σειρά $n_0 \leq k \leq n$, το $P(k)$ είναι αληθές. Κατόπιν αποδεικνύουμε ως συνέπεια αυτής της ισχύος ότι και το $P(n)$ ισχύει. Σε αυτήν την περίπτωση η *υπόθεση επαγωγής* είναι $\forall k < n : P(k)$.

Πρέπει βέβαια πάντα να δηλώνουμε την υπόθεση στο βήμα επαγωγής (δηλ. ποια είναι η υπόθεση επαγωγής), καθώς και να επισημαίνουμε το σημείο της απόδειξης, στην οποία η υπόθεση επαγωγής χρησιμοποιείται.

Παράδειγμα:

Αποδείξτε ότι για όλα τα n ≥0 ισχύει η εξής πρόταση *P*:

***P*: $f_n < 2^n$, όπου $f_n = f_{n-1} + f_{n-2}$** *(Fibonacci numbers)*

Απόδειξη:

Αρχίζουμε να εφαρμόζουμε την μέθοδο της μαθηματικής επαγωγής για n = 2 με δεδομένα ότι $f_0 = 0$ και $f_1 = 1$, οπότε και προφανώς ισχύει η πρόταση *P*.

I.Βήμα βάσης

$f_2 = f_{2-1} + f_{2-2}$

$\Leftrightarrow$ (αριθμητικές πράξεις)

$f_2 = \mathbf{f_1 + f_0}$

$\Leftrightarrow$ (αντικατάσταση τιμών)

$f_2 = \mathbf{1+0}$

$\Leftrightarrow$ (αριθμητικές πράξεις)

$f_2 = \mathbf{1} < 2^2$

$\Leftrightarrow$ true

δείχνουμε ότι για n = 2 η πρόταση *P* είναι αληθής.

II.Βήμα επαγωγής

Θεωρούμε $1 \leq k < n$ και υποθέτουμε ότι *P(k)* είναι αληθής, όπου *P(k)*: $f_k < 2^k$. Προσπαθούμε ν' αποδείξουμε ότι η πρόταση *P* είναι αληθής και για n.

$$f_n = f_{n-1} + f_{n-2}$$
$$\Leftrightarrow \text{(υπόθεση επαγωγής)}$$
$$f_n < \mathbf{2^{n-1} + 2^{n-2}}$$
$$\Leftrightarrow \text{(παραγοντοποίηση)}$$
$$2^{n-1} + 2^{n-2} = \mathbf{2^{n-2}(2 + 1)}$$
$$\Leftrightarrow \text{(αριθμητικές πράξεις)}$$
$$2^{n-1} + 2^{n-2} = 2^{n-2}\mathbf{(3)}$$
$$\Leftrightarrow \text{(αλγεβρικές πράξεις)}$$
$$f_n < \mathbf{2^{n-2}(4)}$$
$$\Leftrightarrow \text{(πράξεις δεικτών)}$$
$$2^{n-2}(4) = \mathbf{2^n}$$
$$\Leftrightarrow \text{true}$$

Αποδεικνύοντας την ισχύ της πρότασης *P(n)*, προκύπτει από την αρχή της μαθηματικής επαγωγής, ότι η πρόταση *P(n)* ισχύει για όλα τα $n \geq 0$.

2.5 Μηχανική Υποστήριξη στην Απόδειξη Θεωρημάτων

Ο σκοπός αυτού του κεφαλαίου είναι να συζητήσει τη μηχανική υποστήριξη, την οποία ένα υπολογιστικό σύστημα μπορεί να προσφέρει για την απόδειξη θεωρημάτων γενικώς. Στη συνέχεια γίνεται εξειδίκευση για τη μηχανική υποστήριξη που μπορεί να δοθεί για την απόδειξη θεωρημάτων με μαθηματική επαγωγή.

Σημείωση: Η έννοια της μηχανικής υποστήριξης αναφέρεται σε υπολογιστικές μηχανές (υπολογιστές).

2.5.1 Βασικοί Ορισμοί

Σ' αυτό το τμήμα ορίζονται κάποιες βασικές έννοιες, οι οποίες χρησιμοποιούνται στα επόμενα κεφάλαια.

Ορισμός 2.18

Πρόταση (proposition) είναι ένας ισχυρισμός, ο οποίος είναι είτε αληθής ή ψευδής, αλλά όχι και τα δυο. Εάν μια πρόταση είναι αληθής τότε λέμε ότι η τιμή αληθείας της πρότασης είναι αληθής (true), σ' αντίθετη περίπτωση η τιμή αληθείας της είναι ψευδής (false).

Ορισμός 2.19

Αξίωμα (axiom) είναι πρόταση την οποία δεχόμαστε σαν αληθή χωρίς απόδειξη. Μία μαθηματική δομή ή ένα μαθηματικό μοντέλο ενός φαινόμενου ορίζεται από ένα σύνολο αξιωμάτων. Εξ' ορισμού, ένα αξίωμα είναι μια αληθής πρόταση για τις ιδιότητες μιας μαθηματικής δομής (μαθηματικό μοντέλο ενός φαινόμενου).

Ορισμός 2.20

Κανόνες εξαγωγής συμπερασμάτων (rules of inference) ονομάζονται οι συλλογιστικές αρχές, που πρέπει να χρησιμοποιήσουμε για να αποδείξουμε ότι μια πρόταση εξάγεται από κάποια άλλη και τις οποίες αποδεχόμαστε ως έγκυρες.

Ορισμός 2.21

Θεώρημα (theorem) είναι μία αληθής πρόταση, η οποία εξάγεται από την αλήθεια των αξιωμάτων. Η εξαγωγή των θεωρημάτων γίνεται με εφαρμογή κανόνων εξαγωγής συμπερασμάτων.

Ορισμός 2.22

Απόδειξη (proof) ενός θεωρήματος είναι μια ακολουθία συλλογισμών, η οποία αποδεικνύει ότι το θεώρημα είναι αληθές στη συγκεκριμένη μαθηματική δομή.

Μια απόδειξη (proof) σ' ένα τυπικό σύστημα είναι μια πεπερασμένη ακολουθία καλά-σχηματισμένων τύπων (προτάσεων) στην τυπική γλώσσα, κάθε μια από τις οποίες είναι είτε ένα αξίωμα του συστήματος, ή ένα θεώρημα το οποίο παράχθηκε από ένα ή περισσότερους καλά-σχηματισμένους τύπους με εφαρμογή των κανόνων εξαγωγής συμπερασμάτων του συστήματος.

Ορισμός 2.23

Θεωρία (theory) σε μια μαθηματική δομή είναι το σύνολο όλων των αξιωμάτων μαζί με όλα τα θεωρήματα που μπορούν να παραχθούν από τα αξιώματα.

Διαισθητικά μπορούμε να φανταστούμε μια θεωρία σαν το σύνολο όλων των αληθών προτάσεων σχετικά με (που αφορούν) ένα φαινόμενο. Στην πράξη οι περισσότερες θεωρίες περιέχουν πάρα πολλά γεγονότα για να καταγραφούν, γι' αυτό ένα υποσύνολο αυτών καταγράφεται με σαφήνεια (τα αξιώματα) και κάποιοι κανόνες εξαγωγής συμπερασμάτων παράγουν τα υπόλοιπα θεωρήματα.

Ορισμός 2.24

Αποτέλεσμα το οποίο εισάγεται σε αποδείξεις χωρίς να ξαναπαραχθεί η απόδειξή του ονομάζεται ***λήμμα (lemma)***.

2.5.2 Αυτόματη και Διαλογική Απόδειξη Θεωρημάτων

Σ' αυτό το τμήμα θα περιγράψουμε τι είναι η αυτόματη απόδειξη θεωρημάτων και τι είναι η διαλογική απόδειξη θεωρημάτων.

Η αυτόματη απόδειξη θεωρημάτων αναφέρεται στη μελέτη και στην ανάπτυξη προγραμμάτων τα οποία διαθέτουν λογική σκέψη (σκέπτονται λογικά). Η έκφραση «λογική σκέψη» αναφέρεται σε διαδικασίες οι οποίες συμπεραίνουν (εξάγουν λογικά) νέες προτάσεις (ή τύπους) από υπάρχουσες προτάσεις (ή τύπους), έτσι ώστε η νέα πρόταση (ή τύπος) να είναι πάντα αληθής σε κάθε ερμηνεία όπου οι παλιές προτάσεις είναι αληθείς.

Ένα απλό παράδειγμα λογικής σκέψης είναι ο modus ponens, ο οποίος από τις προτάσεις (τύπους) p→q και p εξάγει λογικά την πρόταση q. Ένα πιο συγκεκριμένο παράδειγμα εφαρμογής του modus ponens είναι το εξής:

Μορφή Πρότασης	Πρόταση
p → q	Η Μαρία σπουδάζει Τεχνολόγος Ηλεκτρολόγος → Η Μαρία είναι σπουδάστρια ΤΕΙ
p	Η Μαρία σπουδάζει Τεχνολόγος Ηλεκτρολόγος
Εξαγόμενη πρόταση q	Η Μαρία είναι σπουδάστρια ΤΕΙ

Ένα δεύτερο παράδειγμα είναι το εξής: Από το τύπο $\forall x$/ Τ f(x) όπου f(x) είναι ένας τύπος (πρόταση) της λογικής πρώτης τάξης, μπορούμε να συμπεράνουμε ότι f(α) είναι αληθές όπου α είναι ένας όρος της γλώσσας την οποία χρησιμοποιούμε. Ένα πιο συγκεκριμένο παράδειγμα είναι το έξης: Από την πρόταση «∀Χ/άνθρωπος θνητός(Χ)» μπορούμε να συμπεράνουμε ότι εάν Χ είναι ο Σωκράτης, τότε η πρόταση «θνητός(Σωκράτης)» είναι επίσης αληθής.

Πρόταση Υπόθεση	Πρόταση Συμπέρασμα ή Πρόταση Θεώρημα
∀Χ/άνθρωπος θνητός(Χ)	θνητός(Σωκράτης)
∀Χ/ Τεχνολόγος_Ηλεκτρολόγος έχει_σπουδάσει_ΤΕΙ(Χ)	Μαρία ∈ Τεχνολόγος_Ηλεκτρολόγος έχει_σπουδάσει_ΤΕΙ(Μαρία)
ή συμβολικά	
$\forall x$/ T f(x)	$\alpha \in$ T f(α)

Είναι εύκολο να διαπιστώσουμε ότι η «πρόταση συμπέρασμα» είναι αληθής σε κάθε ερμηνεία όπου η «πρόταση υπόθεση» είναι αληθής. Τα αυτόματα συστήματα απόδειξης θεωρημάτων αποδεικνύουν αυτόματα, χωρίς την παρέμβαση ανθρώπου νέες προτάσεις συμπεράσματα, χρησιμοποιώντας την βάση γνώσεων (ΒΓ) που διέπουν και κάποιους κανόνες εξαγωγής συμπερασμάτων. Η ΒΓ αυτών των συστημάτων περιέχει αξιώματα και πιθανόν άλλα θεωρήματα τα οποία έχουν παραχθεί με προηγούμενους συλλογισμούς.

Όμως δεν είναι πάντοτε εφικτή η αυτόματη απόδειξη θεωρημάτων. Σε πολύπλοκα προβλήματα χρειάζεται η ανθρώπινη παρέμβαση. Μια σημαντική παρέμβαση του ανθρώπου-χρήστη είναι στο να επιλέγει α) τον συμπερασματικό κανόνα που πρέπει να εφαρμοστεί από το σύστημα και β) που, σε ποιες προτάσεις, θα εφαρμοστεί ο επιλεχθέντας κανόνας. Σ' αυτή την περίπτωση έχουμε τα ***διαλογικά συστήματα απόδειξης θεωρημάτων***. Ο βαθμός παρέμβασης στην διαδικασία απόδειξης είναι διαφορετικός από σύστημα σε σύστημα. Τα διαλογικά συστήματα στα οποία ο χρήστης αναπτύσσει την απόδειξη και το σύστημα εξασφαλίζει σε κάποιο βαθμό «σημασιολογική ορθότητα» ονομάζονται και ***proof editors*** (***συντάκτες απόδειξης***). Διαλογικά συστήματα απόδειξης θεωρημάτων χρησιμοποιούνται σε προβλήματα που απαιτούν πολύπλοκη ή δύσκολη απόδειξη. Τέτοια συστήματα έχουν κατασκευαστεί στην περιοχή των μαθηματικών για απόδειξη δύσκολων μαθηματικών θεωρημάτων. Στην περιοχή της επιστήμης υπολογιστών για επικύρωση λογισμικού (software) και υλικό (hardware).

Διαχείριση της απόδειξης σε διαλογικά συστήματα

Για να γίνουν τα διαλογικά συστήματα απόδειξης θεωρημάτων ελκυστικά στο χρήστη, θα πρέπει να έχουν πολύ καλά σχεδιασμένη διεπικοινωνία, ώστε να είναι αποδεκτή από αρχάριους χρήστες. Για παράδειγμα, κάποιοι συντάκτες απόδειξης παρέχουν στον χρήστη πληροφορίες για τις συνέπειες από τις αλλαγές στην απόδειξη. Η κατασκευή μιας απόδειξης είναι μια διαδικασία δοκιμής-λάθους (trial-error). Οι προσπάθειες απόδειξης μπορεί να γίνουν πολύ μεγάλες, γι' αυτό απαιτείται καλή διεπικοινωνία, ώστε να διευκολύνει την διαχείριση των αποδείξεων. Άλλα ζητήματα τα οποία αφορούν την χρησιμότητα των διαλογικών συστημάτων απόδειξης είναι η ταχύτητα αποκρίσεως, εάν διαθέτουν υποσύστημα βοήθειας και εάν η όλη διαδικασία απόδειξης είναι κατανοητή ειδικά σε αρχάριους χρήστες.

Η βασική άποψη των συστημάτων διαλογικής απόδειξης είναι ότι ο χρήστης μπορεί με χρήσιμους τρόπους να κατευθύνει την έρευνα της απόδειξης, αλλά για να γίνει αυτό θα πρέπει να γνωρίζει τα εξής:

1. Το περιβάλλον της απόδειξης, δηλαδή τη θεωρία.
2. Τι έχει ήδη γίνει από την απόδειξη.
3. Τι μένει να γίνει από την απόδειξη.
4. Τα όπλα, που διαθέτει για την απόδειξη (λήμματα κ.τ.λ.).

2.5.3 Μηχανική Υποστήριξη Μαθηματικής Επαγωγής

Οι επαγωγικοί ορισμοί δεν είναι μόνο μια μέθοδος για να ορίζονται μη πεπερασμένα σύνολα αλλά επιπλέον είναι η βάση ισχυρών τεχνικών για απόδειξη θεωρημάτων. Εάν ένα σύνολο είναι πεπερασμένο, μια πρόταση της μορφής $\forall x\ P(x)$ μπορεί να αποδειχθεί με εξέταση όλων των περιπτώσεων. Αλλά για μη πεπερασμένα σύνολα κάποιος άλλος μηχανισμός πρέπει να χρησιμοποιηθεί. Αποδείξεις με επαγωγή είναι αποδείξεις σε προτάσεις με καθολική δέσμευση, όπου το πεδίο αναφοράς είναι ένα σύνολο το οποίο ορίζεται επαγωγικά.

Υποθέτουμε ότι θέλουμε ν' αποδείξουμε ότι όλα τα στοιχεία ενός επαγωγικά οριζόμενου συνόλου S έχουν την ιδιότητα P. Δηλαδή θέλουμε ν' αποδείξουμε ότι η πρόταση $\forall x\ P(x)$ είναι αληθής στο πεδίο αναφοράς S. Μια απόδειξη με επαγωγή συνίσταται από δύο μέρη, τα οποία αντιστοιχούν στην πρόταση για τη ***βάση*** ορισμού του συνόλου S και στην πρόταση που αντιστοιχεί στην ***επαγωγή*** ορισμού του συνόλου S.

1. Το βήμα της ***βάσης*** αποδεικνύει ότι η πρόταση $P(x)$ είναι αληθής για κάθε στοιχείο $x \in S$, το οποίο ορίζεται στη βασική πρόταση ορισμού του S.
2. Το βήμα της ***επαγωγής*** αποδεικνύει ότι κάθε στοιχείο, το οποίο κατασκευάζεται χρησιμοποιώντας την επαγωγική πρόταση ορισμού του S, έχει την ιδιότητα P, εάν όλα τα στοιχεία τα οποία χρησιμοποιούνται για την κατασκευή του έχουν αυτή την ιδιότητα.

Ας θεωρήσουμε το σύνολο S των καλοσχηματισμένων παρενθέσεων (ισορροπημένη συμβολοσειρά παρενθέσεων). Ο επαγωγικός ορισμός αυτού του συνόλου S είναι ο εξής:

1. Βάση: ( ) είναι ένα στοιχείο του S.
2. Επαγωγή : Εάν *x* και y είναι στοιχεία του S τότε:
   a. (*x*) είναι ένα στοιχείο του S.
   b. *x*y είναι ένα στοιχείο του S.

Αυτό το επαγωγικό σύνολο S είναι το σύνολο όλων των ακολουθιών παρενθέσεων, που μπορεί να εμφανιστούν σε μια μαθηματική έκφραση, όπως ( ),(( )),( )( ),(( ))( ),((( ))( )),κ.τ.λ.

Οι πιο γνωστές αποδείξεις με επαγωγή είναι αυτές που σχετίζονται με τους φυσικούς αριθμούς, δηλαδή με το σύνολο N. Ο επαγωγικός ορισμός των φυσικών αριθμών είναι ο εξής:

1. Βάση: $0 \in N$.
2. Επαγωγή : Εάν $n \in N$ τότε το $S(n) \in N$, όπου S είναι η συνάρτηση επόμενος (successor).

Η συνάρτηση επόμενος (successor) ορίζεται ως εξής:

$\forall\ n \in N$ ο $S(n)$ είναι ο επόμενός του, δηλαδή ο $n+1$.

Η ανθρώπινη παρέμβαση είναι αναγκαία σε συστήματα τα οποία αποδεικνύουν θεωρήματα με επαγωγή. Κάποιες τετριμμένες δραστηριότητες γίνονται αυτόματα έπειδη η υλοποίησή τους είναι εύκολη. Παραδείγματα τετριμμένων εργασιών είναι οι εξής [Bundy 2001]:

a. Διατήρηση της κατάστασης της απόδειξης.
b. Ταυτοποίηση εκφράσεων.
c. Απλοποίηση εκφράσεων.
d. Η εξαντλητική εφαρμογή κανόνων που μεταφράζουν εκφράσεις σε ισοδύναμες μορφές.

Οι πιο δύσκολές εργασίες εκτελούνται από τους χρήστες μέσω ενός διαλογικού περιβάλλοντος. Παραδείγματα δύσκολων εργασιών είναι οι εξής:

a. Η επιλογή του κανόνα επαγωγής.
b. Η εφαρμογή ενός λήμματος.
c. Η γενίκευση μιας εικασίας.

Αυτές οι εργασίες απαιτούν διορατικότητα για τη δομή της απόδειξης.

Ένα δημοφιλές πλαίσιο για ημιαυτόματα συστήματα απόδειξης θεωρημάτων με επαγωγή, είναι η χρήση ***τακτικών (tactics)***. Μια ***τακτική (tactic)*** είναι ένα πρόγραμμα το οποίο κατευθύνει την έρευνα της απόδειξης. Αυτό το πρόγραμμα μπορεί να εφαρμόσει ένα συμπερασματικό κανόνα ή να συνθέσει δύο ή περισσότερες εφαρμογές τακτικών. Τακτικές κατασκευάζονται για τις τετριμμένες εργασίες. Ο χρήστης μπορεί στη συνέχεια να κατευθύνει την έρευνα της απόδειξης, είτε καλώντας συγκεκριμένους συμπερασματικούς κανόνες, ή καλώντας τακτικές οι οποίες εφαρμόζουν περισσότερους του ενός συμπερασματικούς κανόνες. Έτσι οι τετριμμένες εργασίες γίνονται από το σύστημα. Ο χρήστης βλέπει την απόδειξη είτε σε υψηλό επίπεδο εφαρμογής των τακτικών ή σε χαμηλό επίπεδο εφαρμογής συγκεκριμένων συμπερασματικών κανόνων.

Για να μπορεί ο χρήστης να καθοδηγεί ένα ημιαυτόματο σύστημα επαγωγικής απόδειξης θεωρημάτων, θα πρέπει να του παρέχεται μια καλή διαλογική διεπικοινωνία. Αυτή η διεπικοινωνία πρέπει να παρέχει στο χρήστη τη μέγιστη δυνατή βοήθεια, όταν παίρνει δύσκολες αποφάσεις.

Ο σχεδιασμός της διεπικοινωνίας εξαρτάται από το χρήστη του συστήματος. Οι αρχάριοι χρήστες χρειάζονται να ορίσουν τις υποθέσεις/εικασίες, να βλέπουν την πορεία της απόδειξης και να καθοδηγούν την απόδειξη. Οι πεπειραμένοι χρήστες χρειάζονται να ορίζουν νέες θεωρίες, να βλέπουν μέσω βιβλιοθηκών τις υποθέσεις, τους ορισμούς, τα λήμματα κ.τ.λ. και να μετακινούνται από μια προσπάθεια απόδειξης σε μια άλλη προσπάθεια. Γραφικές διεπικοινωνίες, παραθυρικό περιβαλλον και δομημένος κειμενογράφος είναι οι πιο δημοφιλείς τρόποι διεπικοινωνίας.

Πολλές προσπάθειες έχουν γίνει για μηχανοποίηση της απόδειξης προτάσεων (θεωρημάτων) με επαγωγή [Boyer-Moore],[Bundy1988],[Bundy2001]. Το πλέον σημαντικό σημείο για την απόδειξη τέτοιων προτάσεων/ θεωρημάτων είναι πως θα μετασχηματίσει κάποιος το συμπέρασμα της πρότασης/ θεωρήματος, ώστε να μπορέσει να εφαρμόσει την υπόθεση της επαγωγής (induction hypothesis). Στο [Boyer-Moore] περιγράφονται τακτικές (tactics) οι οποίες κατευθύνουν επαγωγικές αποδείξεις αυτόματα. Το πρόγραμμά τους περιέχει μεγάλες ποσότητες ευρηματικής πληροφορίας, η οποία με επιτυχία κατευθύνει επαγωγικές αποδείξεις. Οι περιγραφές των ευρηματικών (heuristic) μεθόδων τους δεν είναι ευκρινείς γιατί επιτυγχάνουν και γιατί εφαρμόζονται με κάποια συγκεκριμένη σειρά.

Βελτιώσεις αυτών των τακτικών με βάση κάποια σχέδια απόδειξης παρουσιάζονται στο άρθρο [Bundy 1988]. Οι ευρηματικές τεχνικές των Boyer-Moore παριστάνονται σαν ένα σαφές σχέδιο απόδειξης. Ο στόχος του συστήματός τους ήταν να τυποποιήσουν τις ευρηματικές τεχνικές των Boyer-Moore, έτσι ώστε να μπορούν να προβλέψουν κάτω από ποιες συνθήκες επιτυγχάνουν ή αποτυγχάνουν. Επίσης ήθελαν να εξηγήσουν την δομή και τη σειρά εφαρμογής τους.

Σ' αυτή την πτυχιακή έχουμε αναπτύξει το σύστημα Ευκλείδης, το οποίο υποστηρίζει το χρήστη στην απόδειξη προτάσεων/ θεωρημάτων με επαγωγή. Τα επαγωγικά σύνολα είναι τα σύνολα των φυσικών αριθμών και υποσύνολα των ακεραίων αριθμών. Ο χρήστης μέσα από ένα φιλικό διαλογικό περιβάλλον αποφασίζει για την πράξη μετασχηματισμού της πρότασης/ του θεωρήματος της βάσης ή της επαγωγής και το σύστημα εκτελεί την πράξη.

Σημείωση: Τακτική (tactic) είναι μία συνάρτηση, η οποία αναλύει/ μετασχηματίζει μια πρόταση/ θεώρημα σε άλλη μορφή πιο κατάλληλη για απόδειξη της πρότασης/ θεωρήματος.

3 Αναπαραστάσεις

Σε αυτό το κεφάλαιο θα παρουσιάσουμε την σύνταξη των θεωρημάτων, αξιωμάτων, λημμάτων και νόμων του κατηγορηματικού λογισμού, τα οποία γίνονται αποδεκτά από το σύστημα *Ευκλείδης*.

Επιπλέον, θα μιλήσουμε για το πώς αναπαριστάνουμε στο σύστημά μας όλα τα στοιχεία που χρησιμοποιούνται σε μια απόδειξη δια της μαθηματικής επαγωγής. Δηλαδή θα μιλήσουμε για την αναπαράσταση των θεωρημάτων, αξιωμάτων, λημμάτων, των όρων του προτασιακού και κατηγορηματικού λογισμού, της ισότητας και των μετασχηματισμών που εφαρμόζονται. Σ' ένα ημι-αυτόματο σύστημα απόδειξης θεωρημάτων ο χρήστης χρειάζεται να γνωρίζει τα προηγούμενα βήματα της απόδειξής του, γι' αυτό κάθε βήμα της απόδειξης χρειάζεται να αναπαριστάνεται επίσης.

3.1 Σύνταξη Στοιχείων Συστήματος

3.1.1 Σύνταξη Θεωρημάτων, Αξιωμάτων & Λημμάτων

Ένα θεώρημα, ένα αξίωμα ή ένα λήμμα μπορεί να δίνεται με τη μορφή:

$$A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m,$$

όπου $A_1,...,A_n$, $B_1,..,B_m$ είναι άτομα, το ',' υποδηλώνει τη σύζευξη και το '$\rightarrow$' υποδηλώνει τη συνεπαγωγή.

Επιπλέον η ποσοτικοποίηση (quantification) όλων των μεταβλητών, εφόσον δε δηλώνεται διαφορετικά, είναι καθολική (universal) με εμβέλεια όλο το θεώρημα.

Παραδείγματα

1. *Θεώρημα:* $\forall$ X: N, Y: N (p(a,X)$\wedge$q(b,Y)$\rightarrow$r(a,X)$\wedge$s(Y))

   Ισοδύναμη αναπαράσταση: p(a,X), q(b,Y) $\rightarrow$ r(a,X), s(Y)

2. *Θεώρημα:* $\forall$ X: N, Y: N (even (X)$\wedge$even (Y)$\rightarrow$even(X+Y))

   Ισοδύναμη αναπαράσταση: even(X),even(Y) $\rightarrow$ even(plus(X, Y))

3. *Θεώρημα:* $\forall$ X: N, Y: N (p(a,X) $\wedge$ q(b,Y))

   Ισοδύναμη αναπαράσταση: true $\rightarrow$ (p(a,X) $\wedge$ q(b,Y))

4. *Θεώρημα:* true

   Ισοδύναμη αναπαράσταση: true $\rightarrow$ true

5. *Αξίωμα:* $\forall$ M: N (even(succ(succ(M)))$\rightarrow$ even(M))

   Ισοδύναμη αναπαράσταση: even(succ(succ(M)))$\rightarrow$ even(M)

6. *Αξίωμα:* (even(succ(0))$\rightarrow$ false)

   Ισοδύναμη αναπαράσταση: even(succ(0))$\rightarrow$ false)

7. *Λήμμα:* $\forall$ X: N, Y: N (even (X)$\wedge$even (Y)$\rightarrow$even(X+Y))

   Ισοδύναμη αναπαράσταση: even(X),even(Y) $\rightarrow$ even(plus(X,Y)

3.1.2 Σύνταξη Νόμων Προτασιακού και Κατηγορηματικού Λογισμού

Οι νόμοι του προτασιακού και του κατηγορηματικού λογισμού, τους οποίους χρησιμοποιούμε είναι λογικές ισοδυναμίες.

Θεωρούμε δυο τύπους του προτασιακού λογισμού φ και φ΄ ,που είναι ισοδύναμοι, γεγονός που συμβολίζεται $\varphi \Leftrightarrow \varphi'$. Έστω ότι ο τύπος ψ περιέχει τον τύπο φ, τότε έχουμε την εξής ισοδυναμία $\psi \Leftrightarrow \psi'\{\varphi|\varphi'\}$.

Δηλαδή ο τύπος ψ είναι λογικά ισοδύναμος, εάν κάθε εμφάνιση του τύπου φ στον τύπο ψ αντικατασταθεί με το τύπο φ΄. Σημειώνουμε ότι το σύμβολο ↔ δεν είναι ίδιο με το σύμβολο ⇔.Το σύμβολο ↔ είναι ένας λογικός σύνδεσμος του προτασιακού και του κατηγορηματικού λογισμού, ενώ το σύμβολο ⇔ είναι μετα-σύμβολο, το οποίο μας μιλά για τη σχέση που υφίσταται μεταξύ δύο προτάσεων του προτασιακού ή του κατηγορηματικού λογισμού, ότι δηλαδή είναι ισοδύναμες .

Ένας νόμος του Προτασιακού & Κατηγορηματικού Λογισμού μπορεί να δίνεται με τη μορφή:

$$(A_1,\ldots, A_n \rightarrow B_1,...,B_m) \Leftrightarrow \Phi,$$

όπου $A_1,..., A_n$ και $B_1,...,B_m$ είναι στοιχειώδεις ατομικοί τύποι, Φ είναι ένας τύπος ισοδύναμος του τύπου αριστερά της ⇔, το ',' υποδηλώνει τη σύζευξη, το '→' υποδηλώνει τη συνεπαγωγή και το '⇔' υποδηλώνει την ισοδυναμία των δύο τύπων.

<u>Παραδείγματα</u>

1. Ο νόμος του κατηγορηματικού λογισμού : $(\text{false} \rightarrow \varphi) \Leftrightarrow \text{true}$,
   αν εφαρμοστεί στον τύπο $\forall$ Y: N $(\text{false} \rightarrow p(Y))$
   παίρνουμε τον τύπο true.
   Αυτός ο μετασχηματισμός εκφράζεται ως εξής:
   $\forall$ Y: N $((\text{false} \rightarrow p(Y)) \Leftrightarrow \text{true})$.
2. Ο νόμος του κατηγορηματικού λογισμού : $\varphi \wedge \text{false} \Leftrightarrow \text{false}$,
   αν εφαρμοστεί στον τύπο $\forall$ Y: N $(\text{even}(Y) \wedge \text{false})$
   παίρνουμε τον τύπο false.
   Αυτός ο μετασχηματισμός εκφράζεται ως εξής:
   $\forall$ Y: N $(\text{even}(Y) \wedge \text{false} \Leftrightarrow \text{false})$.
3. Ο νόμος του κατηγορηματικού λογισμού : $\text{true} \wedge \varphi \Leftrightarrow \varphi$,
   αν εφαρμοστεί στον τύπο $\forall$ Y: N $(\text{true} \wedge \text{even}(Y))$
   παίρνουμε τον τύπο $\forall$ Y: N even(Y).
   Αυτός ο μετασχηματισμός εκφράζεται ως εξής:
   $\forall$ Y: N $(\text{true} \wedge \text{even}(Y) \Leftrightarrow \text{even}(Y))$.
4. Ο νόμος του κατηγορηματικού λογισμού : $\neg\neg\varphi \Leftrightarrow \varphi$,
   όπου το σύμβολο ¬ εκφράζει το not, αν εφαρμοστεί στον τύπο
   $\forall$ Y: N $(\neg\neg\, p(Y))$
   παίρνουμε τον τύπο $\forall$ Y: N p(Y).
   Αυτός ο μετασχηματισμός εκφράζεται ως εξής:
   $\forall$ Y: N $((\neg\neg\, p(Y)) \Leftrightarrow p(Y))$.

3.2 Αναπαράσταση Βασικών Στοιχείων Απόδειξης

3.2.1 Αναπαράσταση Θεωρημάτων

Ένα θεώρημα μπορεί να αναπαρασταθεί στη βάση γνώσεων σαν μία πρόταση γεγονός της Prolog της εξής μορφής:

theorem(TheoremId, LeftForm, RightForm).

Όπου

- 'TheoremId' είναι ένας μοναδικός ακέραιος αριθμός που παριστά την ταυτότητα του θεωρήματος.
- 'LeftForm' είναι μια λίστα με άτομα πριν (αριστερά από) την συνεπαγωγή ή την ισοδυναμία.
- 'RightForm' είναι μια λίστα με άτομα μετά (δεξιά από) την συνεπαγωγή ή την ισοδυναμία .

Οι λίστες των ατόμων 'LeftForm' και 'RightForm' μπορούν να είναι άδειες.

Παραδείγματα

1. Ο ατομικός τύπος:
   «theorem(1, [p(a,X), q(b,Y)], [r(a,X), s(Y)]).»
   παριστά το θεώρημα:
   $\forall X, \forall Y (p(a,X) \wedge q(b,Y) \rightarrow r(a,X) \wedge s(Y))$.
2. Ο ατομικός τύπος:
   « theorem(2, [even(X), even(Y)],[even(plus(X,Y))]).»
   παριστά το θεώρημα:
   $\forall X, \forall Y (even(X) \wedge even(Y) \rightarrow even(X + Y))$.
3. Ο ατομικός τύπος:
   « theorem(3, [true], [p(a,X), q(b,Y)]).»
   παριστά το θεώρημα:
   $\forall X, \forall Y (true \rightarrow p(a,X) \wedge q(b,Y))$.
4. Ο ατομικός τύπος:
   «theorem(4, [true], [true]).»
   παριστά το θεώρημα:
   $(true \rightarrow true)$.

3.2.2 Αναπαράσταση Αξιωμάτων, Λημμάτων, Επαγωγών και Νόμων του Προτασιακού και του Κατηγορηματικού Λογισμού.

Τα αξιώματα, λήμματα, οι νόμοι του προτασιακού και του κατηγορηματικού λογισμού και οι επαγωγές παριστάνονται στη βάση γνώσεων σαν προτάσεις γεγονότα της Prolog όπως τα θεωρήματα. Για καθένα από αυτά έχουμε την παρακάτω αναπαράσταση.

1. Τα αξιώματα παριστάνονται από την εξής πρόταση γεγονός :
   «axiom (AxiomId, LeftForm, RightForm).»
2. Τα λήμματα παριστάνονται από την εξής πρόταση γεγονός:
   «lemma (LemmaId, LeftForm, RightForm).»

3. Οι νόμοι του προτασιακού και κατηγορηματικού λογισμού παριστάνονται από την εξής πρόταση γεγονός:
   «fol_law (Fol_lawId, LeftForm, RightForm).»
4. Οι επαγωγές παριστάνονται από την εξής πρόταση γεγονός :
   «induction (InductionId,LeftForm,RightForm).»

Όπου:

- τα ορίσματα 'LeftForm', 'RightForm' έχουν την ίδια έννοια όπως την περίπτωση των θεωρημάτων.
- 'AxiomId' είναι η ταυτότητα του αξιώματος που εκφράζεται με ένα μοναδικό ακέραιο αριθμό.
- 'LemmaId' είναι η ταυτότητα του λήμματος που εκφράζεται με ένα μοναδικό ακέραιο αριθμό.
- 'Fol_lawId' είναι η ταυτότητα του νόμου του κατηγορηματικού ή προτασιακού λογισμού που εκφράζεται με ένα μοναδικό ακέραιο αριθμό.
- 'InductionId' είναι η ταυτότητα της επαγωγής που εκφράζεται με ένα μοναδικό ακέραιο αριθμό.

Παραδείγματα

1. *Axiom:* ∀ v: N (0+v = v)
   Αναπαράσταση σε βάση γνώσεων:
   axiom(1, [eq(plus(0, N), N)], []).

2. *Axiom:* ∀ v: N, m: N ((succ(v) +m) = succ(v+m))
   Αναπαράσταση σε βάση γνώσεων:
   axiom(2, [eq(plus(succ(N), M), succ(plus(N,M)))], []).

3. *Induction:* ∀ X, Y, Z: N, ((X+Y) +Z) → (X+(Y+X))
   Αναπαράσταση σε βάση γνώσεων:
   induction(1,[plus(plus(X,Y),Z)],[plus(X,plus(Y,Z))]).

4. *Fol_law:* ∀ Y: N, true ∧ even(Y) ↔even(Y)
   Αναπαράσταση σε βάση γνώσεων:
   fol_law(1, [[true,even(Y)],[]], [even(Y)]).

3.3 Αναπαράσταση Ισότητας

Η ισότητα αναπαριστάνεται σαν πρόταση γεγονός της Prolog, ως εξής:

eq(LeftPart, RightPart)

όπου το eq δηλώνει την ισότητα ενώ τα LeftPart και RightPart είναι αντίστοιχα τα ορίσματα αριστερά και δεξιά της ισότητας (=).

Παράδειγμα

Έχουμε το θεώρημα: ∀ X:N, Y:N, Z:N (X + (Y + Z) = (X + Y) + Z).
Η ισοδύναμη αναπαράσταση της παραπάνω πρότασης σε συναρτησιακή μορφή είναι η εξής: forall X:N, Y:N, Z:N eq(plus(X, plus(Y,Z)), plus(plus(X,Y),Z)).
Αυτό το θεώρημα παριστάνεται στη βάση γνώσεων σαν πρόταση γεγονός της Prolog ως εξής: theorem (3, [eq(plus(X, plus(Y,Z)), plus(plus(X,Y),Z))], []).

Είναι απαραίτητο να συμπεριλάβουμε επίσης μερικά αξιώματα, που ορίζουν την ισότητα (=). Η ακόλουθη *Θεωρία της Ισότητας* είναι ικανοποιητική για τους σκοπούς αυτής της εργασίας. Σ' αυτά τα αξιώματα, χρησιμοποιούμε το σύμβολο (≠) για τον ορισμό των *μη* ίσων ποσοτήτων.

1. $\forall$ x=x.
2. c≠d, για όλα τα ζεύγη c, d διακριτών σταθερών.
3. $\forall(f(x_{1,...,}x_n) \neq g(y_{1,...,}y_m))$ για όλα τα ζεύγη f, g διακριτών συναρτήσεων.
4. $\forall(f(x_{1,...,}x_n) \neq c)$ για κάθε σταθερά c και συνάρτηση f.
5. $\forall(t[x] \neq x)$ για κάθε όρο t[x] που περιέχει το x και διαφέρει από αυτό.
6. $\forall((x_1 \neq y_1)\vee\ldots\vee(x_n \neq y_n) \rightarrow f(x_{1,...,}x_n) \neq f(y_{1,...,}y_n))$ για κάθε συνάρτηση f.
7. $\forall((x_1 = y_1)\wedge\ldots \wedge(x_n = y_n) \rightarrow f(x_{1,...,}x_n) = f(y_{1,...,}y_n))$ για κάθε συνάρτηση f.
8. $\forall((x_1 = y_1)\wedge\ldots \wedge(x_n = y_n) \rightarrow (p(x_{1,...,}x_n) \rightarrow p(y_{1,...,}y_n)))$ για κάθε κατηγορηματικό σύμβολο p (including =).

3.4 Αναπαράσταση Μετασχηματισμών

Οι μετασχηματισμοί έχουν χωριστεί σε δύο κατηγορίες, στους μετασχηματισμούς assign και τους μετασχηματισμούς apply .

3.4.1 Μετασχηματισμοί Assign

Μετασχηματισμοί που καταχωρούν μια τιμή σε μια μεταβλητή στο θεώρημα που πρόκειται να μετασχηματιστεί. Η σύνταξη αυτών των μετασχηματισμών έχει ως εξής:

assign, V, Vorder,

όπου με τον μετασχηματισμό assign ,η τιμή (V) καταχωρείται στην μεταβλητή που έχει πρώτη εμφάνιση στη σειρά Vorder. Όλες οι εμφανίσεις αυτής της μεταβλητής ξεκινώντας από αριστερά προς τα δεξιά, αντικαθίστανται από την τιμή V.

Παράδειγμα

Θεωρούμε ότι έχουμε το θεώρημα :

theorem(1, [eq(plus(X,plus(Y,Z)), plus(plus(X,Y),Z))], []).

Αν ο μετασχηματισμός που θέλουμε να εφαρμόσουμε είναι **[assign, 1, 2]** ακολουθούμε τα παρακάτω βήματα:

- Παίρνουμε όλες τις μεταβλητές του θεωρήματος όπως τις βλέπουμε, δηλαδή X, Y, Z.
- Βλέπουμε από τον μετασχηματισμό [assign, 1, 2] ότι θέλουμε την δεύτερη στη σειρά μεταβλητή, δηλαδή στο παράδειγμά μας είναι η Y.
- Αντικαθιστούμε αυτή τη μεταβλητή με τον αριθμό 1, (για τον συγκεκριμένο μετασχηματισμό) όσες φορές και αν υπάρχει και έτσι προκύπτει ένα καινούργιο θεώρημα:

  theorem(1, [eq(plus(X,plus(1,Z)), plus(plus(X,1),Z))], []).

3.4.2 Μετασχηματισμοί Apply

Μετασχηματισμοί, οι οποίοι εφαρμόζουν ένα αξίωμα ή λήμμα ή νόμο της λογικής πρώτης τάξης ή επαγωγή. Η σύνταξή τους έχει ως εξής:

apply (AxiomOrLemmaOrInductionOrFol_law, Id, AtomAndTerm).

apply (AxiomOrLemmaOrInductionOrFol_law, Id, Literals).

apply (AxiomOrLemmaOrInductionOrFol_law, Id, Theorem).

1. Το όρισμα «AxiomOrLemmaOrInductionOrFol_law» μπορεί να πάρει μια από τις «τιμές» 'axiom', 'lemma', 'induction' ή 'fol_law'.

2. Το όρισμα «Id» είναι ένας μοναδικός προσδιορισμός για κάθε 'axiom', 'lemma', 'induction' ή 'fol_law'.

3. Το όρισμα «AtomAndTerm» παριστάνεται από μια λίστα της μορφής [Atom, Term], όπου:

a.'Atom' είναι μια λίστα της μορφής

[AtomName,Arity,LeftRightSide,LeftToRightOrder]:

- 'AtomName' είναι το όνομα του ατομικού τύπου (ατόμο) από το θεώρημα.
- 'Arity' είναι η πληθυκότητα του ατομικού τύπου (ατόμο) από το θεώρημα.
- 'LeftRightSide' μπορεί να πάρει μία από τις «τιμές» 'left' ή 'right', που υποδηλώνουν την αριστερή ή τη δεξιά μεριά του θεωρήματος αντίστοιχα.
- 'LeftToRightOrder' μπορεί να πάρει ένα θετικό φυσικό αριθμό που υποδηλώνει την σειρά του ατομικού τύπου, στο αντίστοιχο left/right μέρος της πρότασης.

b.'Term' είναι μια λίστα της μορφής

[TermNo,SubtermNo,SubsubtermNo,...].

'Οπου:

- 'TermNo' παριστά την σειρά των όρων στον αντίστοιχο ατομικό τύπο.
- 'SubtermNo' παριστά την σειρά των υποόρων στον αντίστοιχο όρο.
- 'SubsubtermNo' παριστά την σειρά των υποόρων στον αντίστοιχο υποόρο.

Παράδειγμα

Μετασχηματισμός: apply (axiom, 1, [[eq,2,left,1],[1]]).

Επεξήγηση: Το αξίωμα με ταυτότητα 1 να εφαρμοστεί στην πρώτη εμφάνιση του άτομου με όνομα eq, πληθυκότητα 2, στον πρώτο υποόρο του πρώτου όρου. Αυτό το όρισμα είναι στο αριστερό μέρος του θεωρήματος.

4. Το όρισμα 'Literals' παριστάνεται από μια λίστα της μορφής

[LeftRightSide, LiteralsList],

a. Το 'LeftRightSide' μπορεί να πάρει μία από τις «τιμές» 'left' ή 'right' που υποδηλώνουν την αριστερή ή τη δεξιά μεριά του θεωρήματος αντίστοιχα.

b. Το 'LiteralsList' είναι μία λίστα από αριθμούς που αντιπροσωπεύουν την σειρά που έχουν μερικά ορίσματα στο αριστερό ή το δεξί μέρος του θεωρήματος.

Παράδειγμα

Μετασχηματισμός: apply (axiom, 1, [left,[1,2]]).

Επεξήγηση: Το αξίωμα με ταυτότητα 1, να εφαρμοστεί στο πρώτο και δεύτερο όρισμα, του αριστερού μέρους του θεωρήματος.

5. Το όρισμα 'Theorem' παριστάνεται από μια λίστα της μορφής

[[LeftSide,LLiteralsList],[RightSide,RLiteralsList]]

a. 'LeftSide' και 'RightSide' είναι το αριστερό ή το δεξί μέρος του θεωρήματος αντίστοιχα.
b. 'LLiteralsList' είναι μια λίστα από αριθμούς που υποδηλώνουν τη σειρά μερικών ορισμάτων στο αριστερό μέρος του θεωρήματος.
c. 'RLiteralsList' είναι μια λίστα από αριθμούς που υποδηλώνουν τη σειρά μερικών ορισμάτων στο δεξί μέρος του θεωρήματος.

Παράδειγμα

Μετασχηματισμός: apply (fol_law, 6, [[left, [1]], [right, [1]]]).

Επεξήγηση: Ο νόμος της λογικής πρώτης τάξεως fol_law με ταυτότητα 6, να εφαρμοστεί στο πρώτο όρισμα του αριστερού μέρους και στο πρώτο όρισμα του δεξιού μέρους του θεωρήματος.

3.5 Αναπαράσταση των Βημάτων Απόδειξης

Κάθε βήμα απόδειξης διατηρείται ώστε να μπορεί ο χρήστης, είτε να δει όλα τα βήματα απόδειξης που έκανε, ή για να μπορεί να ακυρώσει κάποια βήματα. Κάθε βήμα απόδειξης παριστάνεται από μια πρόταση γεγονός της Prolog της παρακάτω μορφής:

proofstep(ProofstepId,FromTheoremId,ProofTransformation,NextTheoremId).

- Το κατηγόρημα proofstep παριστά ένα βήμα απόδειξης. Τα ορίσματά του παριστούν τις εξής πληροφορίες.
- Το όρισμα ProofstepId υποδηλώνει την ταυτότητα του βήματος.
- Το όρισμα FromTheoremId υποδηλώνει την ταυτότητα του θεωρήματος, στο οποίο εφαρμόζουμε το μετασχηματισμό ProofTransformation.
- Το όρισμα ProofTransformation παριστά τον μετασχηματισμό που εφαρμόζεται.
- Το όρισμα NextTheoremId, παριστά την ταυτότητα του θεωρήματος, το οποίο προκύπτει μετά την εφαρμογή του μετασχηματισμού ProofTransformation.

Τα ορίσματα ProofstepId, FromTheoremId, NextTheoremId είναι ακέραιοι θετικοί αριθμοί π.χ. 1, 100, 5, 108 κ.α..

Το όρισμα ProofTransformation είναι μια λίστα, που μπορεί να πάρει τρεις κυρίως διαφορετικές μορφές, ανάλογα με το είδος του μετασχηματισμού που εφαρμόζουμε. Οι μορφές αυτές είναι οι εξής:

1. [assign, V, Vorder].

   Με τον μετασχηματισμό «assign» ,η τιμή (V) καταχωρείται στην μεταβλητή που έχει πρώτη εμφάνιση στη σειρά Vorder. Όλες οι εμφανίσεις αυτής της μεταβλητής ξεκινώντας από αριστερά προς τα δεξιά, αντικαθίστανται από την τιμή V.

Παράδειγμα
Θεωρούμε ότι έχουμε το θεώρημα :

theorem(1, [eq(plus(X,plus(Y,Z)), plus(plus(X,Y),Z))], []).

Αν ο μετασχηματισμός που θέλουμε να εφαρμόσουμε είναι **[assign, 1, 2]** ακολουθούμε τα παρακάτω βήματα:

- Παίρνουμε όλες τις μεταβλητές του θεωρήματος όπως τις βλέπουμε, δηλαδή X, Y, Z.
- Βλέπουμε από τον μετασχηματισμό [assign, 1, 2] ότι πρέπει να καταχωρηθεί τιμή στην δεύτερη στη σειρά μεταβλητή, δηλαδή στο παράδειγμά μας είναι η Y.
- Αντικαθιστούμε αυτή τη μεταβλητή με τον αριθμό 1, (για τον συγκεκριμένο μετασχηματισμό) όσες φορές και αν υπάρχει και έτσι προκύπτει η νέα μορφή του θεωρήματος:

theorem(1, [eq(plus(X,plus(1,Z)), plus(plus(X,1),Z))], []).

2. [apply, AxiomOrLemmaOrInduction, AxLmIndId, Theorem/ Literals/Atom&Term].

   Με τον μετασχηματισμό «apply» εφαρμόζουμε το αξίωμα, λήμμα, επαγωγή ή νόμο του προτασιακού και κατηγορηματικού λογισμού (AxiomOrLemmaOrInduction), με ταυτότητα(AxLmIndId), στον όρο του θεωρήματος, που του υποδεικνύουμε με τα Theorem, Literals, Atom&Term.

   Παράδειγμα

   Έστω ότι ο μετασχηματισμός που θέλουμε να εφαρμοστεί είναι ο εξής:

   [apply,axiom,3,theorem].

   Έστω το αξίωμα 3 είναι : ∀ M: N (even(succ(succ(M)))→ even(M)) και το θεώρημα που έχουμε είναι : theorem(1, [even(succ(succ(M)),true], []).Τότε εφαρμόζοντας το αξίωμα με ταυτότητα 3 σε όλο το θεώρημα (theorem) προκύπτει το θεώρημα theorem(2, [even(M) ,true], []).

3. [apply, equality_theory, [[AtomName,Arity,left,LeftToRightOrder], Term]].

   Με αυτόν τον μετασχηματισμό «apply» εφαρμόζουμε ένα από τα αξιώματα της *Θεωρίας της Ισότητας*, στο σημείο του θεωρήματος που του υποδεικνύουμε με το

   [[AtomName, Arity, left, LeftToRightOrder], Term]].

   Η εξήγηση των ορισμάτων της παραπάνω λίστας είναι η εξής.
   Ο όρος (Term), βρίσκεται στο άτομο του θεωρήματος, που έχει όνομα (AtomName) και πληθυκότητα (Arity). Το άτομο αυτό είναι στο αριστερό μέρος (left) του θεωρήματος και η σειρά του από τα αριστερά προς τα δεξιά ανάμεσα στα άλλα άτομα είναι LeftToRightOrder.

   Παράδειγμα

   Έστω ότι ο μετασχηματισμός που θέλουμε να εφαρμοστεί είναι [apply, equality_theory, [[eq,2,left,1],[3]]. Αυτό σημαίνει ότι θα ξεκινήσει η εφαρμογή του αξιώματος που επιθυμούμε π.χ. 1 (∀ x = x) από την *Θεωρία της Ισότητας*, στον όρο 3 του ατομικού τύπου, ο οποίος έχει όνομα eq, πληθυκότητα 2, βρίσκεται στο αριστερό μέρος του θεωρήματος και είναι πρώτο στη σειρά μεταξύ των υπόλοιπων ατομικών τύπων.
   Έστω ότι το θεώρημα που έχουμε είναι : theorem(1, [eq(plus(X,Z), plus(X,Z))], []).
   Εφαρμόζοντας τότε τα παραπάνω προκύπτει το θεώρημα: theorem(2, [true], []).

4 Ένα Διαλογικό Σύστημα Απόδειξης Θεωρημάτων με τη Μέθοδο της Μαθηματικής Επαγωγής

4.1 Αρχιτεκτονική Συστήματος και Περιγραφή Κύριων Τμημάτων του

Το σύστημα **Ευκλείδης** αποτελείται από τα εξής υποσυστήματα, Σχήμα 4.1:

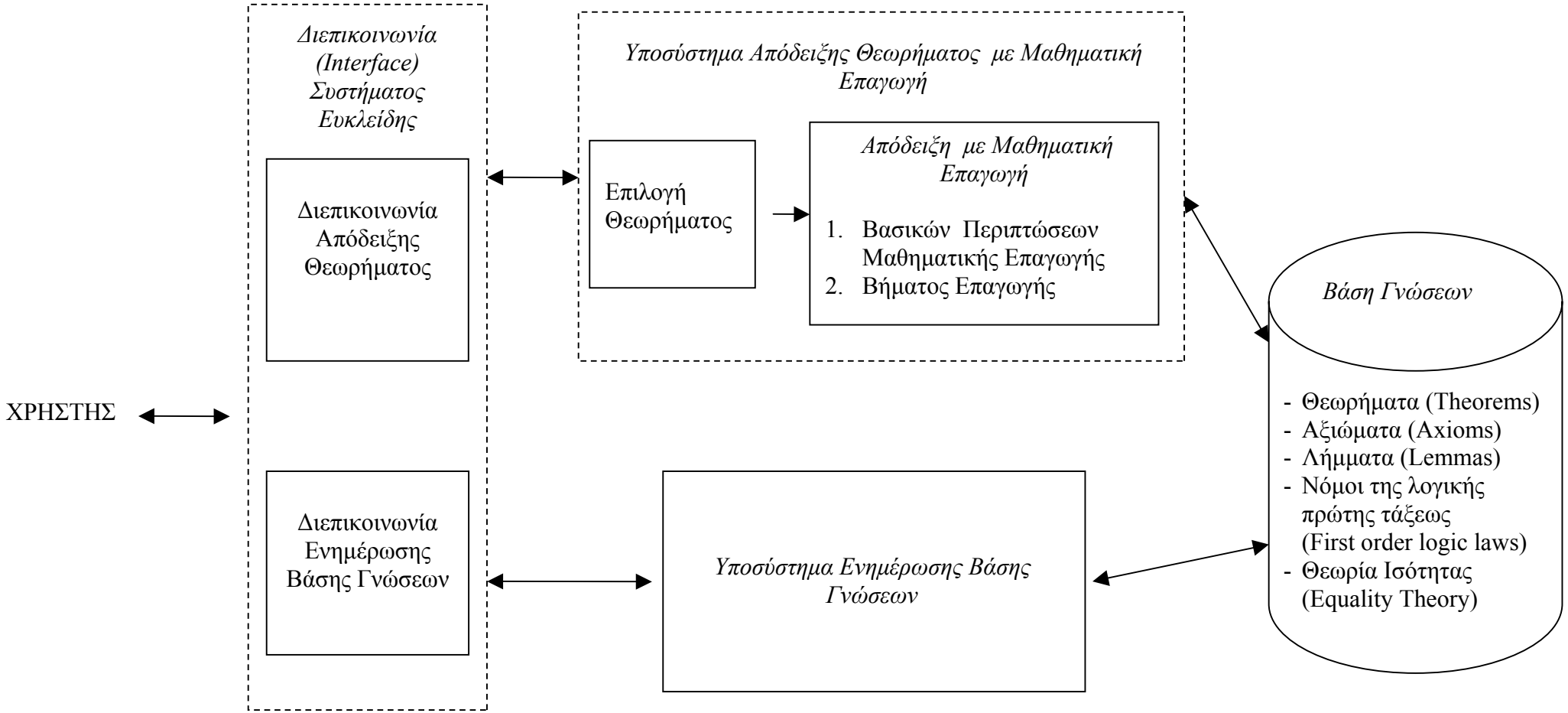
1. Το υποσύστημα «απόδειξης θεωρήματος με μαθηματική επαγωγή».
2. Το υποσύστημα «ενημέρωσης βάσης γνώσεων».
3. Διεπικοινωνία συστήματος Ευκλείδης.

Το υποσύστημα «απόδειξης θεωρημάτων με μαθηματική επαγωγή» είναι το πλέον σημαντικό. Η συνεισφορά αυτής της πτυχιακής εργασίας βρίσκεται στον σχεδιασμό και στην υλοποίηση αυτού του υποσυστήματος. Αυτό θα παρουσιάσουμε με λεπτομέρειες στο κεφάλαιο 4. Το υποσύστημα «ενημέρωσης βάσης γνώσεων» είναι πολύ απλό και δε χρειάζεται ιδιαίτερη παρουσίαση. Την διεπικοινωνία του συστήματος Ευκλείδης θα την μελετήσουμε αναλυτικά παρακάτω.

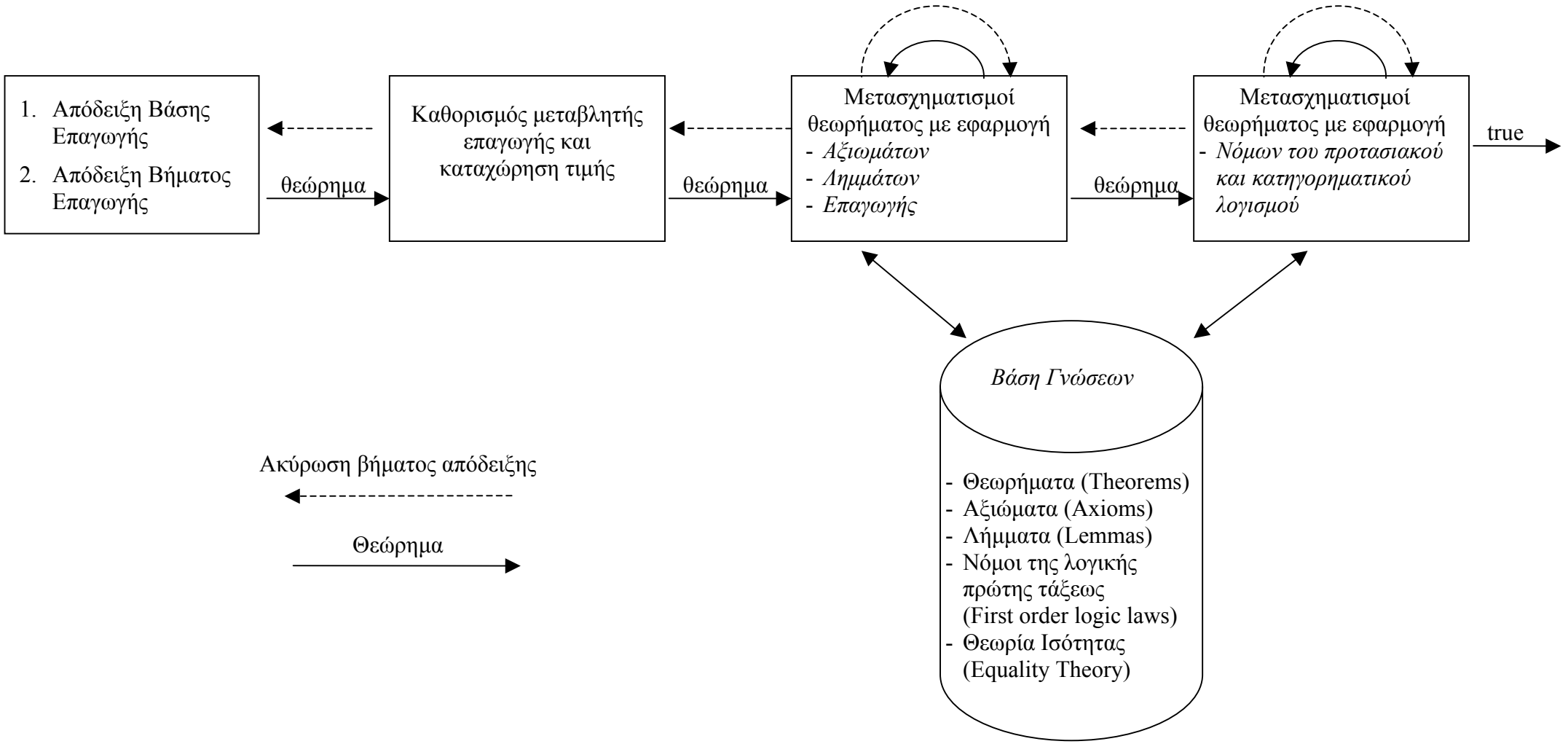
Το υποσύστημα «απόδειξης θεωρημάτων με μαθηματική επαγωγή» αποτελείται από ένα αριθμό τμημάτων που αλληλεπιδρούν μεταξύ τους. Αυτή η αλληλεπίδραση φαίνεται σχηματικά στο Σχήμα 4.2. Αρχικά το υποσύστημα «απόδειξης θεωρημάτων με μαθηματική επαγωγή» δέχεται σαν είσοδο την ταυτότητα του θεωρήματος, που θέλουμε ν' αποδείξουμε.

Το θεώρημα επιλέγεται από τη βάση γνώσεων, στην οποία έχει καταχωρηθεί από το υποσύστημα «ενημέρωσης βάσης γνώσεων». Η αναπαράσταση του θεωρήματος στη βάση γνώσεων είναι σε μη βασική μορφή. Στη συνέχεια επιλέγουμε, εάν θα αποδείξουμε τη βάση της επαγωγής ή το βήμα της. Εάν επιλεγεί η απόδειξη της βάσης της επαγωγής για την πρόταση θα πρέπει να καθοριστεί η επαγωγική μεταβλητή και να καταχωρηθεί αρχική τιμή σ' αυτήν. Εάν επιλεγεί η απόδειξη του βήματος της επαγωγής, αυτό σημαίνει ότι η βάση επαγωγής έχει ήδη αποδειχθεί. Συνεπώς προχωρούμε σε μετασχηματισμό του θεωρήματος που πρέπει ν' αποδειχθεί. Εφαρμόζουμε τους μετασχηματισμούς χρησιμοποιώντας αξιώματα, λήμματα, επαγωγές, νόμους της λογικής πρώτης τάξης και θεωρία της ισότητας. Όλα τα παραπάνω έχουν αποθηκευτεί στη βάση γνώσεων, πάντα σε μη βασική μορφή. Ολοκληρώνουμε την απόδειξή μας για κάποιο θεώρημα, μόνο εάν έχουμε επιλέξει τουλάχιστον μία φορά το βήμα βάσης και το βήμα επαγωγής, σύμφωνα με την απόδειξη θεωρήματος δια της επαγωγικής μεθόδου. Σε αντίθετη περίπτωση δεν μπορούμε να μιλάμε για απόδειξη θεωρήματος, παρά για απόδειξη βήματος βάσης ή απόδειξη βήματος επαγωγής.

Σχηματικά τα βήματα που ακολουθούμε φαίνονται στο Σχήμα 4.1 και στο Σχήμα 4.2:


Διεπικοινωνία (Interface) Συστήματος Ευκλείδης
Διεπικοινωνία Απόδειξης Θεωρήματος
Διεπικοινωνία Ενημέρωσης Βάσης Γνώσεων
ΧΡΗΣΤΗΣ
Υποσύστημα Απόδειξης Θεωρήματος με Μαθηματική Επαγωγή
Επιλογή Θεωρήματος
Απόδειξη με Μαθηματική Επαγωγή
1. Βασικών Περιπτώσεων Μαθηματικής Επαγωγής
2. Βήματος Επαγωγής
Υποσύστημα Ενημέρωσης Βάσης Γνώσεων
Βάση Γνώσεων
- Θεωρήματα (Theorems)
- Αξιώματα (Axioms)
- Λήμματα (Lemmas)
- Νόμοι της λογικής πρώτης τάξεως (First order logic laws)
- Θεωρία Ισότητας (Equality Theory)

Σχήμα 4.1: *Αρχιτεκτονική Συστήματος Ευκλείδης.*



Σχήμα 4.2: *Υποσύστημα απόδειξης θεωρήματος με μαθηματική επαγωγή.*

4.2 Κορυφαίοι Αλγόριθμοι του Συστήματος Ευκλείδης

4.2.1 Εισαγωγή

Στο τμήμα 4.1 παρουσιάσαμε τα τμήματα του υποσυστήματος «απόδειξη θεωρήματος με μαθηματική επαγωγή» του συστήματος Ευκλείδης. Σ' αυτό το κεφάλαιο θα παρουσιάσουμε σημαντικούς αλγόριθμους του υποσυστήματος «απόδειξη θεωρήματος με μαθηματική επαγωγή» καθώς και τον τρόπο με τον οποίο συνδέονται μεταξύ τους.

Θα ξεκινήσουμε με τον αλγόριθμο που περιγράφει γενικά την δομή του υποσυστήματος.

4.2.2 Κορυφαίος Αλγόριθμος Συστήματος Ευκλείδης

Η διαδικασία prove είναι η πρώτη διαδικασία που τρέχει όταν ξεκινήσουμε το σύστημά μας. Δεν έχει παραμέτρους, απλά καλείται για να ξεκινήσει το σύστημα. Επιπλέον συντονίζει τις κλήσεις των διαφόρων τμημάτων του υποσυστήματος «απόδειξης θεωρημάτων».

Είσοδος:-
Έξοδος:-
Στόχος:
prove.
Άλλες δηλώσεις:
epilogi: μεταβλητή που παίρνει μια από τις παρακάτω τιμές : b, ind, lth, e, όπου έχουμε:
b: για απόδειξη της βάσης επαγωγής,
ind: για απόδειξη του βήματος επαγωγής,
lth: για εκτύπωση του τελευταίου θεωρήματος,
e: για το τέλος της διαδικασίας.

```
διαδικασία prove( );
αρχή_διαδ
   επανάλαβε
      εκτύπωσε 'Give the theorem';
      διάβασε TheoremId;
      εκτύπωσε τις επιλογές του χρήστη
                  {  b for base step
                     ind for induction step
                     lth to see theorem
                     e for end }
      διάβασε epilogi;
         περίπτωση
            epilogi = b
                { prove_basis(TheoremId);}
            epilogι = ind
```

```
            { prove_step(TheoremId);}
        epilogι = lth
            { list_theorem1(TheoremId);}
    τέλος_περ
μέχρι epilogι = e.
τέλος_διαδ
```

Αλγόριθμος 4.1:*Αλγόριθμος κορυφαίας διαδικασίας συστήματος Ευκλείδης.*

Η διαδικασία αυτή δέχεται ως είσοδο την ταυτότητα του θεωρήματος το οποίο επιθυμούμε ν' αποδείξουμε, ενώ δεν επιστρέφει κάποια παράμετρο στην έξοδο. Η ***prove_basis****(TheoremId) ξεκινάει την απόδειξη της βάσης επαγωγής και συντονίζει τις κλήσεις των διαφόρων τμημάτων του υποσυστήματος αυτού.*

Είσοδος:
TheoremId : Το θεώρημα που θα αποδειχθεί.
Έξοδος:-
Στόχος:
prove_basis(2).
Άλλες δηλώσεις:
Action : μεταβλητή που παίρνει μια από τις παρακάτω τιμές : lth, lps, allth, allps, tr, c, s, e. *Η εξήγηση των ενεργειών αυτών των τιμών είναι:*

- *lth: εκτύπωσε το τελευταίο θεώρημα,*
- *lps: εκτύπωσε το τελευταίο βήμα απόδειξης,*
- *allth: εκτύπωσε όλα τα θεωρήματα,*
- *allps: εκτύπωσε όλα τα βήματα απόδειξης,*
- *tr: περισσότεροι μετασχηματισμοί,*
- *c: ακύρωσε το τελευταίο βήμα απόδειξης,*
- *s: εφαρμογή νόμων πρώτης τάξης καθώς και θεωρία της ισότητας,*
- *e: τέλος.*

```
διαδικασία prove_basis(TheoremId);
αρχή_διαδ
   επανάλαβε
     εκτύπωσε 'Give the transformation';
     διάβασε Transf
        περίπτωση
          Transf = assign
              { assign_values, get_and_perform_basis__Action(Action);}
          Transf = apply(AxiomOrLemmaOrInduction, AxLmIndId, Literals)
              { perform_transf_apply( TheoremId, Transf, NewTheoremId),
                                  get_and_perform_basis__Action(Action);}
        τέλος_περ
μέχρι Action = e.
τέλος_διαδ
```

Αλγόριθμος 4.2:*Αλγόριθμος κορυφαίας διαδικασίας συστήματος Ευκλείδης.*

Η διαδικασία αυτή δέχεται ως είσοδο την ταυτότητα του θεωρήματος το οποίο επιθυμούμε ν' αποδείξουμε, ενώ δεν επιστρέφει κάποια παράμετρο στην έξοδο. Η

prove_step*(TheoremId) ξεκινάει την απόδειξη του βήματος επαγωγής και συντονίζει τις κλήσεις των διαφόρων τμημάτων του υποσυστήματος αυτού.*

Είσοδος:
TheoremId : Το θεώρημα που θα αποδειχθεί.
Έξοδος:-
Στόχος:
prove_step(2).
Άλλες δηλώσεις:
Action : μεταβλητή που παίρνει μια από τις παρακάτω τιμές : lth, lps, allth, allps, tr, c, s, e. *Η εξήγηση των ενεργειών αυτών των τιμών είναι:*

- *lth: εκτύπωσε το τελευταίο θεώρημα,*
- *lps: εκτύπωσε το τελευταίο βήμα απόδειξης,*
- *allth: εκτύπωσε όλα τα θεωρήματα,*
- *allps: εκτύπωσε όλα τα βήματα απόδειξης,*
- *tr: περισσότεροι μετασχηματισμοί,*
- *c: ακύρωσε το τελευταίο βήμα απόδειξης,*
- *s: εφαρμογή νόμων πρώτης τάξης καθώς και θεωρία της ισότητας,*
- *e: τέλος.*

διαδικασία **prove_step**(TheoremId);
αρχή_διαδ
επανάλαβε
εκτύπωσε 'Give the transformation';
διάβασε Transf
περίπτωση
Transf = assign
{ ***assign_values, get_and_perform_step__Action(Action);***}
Transf = apply(AxiomOrLemmaOrInduction, AxLmIndId, Literals)
{ ***perform_transf_apply(TheoremId, Transf, NewTheoremId),***
get_and_perform_ step__Action(Action);}
τέλος_περ
μέχρι Action = e**.**
τέλος_διαδ

Αλγόριθμος 4.3 :*Αλγόριθμος κορυφαίας διαδικασίας συστήματος Ευκλείδης.*

Η διαδικασία αυτή δέχεται ως είσοδο την ταυτότητα του θεωρήματος το οποίο επιθυμούμε ν' αποδείξουμε, ή το τελευταίο θεώρημα που έχει δημιουργηθεί, ενώ δεν επιστρέφει κάποια παράμετρο στην έξοδο. Η ***list_theorem1****(TheoremId) εκτυπώνει το τελευταίο θεώρημα που έχει δημιουργηθεί ή το θεώρημα που επιθυμούμε ν' αποδείξουμε.*

Είσοδος:
TheoremId : Το τελευταίο θεώρημα που έχει δημιουργηθεί ή το θεώρημα που θα αποδειχθεί.

Έξοδος:-
Στόχοι:
list_theorem1(2).
list_theorem1(100).

διαδικασία **list_theorem1**(TheoremId);
αρχή_διαδ
διάβασε TheoremId;
εκτύπωσε το θεώρημα;
τέλος_διαδ

Αλγόριθμος 4.4:*Αλγόριθμος κορυφαίας διαδικασίας συστήματος Ευκλείδης*

Η διαδικασία αυτή δέχεται ως είσοδο μια από τις τιμές: lth, lps, allth, allps, tr, c, s, e, κάθε μία από τις οποίες αντιπροσωπεύει και μια συγκεκριμένη ενέργεια του προγράμματος. Η εξήγηση των ενεργειών αυτών των τιμών είναι:

- *lth: εκτύπωσε το τελευταίο θεώρημα,*
- *lps: εκτύπωσε το τελευταίο βήμα απόδειξης,*
- *allth: εκτύπωσε όλα τα θεωρήματα,*
- *allps: εκτύπωσε όλα τα βήματα απόδειξης,*
- *tr: περισσότεροι μετασχηματισμοί,*
- *c: ακύρωσε το τελευταίο βήμα απόδειξης,*
- *s: εφαρμογή νόμων πρώτης τάξης καθώς και θεωρία της ισότητας,*
- *e: τέλος.*

Είσοδος:
Action : μεταβλητή που παίρνει μια από τις παρακάτω τιμές: lth, lps, allth, allps, tr, c, s, e.
Έξοδος:-
Στόχος:
get_and_perform_basis__Action(tr).
Άλλες δηλώσεις :
NewAction : μεταβλητή που παίρνει μια από τις παρακάτω τιμές : lth, lps, allth, allps, tr, c, s, e.

διαδικασία **get_and_perform_basis__Action(**Action);
αρχή_διαδ
επανάλαβε
εκτύπωσε τις επιλογές του χρήστη
{ lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
tr for more transformations
c for cancelling last step
s for symbolic_evaluation
e for end }
διάβασε Action;
περίπτωση
Action = lth
{list_theorem(TheoremId), get_and_perform_basis__Action(NewAction);}

```
        Action = lps
           {list_proofstep(ProofStepId), get_and_perform_basis__Action(NewAction);}
        Action = allth
           {list_all_theorems, get_and_perform_basis__Action(NewAction); }
        Action = allps
           {list_all_proofsteps, get_and_perform_basis__Action(NewAction); }
        Action = c
           {delete_proofstep, get_and_perform_basis__Action(NewAction); }
        Action = s
           {symbolic_evaluation_control(Control); }
        Action = tr
           {prove_basis(NewTheoremId);}
         τέλος_περ
    μέχρι Action = e.
τέλος_διαδ
```

Αλγόριθμος 4.5:*Αλγόριθμος συστήματος Ευκλείδης.*

Η διαδικασία αυτή δέχεται ως είσοδο μια από τις τιμές: lth, lps, allth, allps, tr, c, s, e, κάθε μία από τις οποίες αντιπροσωπεύει και μια συγκεκριμένη ενέργεια του προγράμματος. Η εξήγηση των ενεργειών αυτών των τιμών είναι:

- *lth: εκτύπωσε το τελευταίο θεώρημα,*
- *lps: εκτύπωσε το τελευταίο βήμα απόδειξης,*
- *allth: εκτύπωσε όλα τα θεωρήματα,*
- *allps: εκτύπωσε όλα τα βήματα απόδειξης,*
- *tr: περισσότεροι μετασχηματισμοί,*
- *c: ακύρωσε το τελευταίο βήμα απόδειξης,*
- *s: εφαρμογή νόμων πρώτης τάξης καθώς και θεωρία της ισότητας,*
- *e: τέλος.*

Είσοδος:
Action : μεταβλητή που παίρνει μια από τις παρακάτω τιμές: lth, lps, allth, allps, tr, c, s, e.
Έξοδος:-
Στόχος:
get_and_perform_step__Action(allth).
Άλλες δηλώσεις :
NewAction : μεταβλητή που παίρνει μια από τις παρακάτω τιμές : lth, lps, allth, allps, tr, c, s, e.

```
διαδικασία get_and_perform_step__Action(Action);
αρχή_διαδ
   επανάλαβε
     εκτύπωσε τις επιλογές του χρήστη
                          {  lth to see last theorem
                             lps to see last proofstep
                             allth to see all theorems
                             allps to see all proofsteps
                             tr for more transformations
```

```
                                c for cancelling last step
                                s for symbolic_evaluation
                                e for end }
        διάβασε Action;
          περίπτωση
         Action = lth
           {list_theorem(TheoremId), get_and_perform_step__Action(NewAction);}
         Action = lps
           {list_proofstep(ProofStepId), get_and_perform_step__Action(NewAction); }
         Action = allth
           {list_all_theorems, get_and_perform_step__Action(NewAction); }
         Action = allps
           {list_all_proofsteps, get_and_perform_step__Action(NewAction); }
         Action = c
           {delete_proofstep, get_and_perform_step__Action(NewAction); }
         Action = s
           {symbolic_evaluation_control(Control); }
         Action = tr
           {prove_basis(NewTheoremId);}
          τέλος_περ
      μέχρι Action = e.
τέλος_διαδ
```

Αλγόριθμος 4.6:*Αλγόριθμος συστήματος Ευκλείδης.*

4.3 Αλγόριθμοι Μετασχηματισμού Θεωρήματος

Παρακάτω περιγράφεται το τμήμα, στο οποίο καθορίζεται η μεταβλητή επαγωγής και το οποίο είναι αρμόδιο για καταχώρηση τιμής σε αυτήν. Επίσης περιγράφονται τα τμήματα, που πραγματοποιούν μετασχηματισμούς στο θεώρημα με εφαρμογή αξιωμάτων, λημμάτων, επαγωγής και νόμων του προτασιακού και κατηγορηματικού λογισμού.

4.3.1 Αλγόριθμος Καθορισμού Μεταβλητής Επαγωγής και Καταχώρησης Τιμής σ' αυτήν

Η διαδικασία αυτή καθορίζει τη μεταβλητή επαγωγής και καταχωρεί κάποια τιμή σ' αυτή. Οι μεταβλητές ***LastThId*** *και* ***New_LastThId*** *αντιπροσωπεύουν αντίστοιχα, την ταυτότητα του τελευταίου θεωρήματος και του θεωρήματος που προκύπτει μετά από μετασχηματισμό αυτού. Αρχική τιμή για το* ***LastThId*** *είναι το 99, ενώ το* ***New_LastThId*** *είναι ίσο με το* ***LastThId*** *προσαυξημένο κατά ένα (New_LastThId =LastThId+1).*

Είσοδος:-
Έξοδος:-

Στόχος:
assign_values.

διαδικασία **assign_values**( );
αρχή_διαδ
διάβασε TheoremId;
διάβασε LastThId;
δημιούργησε το καινούργιο θεώρημα **New_LastThId**;
αντικατέστησε το **LastThId** με το **New_LastThId**;
print_theorem(Left, Right);
% Εκτύπωσε το θεώρημα.
εκτύπωσε 'Give variable order e.g. 1,2,3.';
διάβασε Vorder;
% Καθορισμός Μεταβλητής Επαγωγής.
get_var(VarsList, Vorder, Var);
% Βρες τη μεταβλητή Var, που έχει θέση Vorder, ανάμεσα στη λίστα όλων των μεταβλητών VarsList.
εκτύπωσε 'Give variable value:';
διάβασε V;
καταχώρησε το νέο θεώρημα, αντικαθιστώντας στο αρχικό τη μεταβλητή που βρίσκεται στη θέση Vorder με την τιμή V;
δημιούργησε το **ProofTransformation;**
proofstep(ProofStepId, Id, ProofTransformation, New_LastThId);
% Δημιούργησε το βήμα απόδειξης για τον συγκεκριμένο μετασχηματισμό.
print_varsList_theorem(Left, Right);
% Εκτύπωσε το καινούργιο θεώρημα και τις εναπομένουσες μεταβλητές.
τέλος_διαδ

Αλγόριθμος 4.7*:Αλγόριθμος καθορισμού της μεταβλητής επαγωγής και καταχώρηση τιμής σε αυτήν.*

Η διαδικασία αυτή δέχεται ως είσοδο τα Left_th και Right_th, που είναι αντίστοιχα το αριστερό και το δεξί μέρος του θεωρήματος και δεν επιστρέφει έξοδο. Η ***print_theorem****(Left_th, Right_th) εκτυπώνει το δεξί και αριστερό μέρος του θεωρήματος καθώς και τις μεταβλητές που υπάρχουν σ' αυτό.*

Είσοδος:
Left_th: το αριστερό μέρος του θεωρήματος.
Right_th: το δεξί μέρος του θεωρήματος.
Έξοδος:-
Στόχος:
print_theorem([even(X),even(Y)],[even(plus(X,Y))]).

διαδικασία **print_theorem**(Left_th, Right_th);
αρχή_διαδ
term_variables_bag*(*Left_th, LeftVarsList);
% Βρες από το Left_th όλες τις μεταβλητές και δώσε τη λίστα LeftVarsList.

```
      term_variables_bag(Right_th, RightVarsList);
        % Βρες από το Right_th όλες τις μεταβλητές και δώσε τη λίστα RightVarsList.
      append(LeftVarsList, RightVarsList,LRVarsList);
        % Ένωσε τις λίστες LeftVarsList,RightVarsList και δώσε τη λίστα LRVarsList.
      διάβασε QuotedNamesL;
        % QuotedNamesL είναι μια λίστα με ονόματα μεταβλητών που έχουμε ορίσει.
      term_variables_bag(LRVarsList, New_LR);
        % Βρες από το LRVarsList όλες τις μεταβλητές και δώσε τη λίστα New_LR.
      assign_vars_quotedNames(New_LR, QuotedNamesL);
        % Αντικατέστησε τις μεταβλητές της λίστας New_LR με ονόματα μεταβλητών
          από τη λίστα  QuotedNamesL.
      εκτύπωσε 'THEOREM.';
      εκτύπωσε 'Theorem Left: ';
      εκτύπωσε  Left_th;
      εκτύπωσε 'Theorem Right: ';
      εκτύπωσε  Right_th;
      εκτύπωσε 'THEOREM VARIABLES';
      εκτύπωσε 'Variables:';
      εκτύπωσε  LRVarsList;
τέλος_διαδ
```

Αλγόριθμος 4.8*:Βοηθητικός αλγόριθμος υλοποίησης για τον καθορισμό μεταβλητής επαγωγής και καταχώρηση τιμής σε αυτήν.*

Η διαδικασία αυτή δέχεται ως είσοδο μια λίστα μεταβλητών Vars τις οποίες αντικαθιστά με ονόματα μεταβλητών, που έχουμε ήδη ορίσει και μας δίνει ως έξοδο την λίστα Qnames.

Σημείωση*: Λεπτομέρειες για αυτή τη διαδικασία στο κώδικα.*

Είσοδος:
Vars: μια λίστα μεταβλητών, π.χ. [X, Y, Z]
Έξοδος:
Qnames: Η παραπάνω λίστα Vars έχοντας όμως ονομάσει τις μεταβλητές της με ονόματα που έχουμε ορίσει αρχικά.
Στόχος:
assign_vars_quotedNames([X,Y],['A','B','C','D','E','F','G','H','I'|...]).

```
διαδικασία assign_vars_quotedNames(Vars, Qnames);
αρχή_διαδ
   …
τέλος_διαδ
```

Αλγόριθμος 4.9:*Βοηθητικός αλγόριθμος υλοποίησης για τον καθορισμό μεταβλητής επαγωγής και καταχώρηση τιμής σε αυτήν.*

Η διαδικασία αυτή δέχεται ως είσοδο μια λίστα μεταβλητών Vars και ένα αριθμό N, ο οποίος προσδιορίζει τη θέση (εντός αυτής της λίστας) της μεταβλητής V, που παίρνουμε στην έξοδο.

***Σημείωση**: Λεπτομέρειες για αυτή τη διαδικασία στο κώδικα.*

Είσοδος:
Vars: μια λίστα μεταβλητών, όπως την βλέπουμε στο θεώρημα.
N: ένας ακέραιος αριθμός, που δηλώνει τη θέση κάποιας μεταβλητής μέσα στην παραπάνω λίστα.

Έξοδος:
V: η μεταβλητή που αντιστοιχεί στη θέση N των ορισμάτων της λίστας Vars.

Στόχος:
get_var([X,Y,Z],1,X).

διαδικασία **get_var**(Vars, N, V);
αρχή_διαδ
…
τέλος_διαδ

Αλγόριθμος 4.10:*Βοηθητικός αλγόριθμος υλοποίησης για τον καθορισμό μεταβλητής επαγωγής και καταχώρηση τιμής σε αυτήν.*

Η διαδικασία αυτή δεν δέχεται είσοδο αλλά δίνει ως έξοδο την ταυτότητα του βήματος απόδειξης NewProofStepId, την ταυτότητα του θεωρήματος στο οποίο εφαρμόζουμε τον μετασχηματισμό FromTheoremId, την ταυτότητα του θεωρήματος που προέκυψε NextTheoremId, καθώς και το μετασχηματισμό που εφαρμόσθηκε ProofTransformation.

Είσοδος:-
Έξοδος:
NewProofStepId: η ταυτότητα του βήματος απόδειξης.
FromTheoremId: η ταυτότητα του θεωρήματος, στο οποίο εφαρμόζουμε τον μετασχηματισμό.
ProofTransformation: ο μετασχηματισμός που εφαρμόζουμε.
NextTheoremId: η ταυτότητα του θεωρήματος που προέκυψε μετά την εφαρμογή του μετασχηματισμού.

Στόχοι:
- Για τον καθορισμό μεταβλητής επαγωγής και καταχώρηση τιμής σε αυτήν: proofstep(1,2,[assign,0,1],100)**.**
- Για μετασχηματισμούς θεωρημάτων με εφαρμογή αξιωμάτων, λημμάτων, επαγωγών, και νόμων λογικής πρώτης τάξης. proofstep(2,100,[apply,axiom,3,[[even,1,left,1], [0]]],101)**.**

διαδικασία **proofstep**(NewProofStepId, FromTheoremId, ProofTransformation, NextTheoremId);
αρχή_διαδ
διάβασε ProofStepId;

δημιούργησε το καινούργιο βήμα απόδειξης **NewProofStepId**;
αντικατέστησε το **ProofStepId** με το **NewProofStepId**;
καταχώρησε το νέο βήμα απόδειξης;
τέλος_διαδ

Αλγόριθμος 4.11:*Αλγόριθμος υλοποίησης του βήματος απόδειξης.*

Η διαδικασία αυτή δέχεται ως είσοδο τα Left_th και Right_th, που είναι αντίστοιχα το αριστερό και το δεξί μέρος του θεωρήματος και δεν επιστρέφει έξοδο. Η ***print_varsList_theorem****(Left_th, Right_th) εκτυπώνει τα Left_th και Right_th καθώς και λίστα των μεταβλητών που υπάρχουν στο θεώρημα.*

Είσοδος:
Left_th: το αριστερό μέρος του θεωρήματος.
Right_th: το δεξί μέρος του θεωρήματος.
Έξοδος:-
Στόχος:
print_varsList_theorem([even(X),even(Y)],[even(plus(X,Y))]).

διαδικασία **print_varsList_theorem**(Left_th, Right_th);
αρχή_διαδ
εκτύπωσε '********** '.;
term_variables_bag*(*Left_th, LeftVarsList);
% Βρες από το Left_th όλες τις μεταβλητές και δώσε τη λίστα LeftVarsList.
term_variables_bag(Right_th, RightVarsList);
% Βρες από το Right_th όλες τις μεταβλητές και δώσε τη λίστα RightVarsList.
append(LeftVarsList, RightVarsList,LRVarsList);
% Ένωσε τις λίστες LeftVarsList,RightVarsList και δώσε τη λίστα LRVarsList.
διάβασε QuotedNamesL;
% QuotedNamesL είναι μια λίστα με ονόματα μεταβλητών που έχουμε ορίσει.
assign_vars_quotedNames(New_LR, QuotedNamesL);
% Αντικατέστησε τις μεταβλητές της λίστας New_LR με ονόματα μεταβλητών από τη λίστα QuotedNamesL.
εκτύπωσε 'THEOREM';
εκτύπωσε 'write Left Theorem: ';
εκτύπωσε Left_th;
εκτύπωσε 'write Right Theorem: ';
εκτύπωσε Right_th;
εκτύπωσε 'THEOREM VARIABLES';
εκτύπωσε 'write: ';
εκτύπωσε LRVarsList;
τέλος_διαδ

Αλγόριθμος 4.12:*Βοηθητικός αλγόριθμος υλοποίησης για τον καθορισμό μεταβλητής επαγωγής και καταχώρηση τιμής σε αυτήν.*

4.3.2 Αλγόριθμος Μετασχηματισμού Θεωρήματος με Εφαρμογή Αξιωμάτων, Λημμάτων και Επαγωγής.

Η διαδικασία αυτή δέχεται είσοδο την ταυτότητα του θεωρήματος TheoremId στο οποίο θα εφαρμόσουμε τον μετασχηματισμό Transf και επιστρέφει έξοδο την ταυτότητα του θεωρήματος που θα προκύψει . Η ***perform_transf_apply****(TheoremId, Transf, NewTheoremId)* *ξεκινάει τον μετασχηματισμό του τελευταίου θεωρήματος εφαρμόζοντας αξιώματα, λήμματα και επαγωγή και συντονίζει τις κλήσεις των διαφόρων τμημάτων του υποσυστήματος αυτού.*

Είσοδος:
TheoremId: το θεώρημα στο οποίο θα εφαρμόσουμε τον μετασχηματισμό.
Transf: ο μετασχηματισμός που θα εφαρμόσουμε.
Έξοδος:
NewTheoremId: το θεώρημα που προκύπτει από την εφαρμογή αυτού του μετασχηματισμού.
Στόχοι:

- perform_transf_apply(101,apply(axiom,1,[[eq,2,left,1],[1]]), NewTheoremId), όπου NewTheoremId = 102.
- perform_transf_apply(2,apply(axiom,1,[[even,1,right,1],[1]]), NewTheoremId), όπου NewTheoremId = 100.

διαδικασία **perform_transf_apply**(TheoremId, Transf, NewTheoremId);
αρχή_διαδ
διάβασε TheoremId;
διάβασε LastThId;
διάβασε Transf = apply(AxOrLemOrInd, AxLmIndId, Theorem);
περίπτωση
Theorem = [[AtomName,Arity,LeftOrRight,LeftToRightOrder],TermNo]
{
get_literal_from_theorem(LRForm,LeftToRightOrder,TheoremLiteral),
perform_apply(TheoremLiteral,AxOrLemOrInd,AxLmIndId,TermNo,
NewTheoremLiteral),
put_newLiteral_to_LRtheorForm(RightForm, LeftToRightOrder,
NewTheoremLiteral, NewLRForm),
ProofTransformation = [apply, AxOrLemOrInd, AxLmIndId, Theorem];
}
Theorem = [LeftOrRight, LiteralsList]
{
get_literals_from_theorem(LRForm, LiteralsList, TheoremLiterals),
apply_axiomToLiterals(TheoremLiterals, AxOrLemOrInd, AxLmIndId,
NewTheoLiterals),
put_newTheoLiterals_to_LRtheoForm(LRForm, LiteralsList,
NewTheoLiterals, NewLRForm),
ProofTransformation=[apply, AxOrLemOrInd, AxLmIndId, Literals]
}
Theorem = [[left, TheoremLList], [right, TheoremRList]]
{

get_literals_from_theorem(LeftForm,TheoremLList, TheoremLLiterals),
get_literals_from_theorem(RightForm,TheoremRList, TheoremRLiterals),
apply_axiom_to_theorem(TheoremLLiterals, TheoremRLiterals,
AxOrLemOrInd,AxLmIndId, NewTheoLiterals),
put_newTheoLiterals_to_LRtheoForm(LeftForm, TheoremLList,
NewTheoLiterals, NewLeftForm),
ProofTransformation=[apply, AxOrLemOrInd, AxLmIndId, Theorem];
}
τέλος_περ

δημιούργησε το καινούργιο θεώρημα **NewTheoremId**;
αντικατέστησε το **LastThId** με το **NewTheoremId**;
καταχώρησε το νέο θεώρημα, μετά την εφαρμογή του μετασχηματισμού;
δημιούργησε το **ProofTransformation;**
proofstep(ProofStepId, Id, ProofTransformation, NewTheoremId);
% Δημιούργησε το βήμα απόδειξης για τον συγκεκριμένο μετασχηματισμό.
τέλος_διαδ

Αλγόριθμος 4.13:*Αλγόριθμος υλοποίησης για την εφαρμογή μετασχηματισμού.*

Η διαδικασία αυτή δέχεται ως είσοδο μια λίστα ορισμάτων LRForm και ένα αριθμό LeftToRightOrder, ο οποίος προσδιορίζει τη θέση (εντός αυτής της λίστας) του ορίσματος TheoremLiteral , που παίρνουμε στην έξοδο.

Σημείωση*: Λεπτομέρειες για αυτή τη διαδικασία στο κώδικα.*

Είσοδος:
LRForm: μια λίστα ορισμάτων, που θα βρίσκονται είτε στο αριστερό (LeftForm) είτε στο δεξί μέρος(RightForm) ενός θεωρήματος.
LeftToRightOrder: ένας ακέραιος αριθμός, που δηλώνει τη θέση (από αριστερά προς δεξιά) στην οποία βρίσκεται κάποιο όρισμα.
Έξοδος:
TheoremLiteral: το ανταποκρινόμενο όρισμα από τη λίστα ορισμάτων LRForm.
Στόχοι:
- get_literal_from_theorem([eq(plus(X,plus(Y,Z)),plus(plus(X,Y,Z)))], 1, TheoremLiteral), όπου TheoremLiteral = eq(plus(X,plus(Y,Z)),plus(plus(X,Y,Z)).
- get_literal_from_theorem([even(X),even(Y)], 2, TheoremLiteral), όπου TheoremLiteral = even(Y).

διαδικασία **get_literal_from_theorem**(LRForm, LeftToRightOrder, TheoremLiteral);
αρχή_διαδ
…
τέλος_διαδ

Αλγόριθμος 4.14 :*Βοηθητικός αλγόριθμος υλοποίησης για την εφαρμογή μετασχηματισμού.*

Η διαδικασία αυτή δέχεται ως είσοδο μια λίστα ορισμάτων LRForm και μια λίστα αριθμών LRorderList, οι οποίοι προσδιορίζουν τις θέσεις (εντός αυτής της λίστας) των ορισμάτων TheoremLiterals, που παίρνουμε στην έξοδο.

***Σημείωση**: Λεπτομέρειες για αυτή τη διαδικασία στο κώδικα.*

Είσοδος:

LRForm: μια λίστα ορισμάτων, που θα βρίσκονται είτε στο αριστερό (LeftForm) είτε στο δεξί μέρος (RightForm) ενός θεωρήματος.

LRorderList: μια λίστα με ακέραιους αριθμούς, οι οποίοι δηλώνουν τις θέσεις (από αριστερά προς δεξιά) στις οποίες βρίσκονται κάποια ορίσματα.

Έξοδος:

TheoremLiterals: το ανταποκρινόμενα ορίσματα από τη λίστα ορισμάτων LRForm.

Στόχοι:

- get_literals_from_theorem([true, even(X)], [1,2], TheoremLiterals), όπου TheoremLiterals = [true,even(X)].
- get_literals_from_theorem([true, even(X),even(Y)], [2,3], TheoremLiterals), όπου TheoremLiterals = [even(X),even(Y)].

διαδικασία **get_literals_from_theorem**(LRForm, LRorderList, TheoremLiterals);
αρχή_διαδ
…
τέλος_διαδ

Αλγόριθμος 4.15:*Βοηθητικός αλγόριθμος υλοποίησης για την εφαρμογή μετασχηματισμού.*

Η διαδικασία αυτή μετασχηματίζει τον όρο Term του ορίσματος με θέση N, σε ένα νέο όρο NewTerm (που παίρνουμε στην έξοδο), μετά την εφαρμογή του AxOrLemOrInd, με ταυτότητα AxLmIndId.

***Σημείωση**: Λεπτομέρειες για αυτή τη διαδικασία στο κώδικα.*

Είσοδος:

Term: το όρισμα του θεωρήματος που θα μετασχηματιστεί.

AxOrLemOrInd: μια από τις λέξεις axiom, lemma, induction.

AxLmIndId: ένας ακέραιος αριθμός, που είναι η ταυτότητα του αξιώματος, λήμματος, επαγωγής.

N: ένας ακέραιος αριθμός, που δηλώνει τη θέση (από αριστερά προς δεξιά) στην οποία βρίσκεται ο όρος του ορίσματος που επιθυμούμε να μετασχηματίσουμε.

Έξοδος:

NewTerm: το όρισμα που προκύπτει μετά την εφαρμογή του μετασχηματισμού.

Στόχοι:

- perform_apply(eq(plus(0,plus(Y,Z)),plus(plus(0,Y),Z)),axiom,1,[1],NewTerm), όπου NewTerm = eq(plus(Y,Z),plus(plus(0,Y),Z)).
- perform_apply([even(0),even(Y)],axiom,3,[1],NewTerm), όπου NewTerm = [true,even(Y)].

διαδικασία **perform_apply**(Term, AxOrLemOrInd, AxLmIndId, [N], NewTerm);
αρχή_διαδ
…
τέλος_διαδ

Αλγόριθμος 4.16:*Βοηθητικός αλγόριθμος υλοποίησης για την εφαρμογή μετασχηματισμού.*

Η διαδικασία αυτή δέχεται ως είσοδο μια λίστα ορισμάτων LRForm και έναν αριθμό LeftToRightOrder , ο οποίος προσδιορίζει τη θέση (εντός αυτής της λίστας) του ορίσματος που θα αντικατασταθεί με το όρισμα NewTheoremLiteral και θα δώσει τη νέα λίστα ορισμάτων NewLRForm, που παίρνουμε στην έξοδο.

Σημείωση*: Λεπτομέρειες για αυτή τη διαδικασία στο κώδικα.*

Είσοδος:
LRForm: μια λίστα ορισμάτων, που θα βρίσκονται είτε στο αριστερό (LeftForm) είτε στο δεξί μέρος (RightForm) ενός θεωρήματος.
LeftToRightOrder: ένας ακέραιος αριθμός, που δηλώνει τη θέση (από αριστερά προς δεξιά) στην οποία βρίσκεται κάποιο όρισμα.
NewTheoremLiteral: η νέα μορφή κάποιου ορίσματος μετά τον μετασχηματισμό του.
Έξοδος:
NewLRForm: η καινούργια λίστα των ορισμάτων που προκύπτει και αντικαθιστά την LRForm.
Στόχοι:
- put_newLiteral_to_LRtheorForm([eq(plus(X,plus(Y,Z)),plus(plus(X,Y,Z)))], 1, eq(0,plus(Y,Z)),plus(plus(X,Y,Z))),NewLRForm),
  όπου NewLRForm = [eq(plus(0,plus(Y,Z)),plus(plus(X,Y,Z)))].
- put_newLiteral_to_LRtheorForm([even(0),even(Y)], 1, true, NewLRForm),
  όπου NewLRForm = [true,even(Y)].

διαδικασία **put_newLiteral_to_LRtheorForm**(LRForm, LeftToRightOrder, NewTheoremLiteral, NewLRForm);
αρχή_διαδ
…
τέλος_διαδ

Αλγόριθμος 4.17:*Βοηθητικός αλγόριθμος υλοποίησης για την εφαρμογή μετασχηματισμού.*

Η διαδικασία αυτή δέχεται ως είσοδο μια λίστα ορισμάτων LRForm και μια λίστα αριθμών LRorderList, οι οποίοι προσδιορίζουν τις θέσεις (εντός αυτής της λίστας) των ορισμάτων που θα αντικατασταθούν με τα ορίσματα NewTheoLiterals και θα δώσει τη νέα λίστα ορισμάτων NewLRForm, που παίρνουμε στην έξοδο.

Σημείωση*: Λεπτομέρειες για αυτή τη διαδικασία στο κώδικα.*

Είσοδος:

LRForm: μια λίστα ορισμάτων, που θα βρίσκονται είτε στο αριστερό (LeftForm) είτε στο δεξί μέρος (RightForm) ενός θεωρήματος.

LRorderList: μια λίστα με ακέραιους αριθμούς, οι οποίοι δηλώνουν τις θέσεις (από αριστερά προς δεξιά) στις οποίες βρίσκονται κάποια ορίσματα.

NewTheoLiterals: η νέα μορφή κάποιων ορισμάτων μετά τον μετασχηματισμό τους.

Έξοδος:

NewLRForm: η καινούργια λίστα των ορισμάτων που προκύπτει και αντικαθιστά την LRForm.

Στόχοι:

- put_newTheoLiterals_to_LRtheoForm([true,even(Y)], [1,2], [even(X)], NewLRform), όπου NewLRForm = [even(X)].
- put_newTheoLiterals_to_LRtheoForm([true,even(Y),even(D)], [1,2], [even(X)], NewLRform), όπου NewLRForm = [even (D), even (X)].

διαδικασία **put_newTheoLiterals_to_LRtheoForm** (LRForm, LRorderList, NewTheoLiterals, NewLRForm);

αρχή_διαδ

…

τέλος_διαδ

Αλγόριθμος 4.18:*Βοηθητικός αλγόριθμος υλοποίησης για την εφαρμογή μετασχηματισμού.*

Η διαδικασία αυτή δέχεται ως είσοδο μια λίστα ορισμάτων TheoremLiterals και τον προσδιορισμό κάποιου μετασχηματισμού που θα εφαρμόσει .Ο μετασχηματισμός αυτός ορίζεται από το AxOrLemOrInd και ένα ακέραιο αριθμό AxLmIndId, που είναι η ταυτότητα του νόμου αυτού. Μετά τον μετασχηματισμό παίρνουμε στην έξοδο τη νέα λίστα ορισμάτων NewTheoremLiterals.

Σημείωση*: Λεπτομέρειες για αυτή τη διαδικασία στο κώδικα.*

Είσοδος:

TheoremLiterals: τα ορίσματα του θεωρήματος που θα μετασχηματιστούν.

AxOrLemOrInd: μια από τις λέξεις axiom, lemma, induction.

AxLmIndId: ένας ακέραιος αριθμός, που είναι η ταυτότητα του αξιώματος, λήμματος, επαγωγής.

N: ένας ακέραιος αριθμός, που δηλώνει τη θέση (από αριστερά προς δεξιά) στην οποία βρίσκεται ο όρος του ορίσματος που επιθυμούμε να μετασχηματίσουμε.

Έξοδος:

NewTheoremLiterals: τα ορίσματα που προκύπτουν μετά την εφαρμογή του μετασχηματισμού.

Στόχοι:

- apply_axiomToLiterals([even(X),even(Y)],induction,2,NewTheoremLiterals), όπου NewTheoremLiterals = [even(plus(X,Y))].
- apply_axiomToLiterals([true, even(X)],fol_law, 1, NewTheoremLiterals), όπου NewTheoremLiterals = [even(X)].

διαδικασία **apply_axiomToLiterals**(TheoremLiterals,AxOrLemOrInd,AxLmIndId,
NewTheoremLiterals);

αρχή_διαδ

…

τέλος_διαδ

Αλγόριθμος 4.19:*Βοηθητικός αλγόριθμος υλοποίησης για την εφαρμογή μετασχηματισμού.*

Η διαδικασία αυτή δέχεται ως είσοδο δύο λίστες ορισμάτων TheoremLLiterals και TheoremRLiterals καθώς και τον προσδιορισμό κάποιου μετασχηματισμού που θα εφαρμόσει.Ο μετασχηματισμός αυτός ορίζεται από το AxOrLemOrInd και ένα ακέραιο αριθμό AxLmIndId, που είναι η ταυτότητα του νόμου αυτού. Μετά τον μετασχηματισμό παίρνουμε στην έξοδο τη νέα λίστα ορισμάτων NewTheoremLiterals, που μπορεί να είναι και άδεια. ***Σημείωση****: Λεπτομέρειες για αυτή τη διαδικασία στο κώδικα.*

Είσοδος:

TheoremLLiterals: τα ορίσματα που βρίσκονται στο αριστερό μέρος του θεωρήματος και θα μετασχηματιστούν.

TheoremRLiterals: τα ορίσματα που βρίσκονται στο δεξί μέρος του θεωρήματος και θα μετασχηματιστούν.

AxOrLemOrInd: μια από τις λέξεις axiom, lemma, induction.

AxLmIndId: ένας ακέραιος αριθμός, που είναι η ταυτότητα του αξιώματος, λήμματος, επαγωγής.

Έξοδος:

NewTheoremLiterals: μια λίστα, πιθανόν άδεια, με ορίσματα που προκύπτουν μετά την εφαρμογή του μετασχηματισμού.

Στόχος:

apply_axiom_to_theorem([false],[X],fol_law,3, NewTheoremLiterals), όπου NewTheoremLiterals = [true].

διαδικασία **apply_axiom_to_theorem** (TheoremLLiterals, TheoremRLiterals, AxOrLemOrInd, AxLmIndId, NewTheoremLiterals);

αρχή_διαδ

…

τέλος_διαδ

Αλγόριθμος 4.20:*Βοηθητικός αλγόριθμος υλοποίησης για την εφαρμογή μετασχηματισμού.*

4.3.3 Αλγόριθμος Μετασχηματισμού Θεωρήματος με Εφαρμογή Νόμων Λογικής Πρώτης Τάξης και Αξιωμάτων Θεωρίας Ισότητας

Η διαδικασία αυτή δέχεται σαν είσοδο τον μετασχηματισμό Transf ενώ δεν επιστρέφει έξοδο. Η ***symbolic_evaluation*** *(Transf) ξεκινάει το μετασχηματισμό του τελευταίου θεωρήματος εφαρμόζοντας νόμους της λογικής πρώτης τάξης και συντονίζει τις κλήσεις των διαφόρων τμημάτων του υποσυστήματος αυτού.*

Είσοδος:
Transf: ο μετασχηματισμός που θα εφαρμόσουμε.
Έξοδος:-
Στόχος:
symbolic_evaluation(apply(fol_law,2,[left,[1,2]])).

διαδικασία **symbolic_evaluation** (Transf);
αρχή_διαδ
διάβασε LastThId;
perform_transf_apply (LastThId, Transf, New_LastThId);
τέλος_διαδ

Αλγόριθμος 4.21*:Αλγόριθμος υλοποίησης για τους μετασχηματισμούς θεωρημάτων με εφαρμογή νόμων λογικής πρώτης τάξης.*

Η διαδικασία αυτή δέχεται είσοδο τον όρο Atom&Term, ενώ δεν επιστρέφει έξοδο. Η ***symbolic_evaluation*** *(Atom&Term) εφαρμόζει το πρώτο αξίωμα από τη θεωρία της ισότητας (∀X ισχύει X = X).*

Είσοδος:
Atom&Term: ο όρος Term, που βρίσκεται στο άτομο Atom και στον οποίο θα εφαρμόσουμε το αξίωμα από τη θεωρία της ισότητας.
Έξοδος:-
Στόχος:
symbolic_evaluation([[eq,2,left,1],[]]).

διαδικασία **symbolic_evaluation** (Atom&Term);
αρχή_διαδ
διάβασε LastThId;
symb_eval(LRFormTh,LeftToRightOrder,NewLeftFormTh,NewRightFormTh);
δημιούργησε το καινούργιο θεώρημα **New_LastThId**;
αντικατέστησε το **LastThId** με το **New_LastThId**;
καταχώρησε το νέο θεώρημα, μετά την εφαρμογή του μετασχηματισμού;
δημιούργησε το **ProofTransformation;**
proofstep(ProofStepId, Id, ProofTransformation, New_LastThId);
% Δημιούργησε το βήμα απόδειξης για τον συγκεκριμένο μετασχηματισμό.
τέλος_διαδ

Αλγόριθμος 4.22:*Αλγόριθμος υλοποίησης για τους μετασχηματισμούς θεωρημάτων με εφαρμογή αξιωμάτων θεωρίας ισότητας.*

Η διαδικασία αυτή δέχεται ως είσοδο δύο λίστες LRFormTh και LeftToRightOrder, που είναι αντίστοιχα τα ορίσματα, τα οποία βρίσκονται στο αριστερό ή το δεξί μέρος του θεωρήματος και θα μετασχηματιστούν και ένας ακέραιος αριθμός, που δηλώνει τη θέση (από αριστερά προς δεξιά) στην οποία βρίσκεται κάποιο όρισμα *Το αποτέλεσμα αυτού του μετασχηματισμού θα δοθεί στη λίστα NewLeftFormTh ή την λίστα NewRightFormTh και αν αυτό το αποτέλεσμα είναι [true],τότε και η άλλη λίστα θα πάρει την ίδια τιμή.*
Σημείωση*: Λεπτομέρειες για αυτή τη διαδικασία στο κώδικα.*

Είσοδος:

LRFormTh: τα ορίσματα που βρίσκονται στο αριστερό ή το δεξί μέρος του θεωρήματος και θα μετασχηματιστούν.

LeftToRightOrder: ένας ακέραιος αριθμός, που δηλώνει τη θέση (από αριστερά προς δεξιά) στην οποία βρίσκεται κάποιο όρισμα.

Έξοδος:

NewLeftFormTh: μια λίστα, πιθανόν άδεια, με ορίσματα που προκύπτουν μετά την εφαρμογή του μετασχηματισμού. Όταν το NewRightFormTh γίνει [true], που σημαίνει ότι η απόδειξη έχει ολοκληρωθεί, τότε και το NewLeftFormTh θα γίνει [true].

NewRightFormTh: μια λίστα, πιθανόν άδεια, με ορίσματα που προκύπτουν μετά την εφαρμογή του μετασχηματισμού. Όταν το NewLeftFormTh γίνει [true], που σημαίνει ότι η απόδειξη έχει ολοκληρωθεί, τότε και το NewRightFormTh θα γίνει [true].

Στόχος:

symb_eval([eq(plus(A,B), plus(A,B))], [1], NewLeftFormTh, NewRightFormTh), όπου NewLeftFormTh =true συνεπώς και NewRightFormTh = true.

διαδικασία
symb_eval(LRFormTh,LeftToRightOrder,NewLeftFormTh,NewRightFormTh);

αρχή_διαδ
…
τέλος_διαδ

Αλγόριθμος 4.23:*Βοηθητικός αλγόριθμος υλοποίησης για τους μετασχηματισμούς θεωρημάτων με εφαρμογή αξιωμάτων θεωρίας ισότητας.*

4.4 Διεπικοινωνία συστήματος/ System interface

4.4.1 Εισαγωγή

Η Visual Basic είναι μια γλώσσα προγραμματισμού ευρέως χρησιμοποιούμενη για ανάπτυξη κυρίως διεπικοινωνίας υπολογιστικών συστημάτων. Έχει αναπτυχθεί από την Microsoft Corporation και βασίσθηκε στην γλώσσα προγραμματισμού Basic (QBasic). Η πρώτη έκδοση της Visual Basic αναπτύχθηκε το 1990 και από τότε έχουν αναπτυχθεί ανάλογα πακέτα εφαρμογών σε άλλες γλώσσες προγραμματισμού όπως η C, C++, Pascal και Java.

Η Visual Basic είναι ένα Ολοκληρωμένο Περιβάλλον Ανάπτυξης, με το οποίο μπορούν να αναπτυχθούν, να εκτελεστούν, να δοκιμαστούν και να διορθωθούν εφαρμογές (προγράμματα). Η Visual Basic ήταν μια από της πρώτες γλώσσες προγραμματισμού που προσέφεραν τη δυνατότητα δημιουργίας γραφικού περιβάλλοντος και περιβάλλοντος διεπικοινωνίας μεταξύ χρήστη και προγράμματος (user interface).

Με τη Visual Basic μπορούμε εύκολα να δημιουργήσουμε προγράμματα τα οποία υποστηρίζουν παραθυρικό περιβάλλον όπως είναι οι εφαρμογές των Windows. Με τις βιβλιοθήκες της Microsoft (Visual Basic for Applications - VBA) μπορούμε να επεξεργαστούμε πολλές εφαρμογές για Windows και Mac OS όπως είναι η επικοινωνία με τα Microsoft Word και Excel. Οι εφαρμογές των μακροεντολών μπορούν να αυτοματοποιήσουν ή να μεταβάλλουν τις λειτουργίες πολλών προγραμμάτων. Τέλος με την Visual Basic μπορούμε να κατασκευάσουμε εφαρμογές διαδικτύου.

Οι κατασκευαστές της Sicstus Prolog έχουν αναπτύξει μια εφαρμογή διεπικοινωνίας μεταξύ των δύο γλωσσών προγραμματισμού (Visual Basic – Sicstus Prolog). Η εφαρμογή αυτή είναι μια εύχρηστη μονόδρομη διεπικοινωνία με την Visual Basic. Μας επιτρέπει να φορτώνουμε και να καλούμε προγράμματα κατασκευασμένα σε Sicstus Prolog από την Visual Basic αλλά όχι το αντίθετο (από Prolog σε Visual Basic). Η Visual Basic χρησιμοποιείται για να δημιουργήσει το περιβάλλον εργασίας του χρήστη (διεπικοινωνία Χρήστη - Συστήματος). Η Prolog χρησιμοποιείται για την υλοποίηση της Βάσης Γνώσεων και των μηχανισμών συλλογισμού του συστήματος. Δηλαδή, το ευφυές τμήμα του συστήματος υλοποιείται σε Prolog.

Η διεπικοινωνία προσφέρει τις εξής λειτουργίες:

- Δίνει στην Prolog μια ερώτηση για επεξεργασία.
- Λαμβάνει τα αποτελέσματα (συμβολοσειρές (strings) ή αριθμοί (integer)) τα οποία αντιστοιχίζονται σε συγκεκριμένες μεταβλητές της ερώτησης της Prolog.
- Λαμβάνει τις πληροφορίες σχετικά με τις εξαιρέσεις (ανεπιτυχής επεξεργασία της ερώτησης) που μπορεί να προκύψουν από την ερώτηση Prolog.

4.4.2 Διεπικοινωνία Visual Basic – Prolog

Με τον όρο ***διεπικοινωνία*** εννοούμε την διαδικασία ανταλλαγής πληροφοριών μεταξύ δυο συστημάτων. Διεπικοινωνία μπορεί να έχουμε μεταξύ δυο υπολογιστών που ανταλλάσσουν πληροφορίες (επικοινωνούν) μεταξύ τους με την βοήθεια κάποιου ειδικού αλγορίθμου. Επίσης διεπικοινωνία μπορεί να έχουμε μεταξύ διαφορετικών εξαρτημάτων ή προγραμμάτων στο ίδιο υπολογιστικό σύστημα.

Στο σύστημα ***Ευκλείδης*** που κατασκευάσαμε χρησιμοποιούμε δυο γλώσσες προγραμματισμού διαφορετικής τεχνολογίας, την ***Prolog*** και την ***Visual Basic***. Ο λόγος χρήσης της διεπικοινωνίας είναι να μπορέσουμε να εκμεταλλευτούμε τις δυνατότητες των δυο γλωσσών για να δημιουργήσουμε ένα αποτελεσματικό και εύκολο στη χρήση του σύστημα. Η διεπικοινωνία του συστήματος έχει υλοποιηθεί σε Visual Basic ενώ η αναπαράσταση γνώσεων και οι συλλογιστικοί μηχανισμοί έχουν υλοποιηθεί σε Prolog.

Για την υλοποίηση της επικοινωνίας των δυο γλωσσών είναι απαραίτητη η εγκατάσταση της ***Sicstus Prolog*** (έκδοση 3.11 ή μεγαλύτερη) και της ***Visual Basic*** στον υπολογιστή που εκτελείται το πρόγραμμα. Για την σωστή λειτουργία της επικοινωνίας τα παρακάτω αρχεία θα πρέπει να εμπεριέχονται στο φάκελο που είναι εγκατεστημένο το σύστημα μας:

1. Vbsp.dll
2. Vbsp.bas
3. Sprt311.dll
4. Sprt.sav

Τα αρχεία ***Vbsp.dll, Sprt311.dll*** και ***Sprt.sav*** βρίσκονται στον φάκελο ***bin*** της γλώσσας προγραμματισμού Prolog. Το αρχείο ***Vbsp.bas*** βρίσκεται στον φάκελο ***library****vbsp*** και απαιτείται μόνο όταν εκτελείται ο κώδικας από τη Visual Basic (Source Code)**.**

Όλα τα παραπάνω αρχεία υπάρχουν στο σύστημα που έχουμε κατασκευάσει και δεν χρειάζεται η αντιγραφή τους από τον χρήστη.

Το εκτελέσιμο πρόγραμμα διεπικοινωνίας (αρχείο .exe) μπορεί να τρέξει σε οποιοδήποτε σύστημα με εγκατεστημένο λειτουργικό τα Windows και δεν απαιτείται η εγκατάσταση της Prolog ή της Visual Basic στο σύστημα αυτό.

Η Visual Basic αναλαμβάνει να φορτώσει το πρόγραμμα που θα μετασχηματιστεί καθώς και τη βάση γνώσεων που έχουν υλοποιηθεί σε Prolog. Η εντολή έναρξης της διαδικασίας μετασχηματισμού δίνεται από την Visual Basic και όταν η Prolog εντοπίσει κάποια αποτελέσματα, αυτά επιστρέφουν στην Visual Basic. Λόγω του τρόπου κατασκευής της διεπικοινωνίας πρέπει η περιοχή "***directory***" και το αρχείο "***file name***" που έχει τοποθετηθεί το πρόγραμμα να είναι με αγγλικούς χαρακτήρες. Για παράδειγμα, το μονοπάτι του προγράμματος με όνομα φακέλου ***Final_Code*** μπορεί να είναι ένα από τα ***C:\Program Files\Final_Code*** ή ***G:\Final_Code***. Ενώ ***δεν μπορεί*** να είναι ***C:\Τα προγράμματα μου\Τελικός_Κώδικας*.** Το πρόβλημα αυτό οφείλεται στην υλοποίηση της Sicstus Prolog η οποία δεν αναγνωρίζει Ελληνικούς χαρακτήρες.

Η διεπικοινωνία της Visual Basic με την Prolog γίνεται μέσω των παρακάτω συναρτήσεων και διαδικασιών της Visual Basic. Στόχοι της Prolog μπορούν να

κληθούν μέσα από την Visual Basic. Το αντίθετο, δηλαδή κλήση συναρτήσεων ή διαδικασιών της Visual Basic από την Prolog δεν υποστηρίζεται από την έκδοση αυτή (Sicstus Prolog 3.11) [SICStus Prolog User's Manual].

Function PrologOpenQuery(ByVal Goal As String) As Long

Με αυτή την συνάρτηση δηλώνεται μέσω της Visual Basic στην Prolog ο στόχος (ερώτηση) Prolog που θέλουμε να εκτελεστεί. Η συνάρτηση αυτή επιστρέφει στην Visual Basic ένα αριθμό ταυτότητα (query identifier "qid") που αντιστοιχεί στην ερώτηση «Goal». Αν η τιμή που επιστρέφει η συνάρτηση είναι -1 (qid = -1) τότε υπάρχει σφάλμα στην σύνταξη του στόχου (της ερώτησης) Prolog. Όταν ο στόχος(ερώτηση) είναι σωστός η συνάρτηση επιστρέφει μια τιμή η οποία είναι μοναδικός ακέραιος αριθμός και αντιπροσωπεύει την ταυτότητα της ερώτησης.

Sub PrologCloseQuery(ByVal qid As Long)

Αυτή η διαδικασία μας επιτρέπει να δηλώσουμε τον τερματισμό ενός στόχου (ερώτησης) Prolog. Η PrologOpenQuery και η PrologCloseQuery χρησιμοποιούνται μαζί και μας επιτρέπουν να ξεκινάμε και να τελειώνουμε στόχους (ερωτήσεις) στην Prolog.

Function PrologNextSolution(ByVal qid As Long) As Integer

Αυτή η συνάρτηση επιστρέφει ένδειξη για επιτυχή εκτέλεση του στόχου (ερώτησης) της Prolog με ταυτότητα (κωδικό) ερώτησης τον qid. Επιστρέφει τιμή 1 όταν το αποτέλεσμα είναι επιτυχές, 0 όταν δεν υπάρχει αποτέλεσμα και -1 όταν υπάρχει λάθος στην ερώτηση ή στο πρόγραμμα Prolog.

Function PrologGetLong(ByVal qid As Long, ByVal VarName As String, Value As Long) As Integer

Με αυτή τη συνάρτηση μπορούμε να πάρουμε το αποτέλεσμα από την εκτέλεση μιας ερώτησης Prolog. Το αποτέλεσμα θα πρέπει να είναι κάποιος ακέραιος αριθμός (Long). Η εντολή PrologGetLong χρησιμοποιείται με την ταυτότητα (κωδικό) ερώτησης τον qid. Στην συνάρτηση δίνεται το όνομα της μεταβλητής Prolog "VarName" και το όνομα της μεταβλητής Visual Basic όπου θα επιστραφεί το αποτέλεσμα "Value". Η συνάρτηση επιστρέφει τιμή 1 (*PrologGetLong (qid, μεταβλητή_Prolog, μεταβλητή_Visual_Basic) = 1*) όταν το αποτέλεσμα δοθεί στο πρόγραμμα Visual Basic με επιτυχία. Όταν δεν υπάρχει αποτέλεσμα στην ερώτηση επιστρέφει την τιμή 0 ενώ όταν υπάρχει σφάλμα σε κάποιο σημείο του προγράμματος Prolog επιστρέφει τιμή -1.

Σημείωση*:* Long είναι ένας ακέραιος αριθμός 4 bytes ο οποίος παίρνει τιμές από -2,147,483,648 μέχρι + 2,147,483,648.

Function PrologGetString(ByVal qid As Long, Val VarName As String, Value As String) As Integer

Η συνάρτηση αυτή είναι αντίστοιχη με την συνάρτηση PrologGetLong. Η διάφορα της PrologGetString είναι ότι τα αποτελέσματα που παίρνουμε θα πρέπει να είναι χαρακτήρες (strings). Η συνάρτηση επιστρέφει τιμή 1 όταν το αποτέλεσμα δοθεί στο πρόγραμμα Visual Basic με επιτυχία. Όταν δεν υπάρχει αποτέλεσμα στην ερώτηση επιστρέφει τιμή 0. Τέλος, όταν υπάρχει σφάλμα σε κάποιο σημείο του προγράμματος Prolog επιστρέφει τιμή -1.

Σημείωση: Strings δηλαδή συμβολοσειρές, αποτελούνται από γράμματα, αριθμούς ή άλλα σύμβολα του πληκτρολογίου.

Function PrologGetStringQuoted(ByVal qid As Long, ByVal VarName As String, Value As String) As Integer

Η συνάρτηση αυτή είναι ίδια με την συνάρτηση PrologGetString. Η διαφορά τους είναι ότι η συνάρτηση PrologGetStringQuoted χρησιμοποιεί το προδεδηλωμένο κατηγόρημα **Prolog writeq/2**.

Function PrologQueryCutFail(ByVal Goal As String) As Integer

Με αυτή τη συνάρτηση εισάγεται στην Prolog το πρόγραμμα που θέλουμε να εκτελέσουμε μέσω της Visual Basic. Η συνάρτηση επιστρέφει τιμή 1 στην επιτυχή εισαγωγή του προγράμματος, τιμή 0 στην ανεπιτυχή εισαγωγή και τιμή -1 σε περίπτωση σφάλματος στο πρόγραμμα.

Sub PrologGetException(ByRef Exc As String)

Αυτή η διαδικασία υποστηρίζει τον χειρισμό των εξαιρέσεων. Ο όρος εξαίρεσης επιστρέφεται σαν συμβολοσειρά. Εάν δεν υπάρχει εξαίρεση επιστρέφει την άδεια συμβολοσειρά.

Function PrologInit() As Long

Με αυτή τη συνάρτηση ξεκινά η επικοινωνία μεταξύ Visual Basic και Prolog. Επιστρέφει 1 στην επιτυχή έναρξη της επικοινωνίας και 0 στην περίπτωση σφάλματος.

Function PrologDeInit() As Long

Με αυτή τη συνάρτηση σταματά η επικοινωνία μεταξύ Visual Basic και Prolog. Επιστρέφει 1 όταν η επικοινωνία διακοπεί επιτυχώς και -1 στην περίπτωση σφάλματος.

4.4.3 Περιγραφή Διεπικοινωνίας του συστήματος Ευκλείδης

Όπως αναφέραμε παραπάνω ο χρήστης του συστήματος επικοινωνεί με αυτό μέσω παραθυρικού περιβάλλοντος, το οποίο έχει υλοποιηθεί σε γλώσσα Visual Basic. Η διεπικοινωνία του συστήματος εκτελεί τις παρακάτω λειτουργίες διεπικοινωνίας :

1. Εκκινεί τη γλώσσα προγραμματισμού Prolog μέσω της Visual Basic.
2. Φορτώνει το σύστημα που υλοποιήθηκε σε Prolog.
3. Δίνει την δυνατότητα στον χρήστη να επιλέξει ένα αρχικό θεώρημα το οποίο και θα αποδείξει.
4. Μετασχηματίζει το αρχικό θεώρημα ώστε να επιτύχει την απόδειξή του.
5. Προβάλλει στην οθόνη το αρχικό θεώρημα καθώς και κάθε θεώρημα που προκύπτει από τους μετασχηματισμούς, όπως επίσης και το εκάστοτε βήμα απόδειξης.
6. Δίνει τη δυνατότητα στο χρήστη να προβάλλει στην οθόνη όλα τα θεωρήματα ή όλα τα βήματα απόδειξης.
7. Δίνει την δυνατότητα στον χρήστη να ακυρώσει το τελευταίο βήμα απόδειξης.

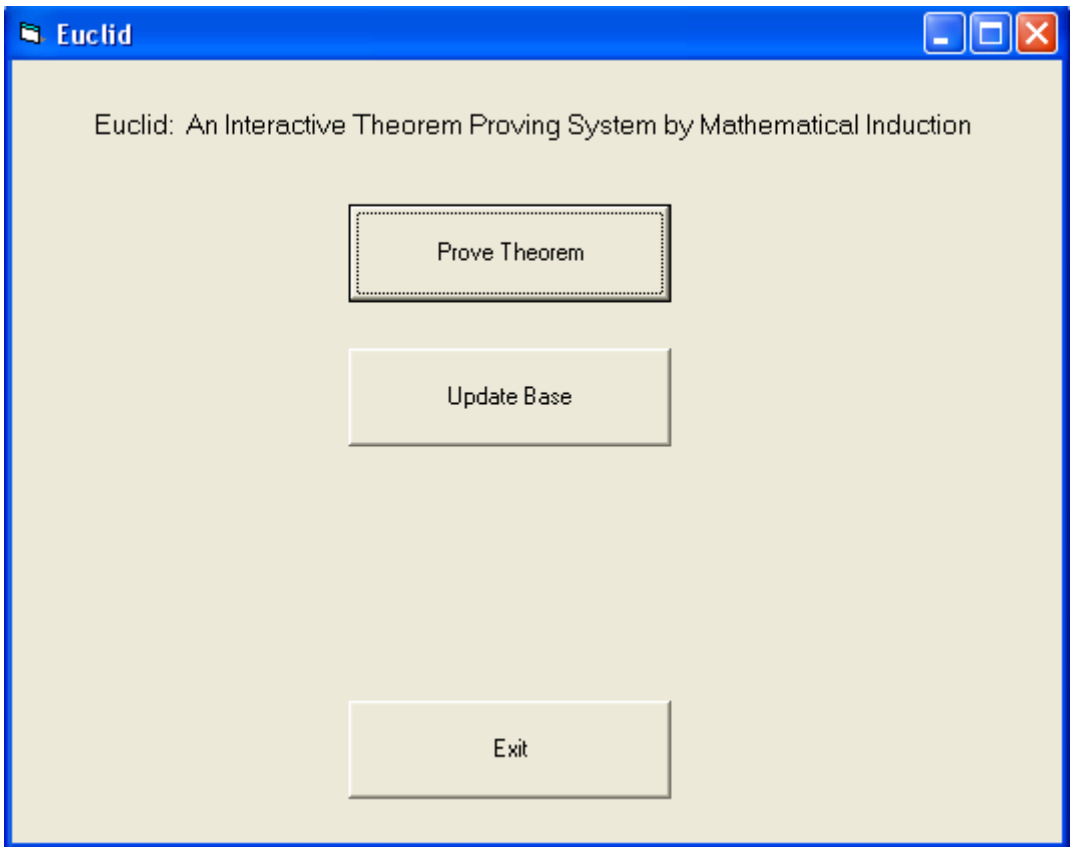
Παρακάτω θα αναπτύξουμε κάθε μια από τις λειτουργίες αυτές.

Αρχικά η Visual Basic εκκινεί την γλώσσα προγραμματισμού Prolog χρησιμοποιώντας την εντολή *PrologInit*. Αν η εκκίνηση πραγματοποιηθεί με επιτυχία τότε το σύστημα περνάει στην επόμενη διαδικασία διαφορετικά τερματίζει.

Στη συνέχεια η Visual Basic φορτώνει στην Prolog δύο βιβλιοθήκες, την terms και την lists και το τμήμα του Ευκλείδη που έχει υλοποιηθεί σε Prolog. Αυτό επιτυγχάνεται χρησιμοποιώντας την εντολή *PrologQueryCutFail*. Οι διαδικασίες αυτές είναι ανεξάρτητες από τις υπόλοιπες.

Μετά εμφανίζεται το κύριο παράθυρο της εικόνας 4.1 με τις παρακάτω δύο επιλογές.

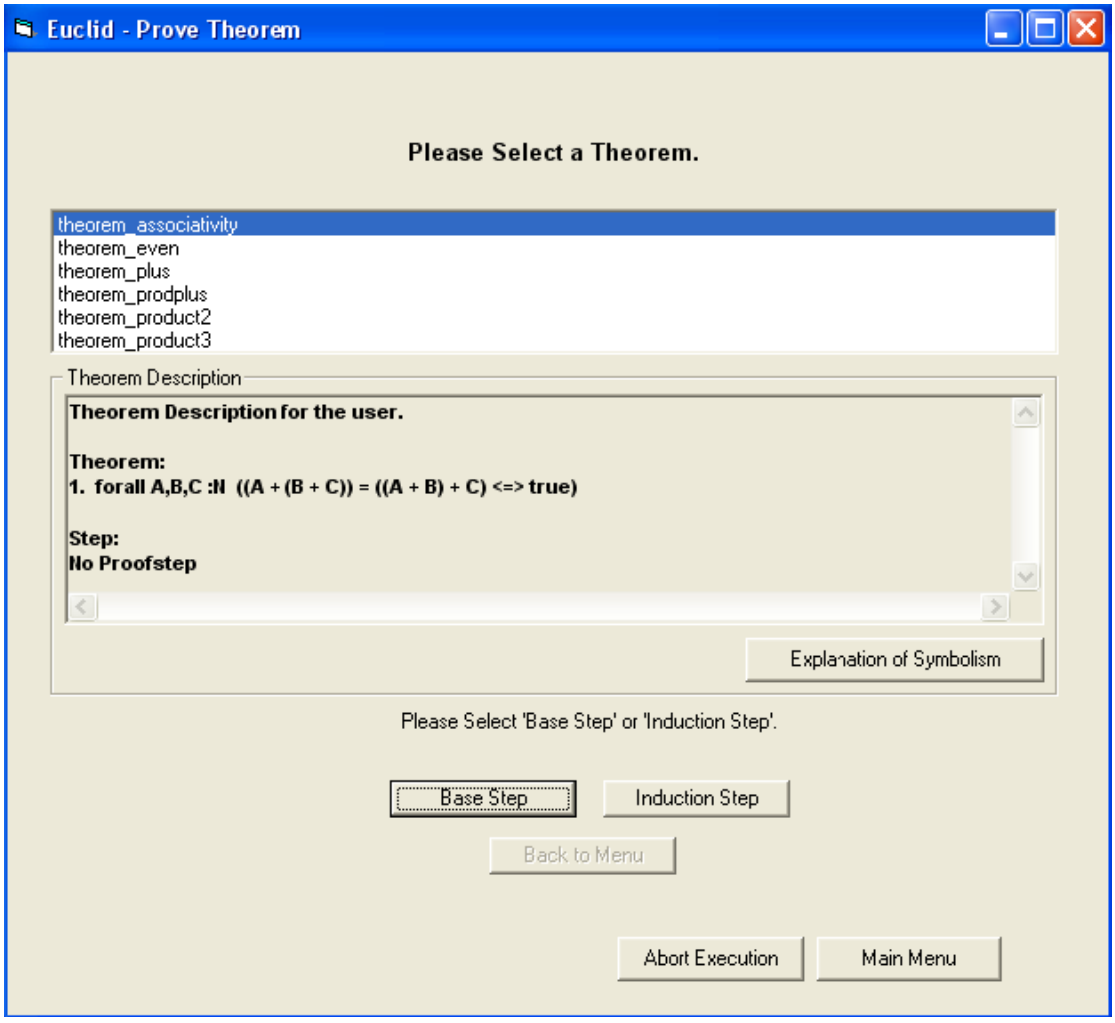
1. Επιλογή ενός θεωρήματος για απόδειξη.
2. Ενημέρωση της βάσης γνώσεως.



Εικόνα 4.1: *Κύριος κατάλογος επιλογών.*

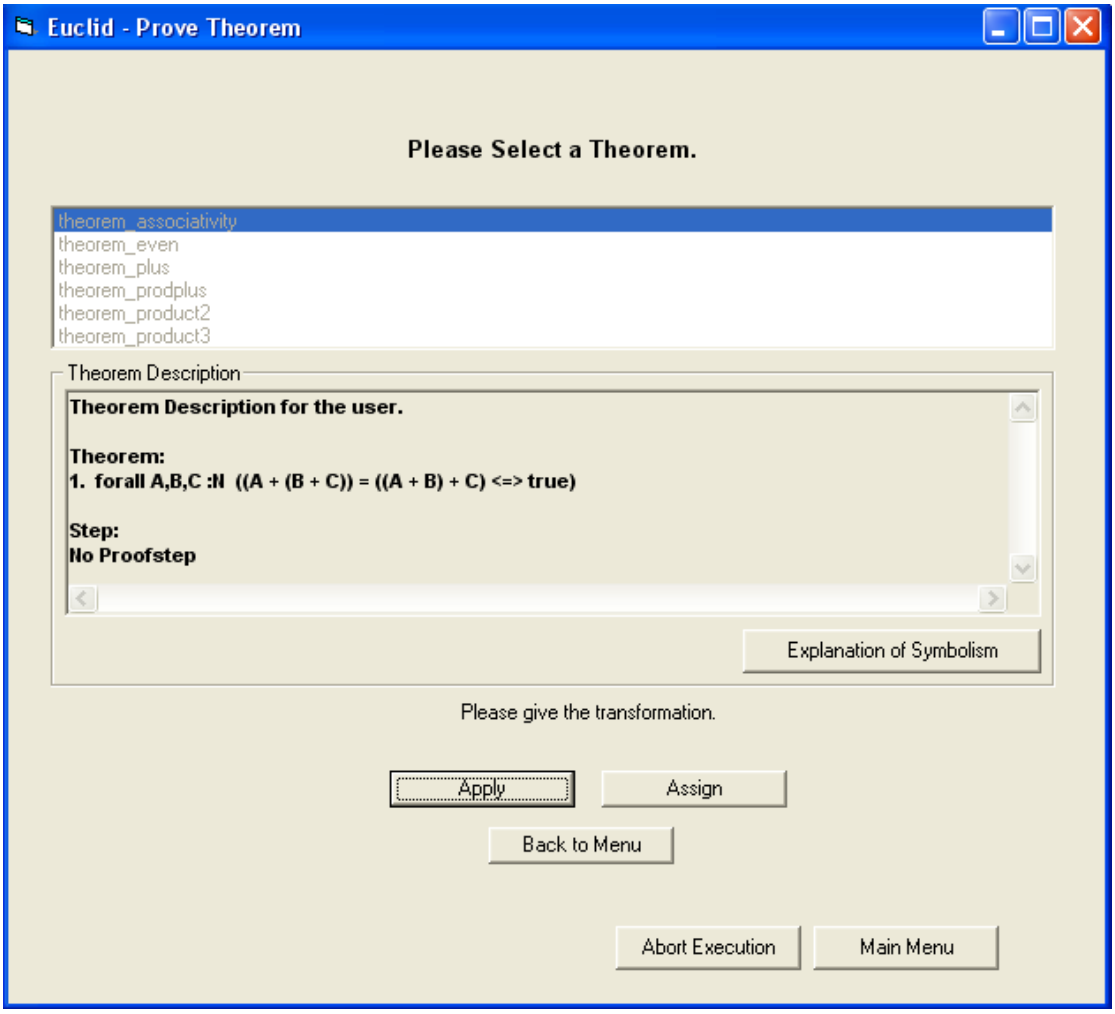
Επιλογή ενός θεωρήματος για απόδειξη /Select a theorem to be proved.

Εμφανίζεται ένα καινούργιο παράθυρο το οποίο περιέχει μια λίστα με όλα τα θεωρήματα, που μπορούν ν' αποδειχθούν, καθώς και την επιλογή για απόδειξη του βήματος βάσης ή του βήματος επαγωγής, εικόνα 4.2.



Εικόνα 4.2: *Αρχικά θεωρήματα προς απόδειξη και επιλογή βήματος βάσης και βήματος επαγωγής.*

Η περιγραφή του θεωρήματος που επιλέξαμε επιδεικνύεται για πλήρη γνώση του θεωρήματος, το οποίο θα αποδειχθεί στη συνέχεια. Αφού επιλέξουμε το θεώρημα που θέλουμε ν' αποδείξουμε και το βήμα βάσης (Base step) ή το βήμα επαγωγής (Induction step), εμφανίζεται ένα καινούργιο παράθυρο, το οποίο φαίνεται στην εικόνα 4.3.

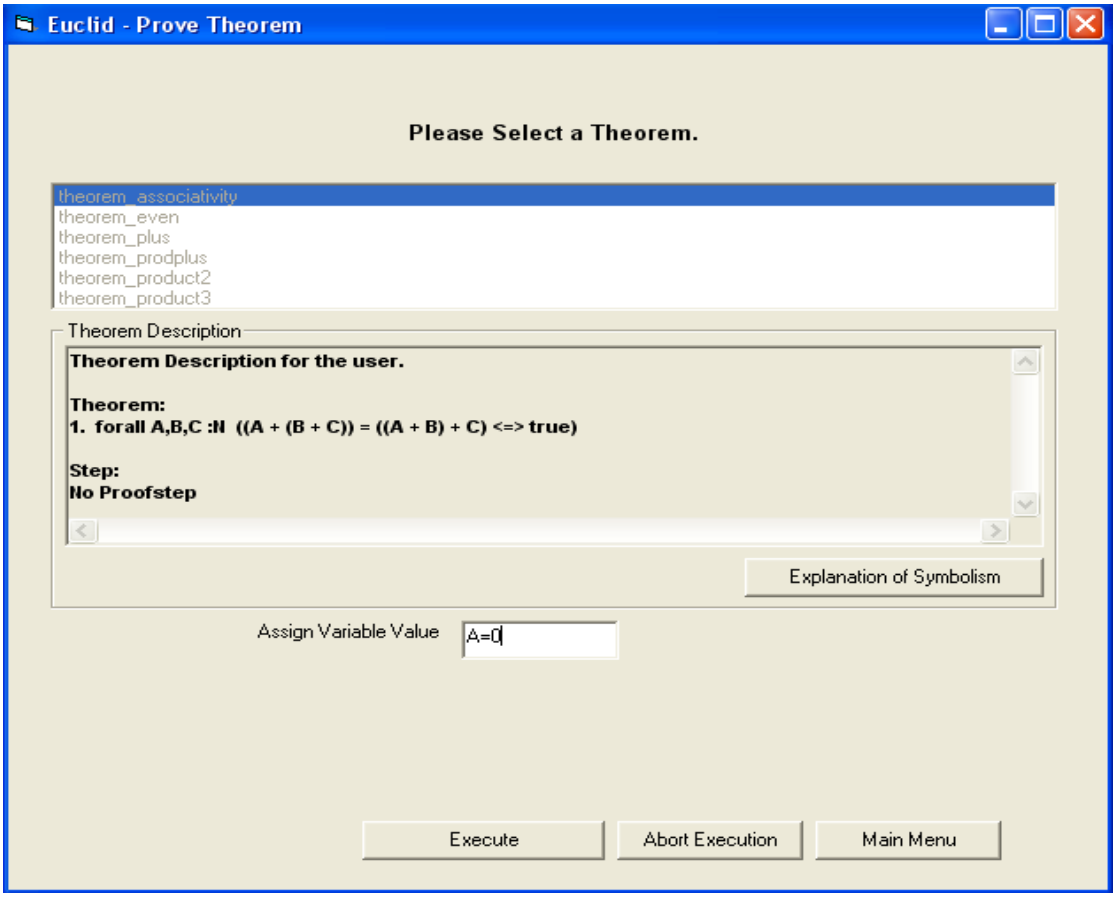


Εικόνα 4.3: *Μετασχηματισμός προγράμματος.*

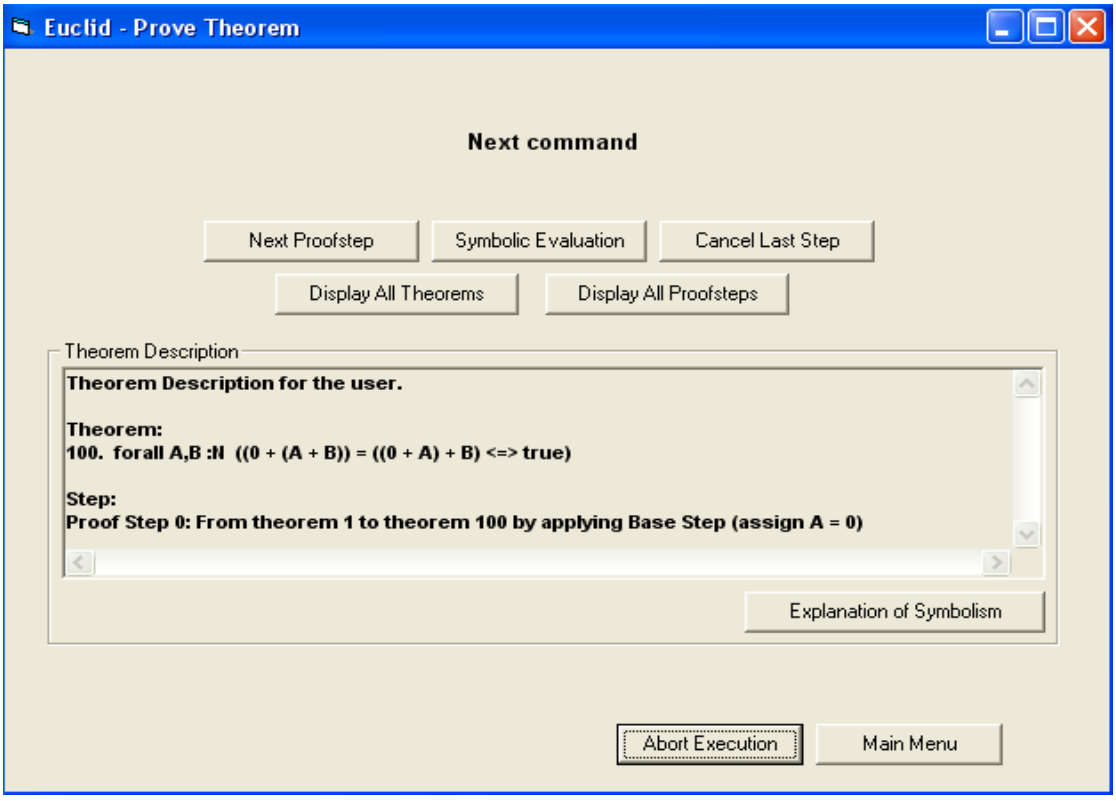
Επιλέγοντας το κουμπί "*assign*" στο παράθυρο 4.3, αρχίζει η διαδικασία απόδειξης της βάσης επαγωγής ή του βήματος επαγωγής για το θεώρημα που έχουμε επιλέξει. Το παράθυρο της εικόνας 4.4. ανοίγει για να ξεκινήσει η απόδειξη.

Στο κενό πλαίσιο που υπάρχει, δηλώνουμε το όνομα της μεταβλητής στην οποία επιθυμούμε να καταχωρήσουμε κάποια τιμή, καθώς και την τιμή αυτή με τη μορφή "*Μεταβλητή = Τιμή*". Η μεταβλητή αυτή ονομάζεται μεταβλητή επαγωγής. Επιλέγοντας το κουμπί "*Execute*", καταχωρείται η τιμή που θέλουμε στην μεταβλητή επαγωγής.

Όταν ένας μετασχηματισμός εφαρμοστεί με επιτυχία, το θεώρημα που έχει προκύψει προστίθεται στην λίστα των μετασχηματισμένων θεωρημάτων.



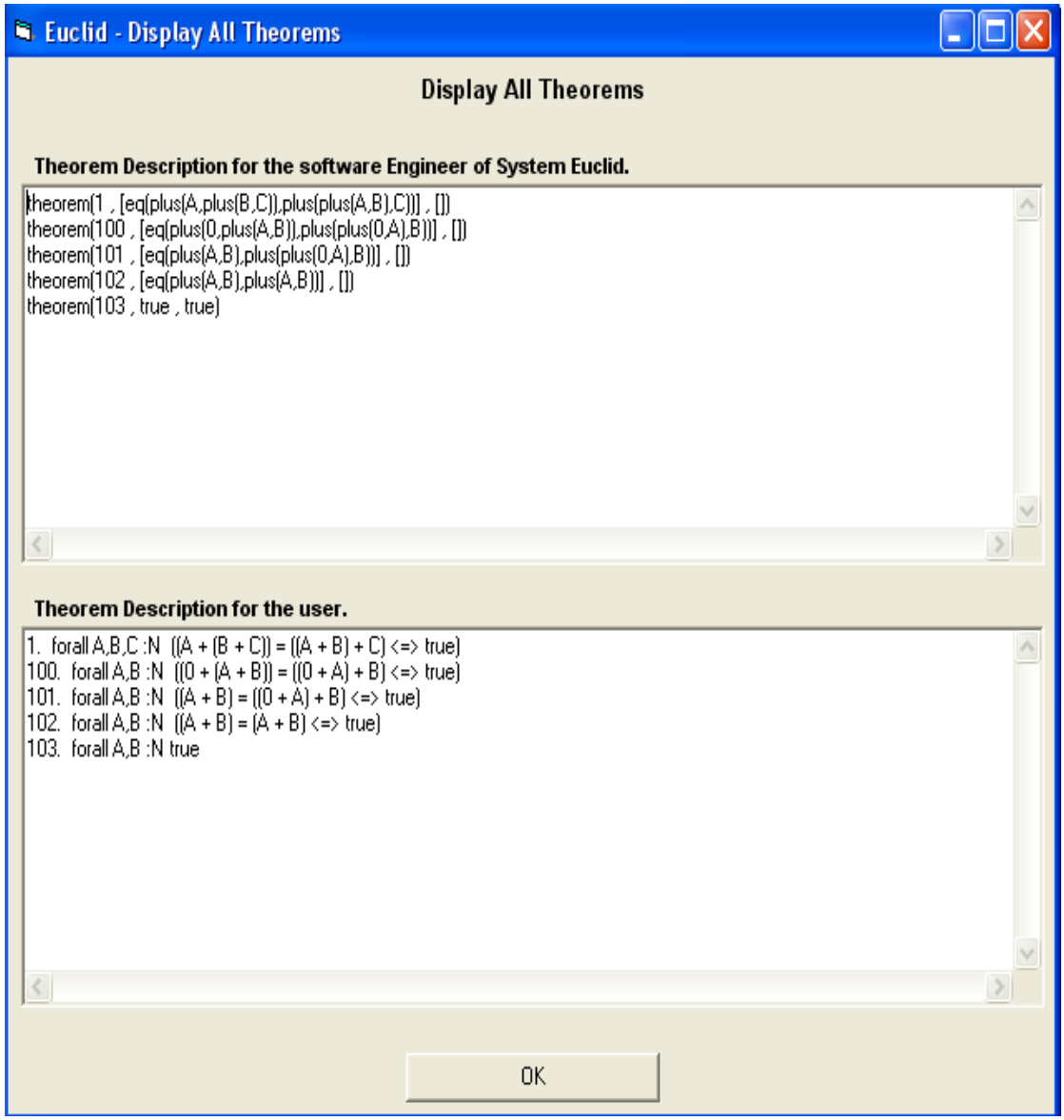
***Εικόνα 4.4**: Δήλωση της μεταβλητής επαγωγής και καταχώρηση τιμής σε αυτήν.*



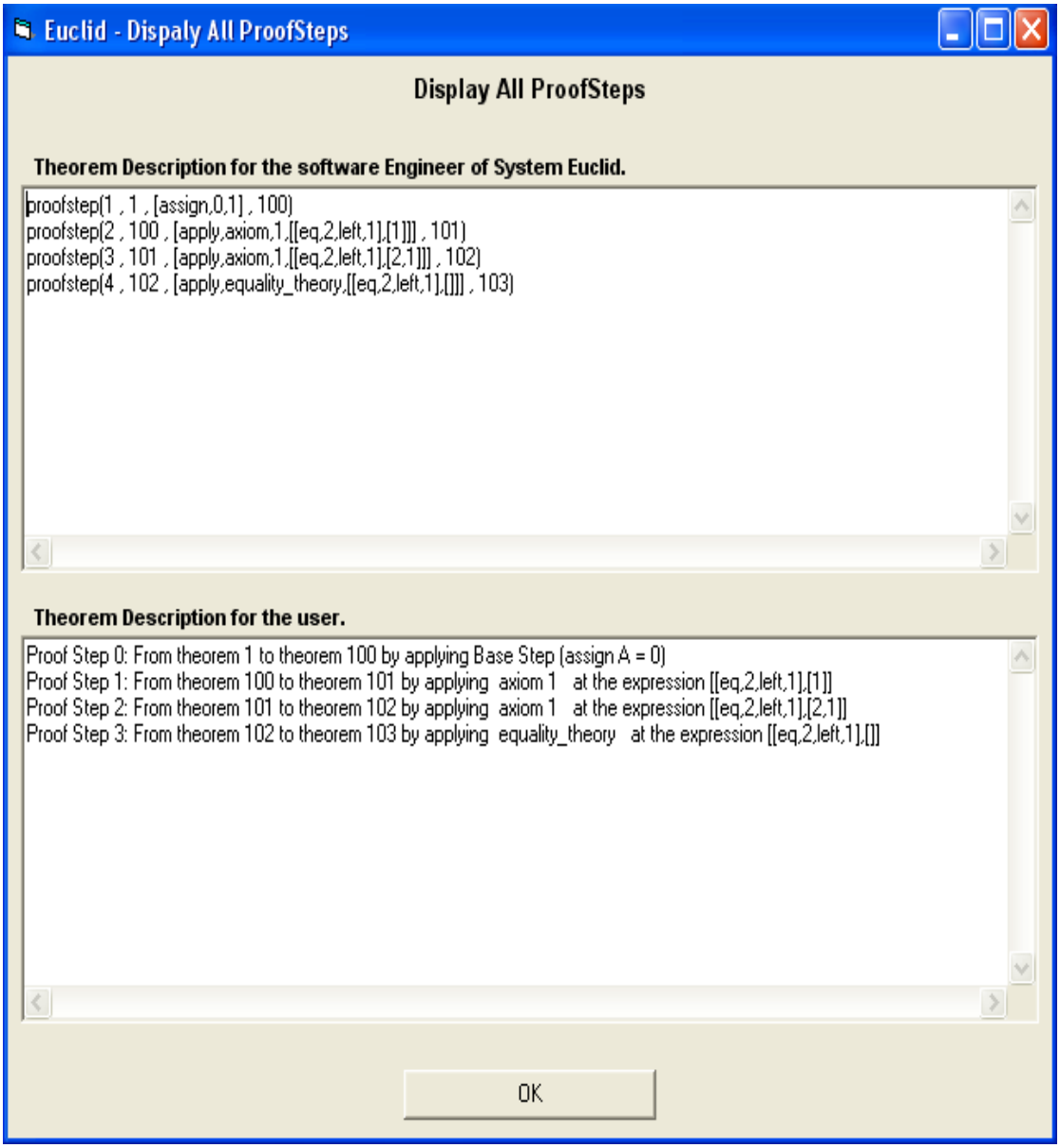
***Εικόνα 4.5:** Κατάλογος επιλογών μετά από κάποιο μετασχηματισμό.*

Στην εικόνα 4.5 εμφανίζεται ένας κατάλογος με τις επιλογές που έχουμε μετά από κάποιο μετασχηματισμό. Συγκεκριμένα με το κουμπί "Next Proofstep" συνεχίζουμε την απόδειξη εφαρμόζοντας κάποιον άλλο μετασχηματισμό. Όταν θέλουμε να εφαρμόσουμε κάποιο μετασχηματισμό χρησιμοποιώντας κυρίως αξιώματα από τη θεωρία της ισότητας πατάμε το κουμπί "Symbolic Evaluation" και συνεχίζουμε από εκεί. Για να ακυρώσουμε τον τελευταίο μετασχηματισμό πατάμε το κουμπί "Cancel Last Step". Όταν θέλουμε να δούμε όλα τα θεωρήματα ή όλα τα βήματα απόδειξης πατάμε αντίστοιχα τα κουμπιά "Display All Theorems" (εικόνα 4.6) και "Display All Proof steps" (εικόνα 4.7).

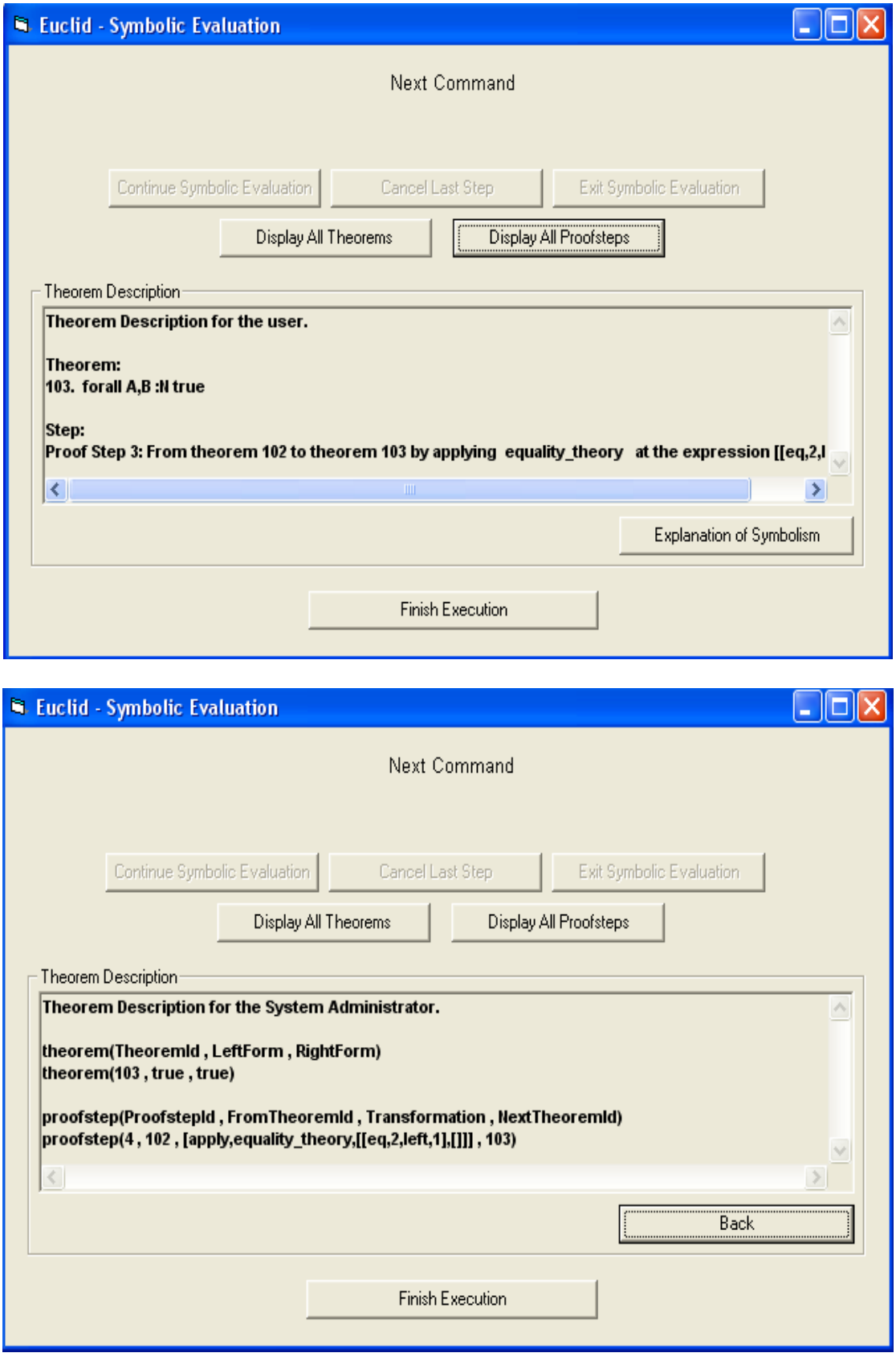
Σ' αυτό το σημείο αξίζει ν' αναφερθούμε και στο κουμπί "Explanation of Symbolism". Στο πλαίσιο πάνω από αυτό το κουμπί, βλέπουμε κάθε φορά το τελευταίο θεώρημα, καθώς και το τελευταίο βήμα απόδειξης σε μορφή κατανοητή για τον χρηστή. Πατώντας όμως το "Explanation of Symbolism" μπορούμε να δούμε ακριβώς τα ίδια στοιχεία στη μορφή που έχουν για τον διαχειριστή του συστήματος (εικόνα 4.8).



Εικόνα 4.6: *Εμφάνιση όλων των θεωρημάτων.*

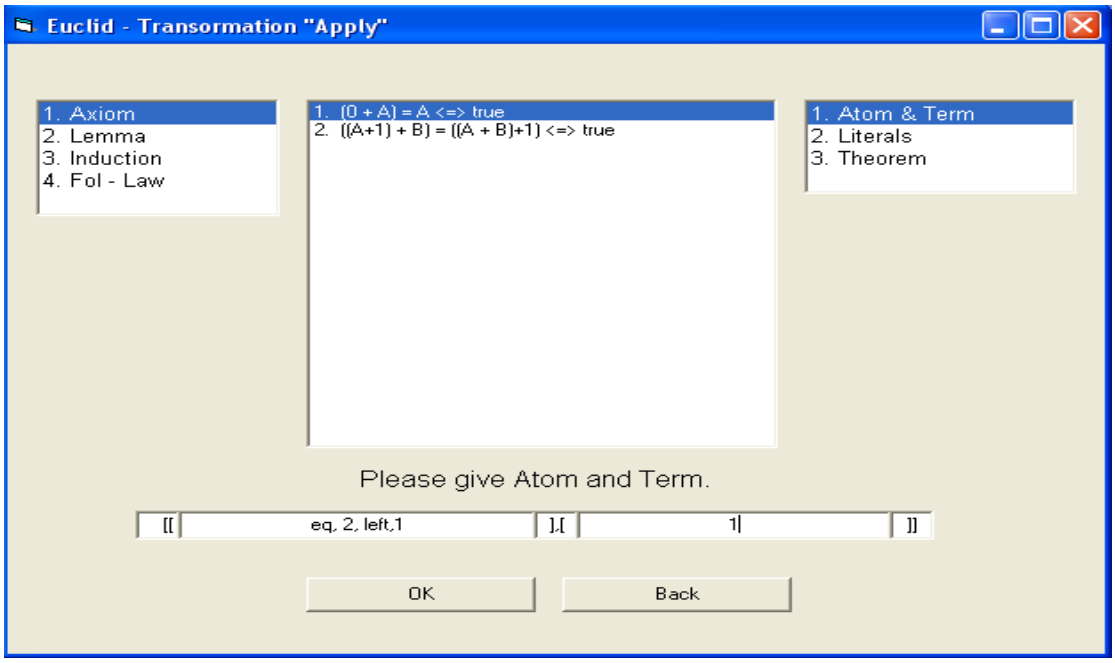


Εικόνα 4.7: *Εμφάνιση όλων των βημάτων απόδειξης.*

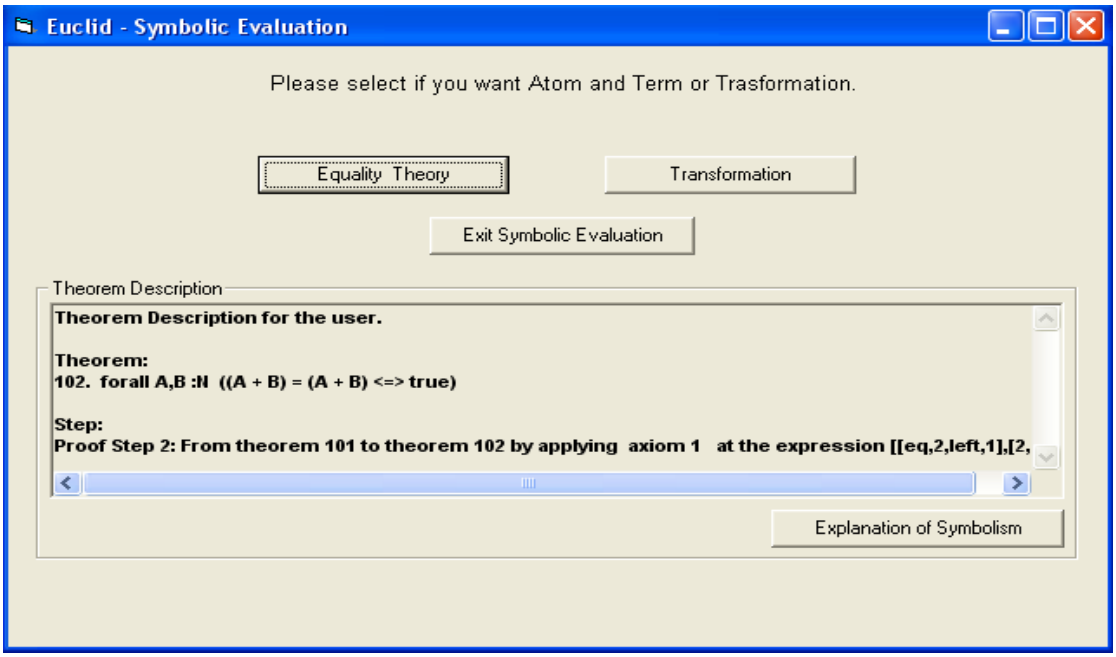


Εικόνα 4.8: *Παράδειγμα λειτουργίας του κουμπιού "Explanation of Symbolism".*

Στην εικόνα 4.9 εμφανίζεται ένας κατάλογος με τις επιλογές που έχουμε για την εφαρμογή κάποιου μετασχηματισμού. Αρχικά επιλέγουμε αν θα εφαρμόσουμε κάποιο αξίωμα, λήμμα, επαγωγή ή νόμο της λογικής πρώτης τάξης. Ανάλογα με την επιλογή εμφανίζονται τα διαθέσιμα στοιχεία από τη βάση γνώσεων και επιλέγουμε αυτό που επιθυμούμε. Στη συνέχεια επιλέγουμε το που θα εφαρμόσουμε το παραπάνω στοιχείο και δίνουμε τον ακριβή ορισμό του σημείου αυτού. Επιλέγοντας "Atom & Term" το σημείο εφαρμογής θα είναι κάποιος όρος (Term) εντός κάποιου ατόμου (Atom), "Literals" θα είναι κάποιο όρισμα, ενώ το "Theorem" αφορά όλο το θεώρημα.

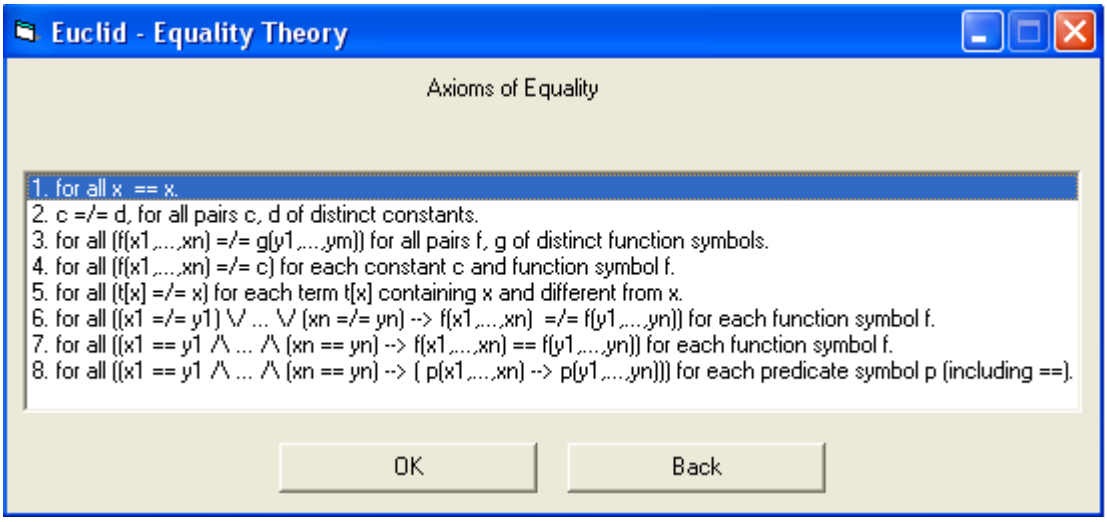


Εικόνα 4.9: *Μετασχηματισμός με εφαρμογή αξιώματος, λήμματος, επαγωγής ή νόμου της λογικής πρώτης τάξης.*



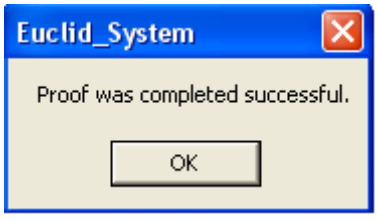
Εικόνα 4.10: *Μετασχηματισμός προγράμματος.*

Η εικόνα 4.10 εμφανίζεται όταν πατήσουμε το κουμπί "Symbolic Evaluation" στον κατάλογο επιλογών της εικόνας 4.5. Πατώντας το κουμπί "Equality Theory" ανοίγει η εικόνα 4.11 με τα αξιώματα της *θεωρίας της ισότητας*, απ' όπου επιλέγουμε αυτό που θα χρησιμοποιήσουμε. Πατώντας το κουμπί "Transformation" ανοίγει η εικόνα 4.9, όπου και εφαρμόζουμε κάποιο μετασχηματισμό, ενώ με το κουμπί "Exit Symbolic Evaluation" ξαναγυρνάμε στον κατάλογο επιλογών της εικόνας 4.5.

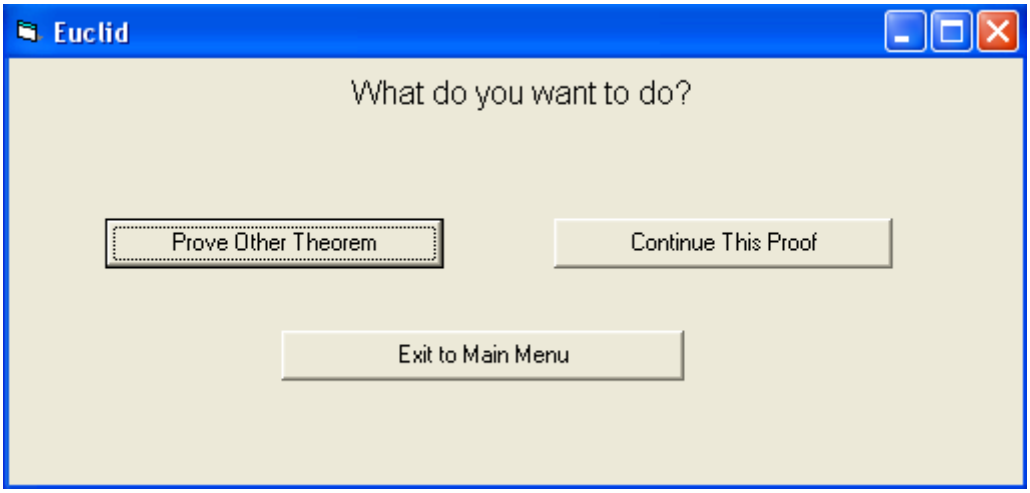


Εικόνα 4.11: *Αξιώματα Θεωρίας Ισότητας.*

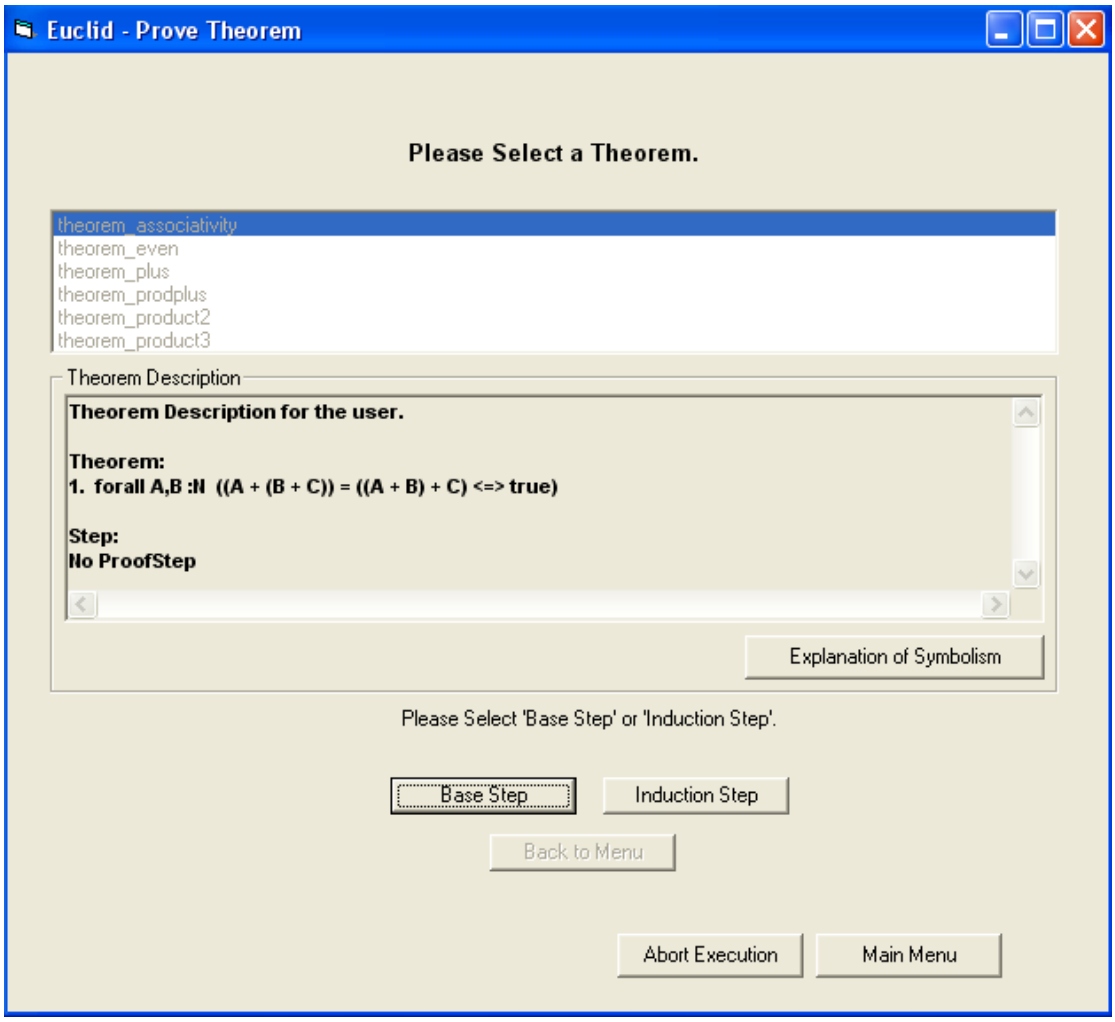
Όταν ολοκληρωθεί επιτυχώς κάποια απόδειξη βήματος βάσης ή βήματος επαγωγής ανοίγει το εικονίδιο 4.12 και πατώντας το κουμπί "OK", ανοίγει η εικόνα 4.13. Αν η απόδειξη του συγκεκριμένου **θεωρήματος** έχει ολοκληρωθεί και θέλουμε ν' αποδείξουμε κάποιο άλλο, πατάμε το κουμπί "Prove Other Theorem", οπότε και ανοίγει η εικόνα 4.2. Αν η απόδειξη του συγκεκριμένου θεωρήματος **δεν** έχει ολοκληρωθεί και θέλουμε ν' αποδείξουμε κάποιο βήμα βάσης ή βήμα επαγωγής με σκοπό την ολοκλήρωση της απόδειξης του **θεωρήματος**, πατάμε το κουμπί "Continue This Proof" και ανοίγει η εικόνα 4.14. Όπως εύκολα μπορούμε να διακρίνουμε, οι εικόνες 4.2 και 4.14 είναι όμοιες, με μόνη διαφορά ότι στην εικόνα 4.2 έχουμε τη δυνατότητα να επιλέξουμε κάποιο άλλο θεώρημα για να αποδείξουμε. Στην εικόνα 4.14 η δυνατότητα αυτή είναι απενεργοποιημένη, αφού συνεχίζουμε την απόδειξη του αρχικού θεωρήματος. Πατώντας το κουμπί "Exit to Main Menu" ανοίγει το εικονίδιο 4.1.



Εικόνα 4.12: *Μήνυμα επιτυχούς απόδειξης.*



Εικόνα 4.13: *Κατάλογος επιλογών μετά την ολοκλήρωση κάποιας απόδειξης .*

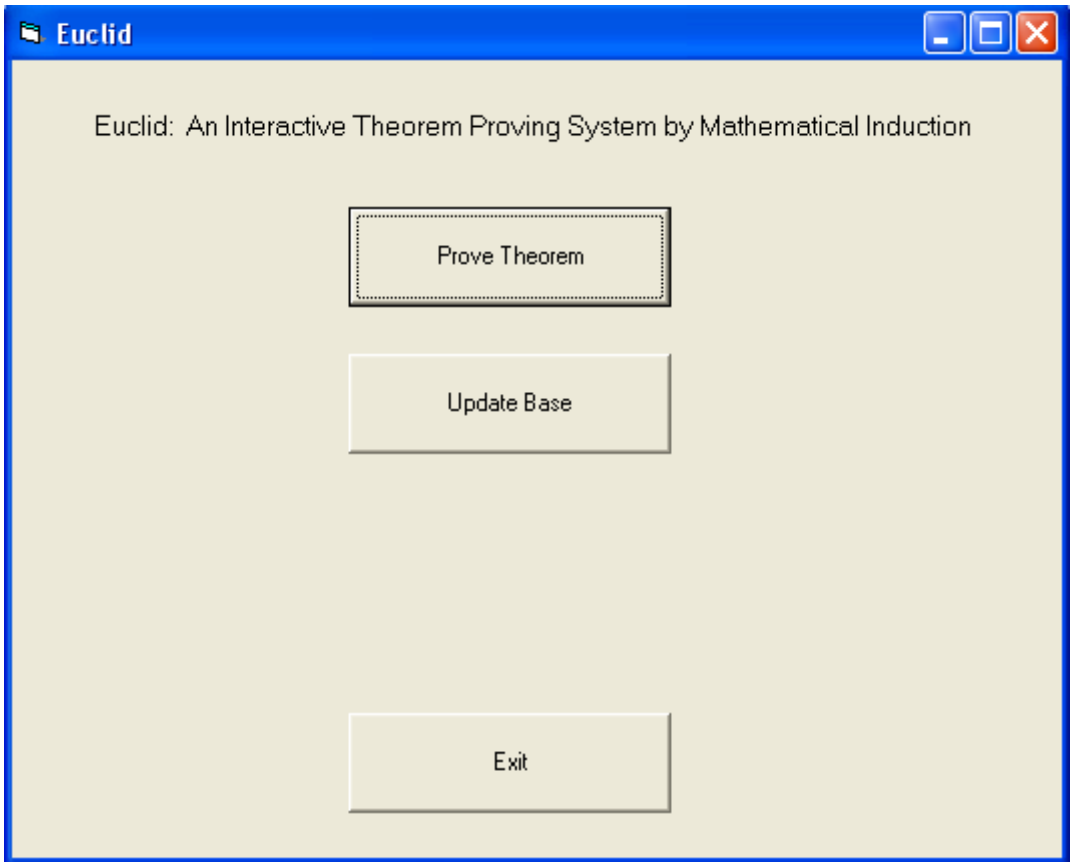


Εικόνα 4.14: *Συνέχιση απόδειξης θεωρήματος.*

4.5 Ένα πλήρες Σενάριο Χρήσης του συστήματος Ευκλείδης

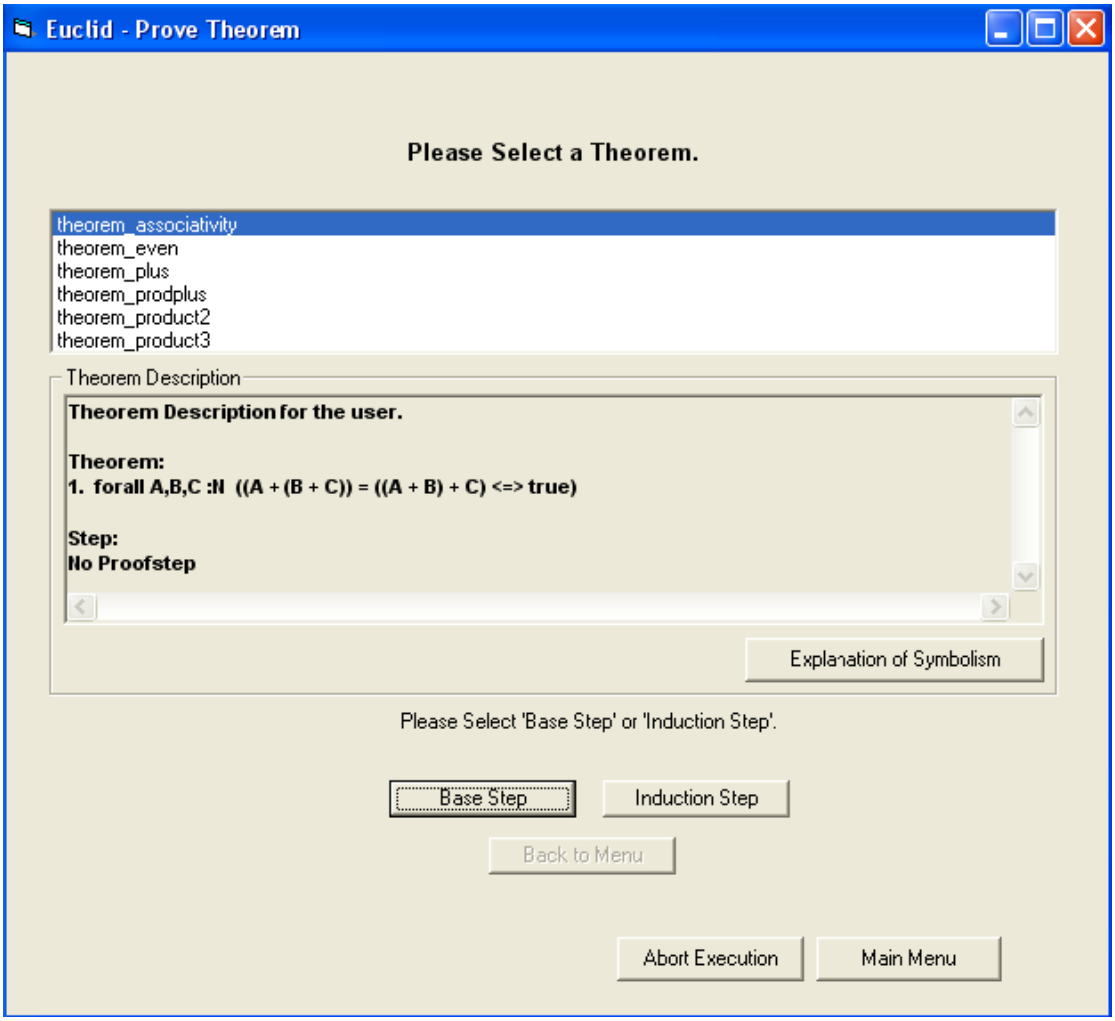
Στο κεφάλαιο αυτό θα δούμε αναλυτικά πως γίνεται η απόδειξη του θεωρήματος “associativity”.

Από το κεντρικό μενού του συστήματος επιλέγουμε το πλήκτρο "**Prove Theorem**", εικόνα 4.15.



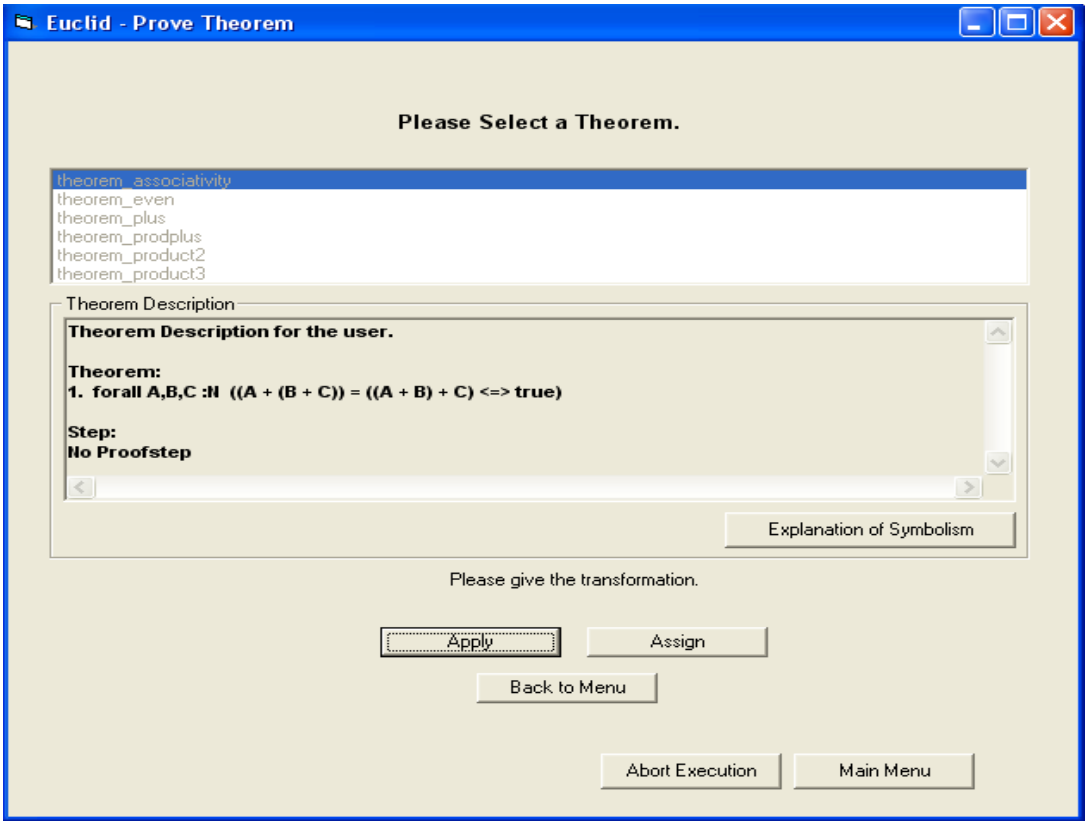
Εικόνα 4.15: *Πλήρες Σενάριο – Κύριος κατάλογος.*

Εμφανίζεται ένα καινούργιο παράθυρο που περιέχει όλα τα αρχικά θεωρήματα που μπορούν να αποδειχθούν, εικόνα 4.16. Εμείς επιλέγουμε το " **associativity** ".

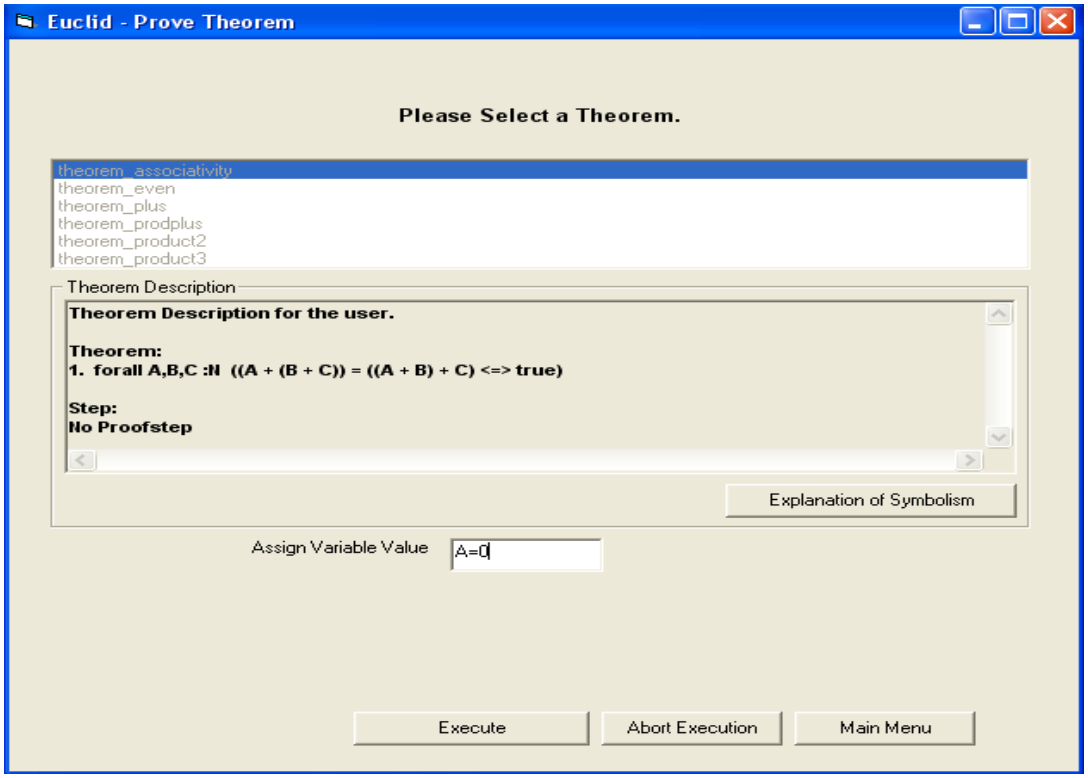


Εικόνα 4.16: *Πλήρες Σενάριο – Επιλογή θεωρήματος για απόδειξη.*

Κάτω από το μενού των θεωρημάτων εμφανίζεται, ένα παράθυρο που περιέχει το θεώρημα που θα αποδείξουμε καθώς και την ταυτότητα αυτού του θεωρήματος, εικόνα 4.16. Επιλέγοντας το πλήκτρο "**Base Step**" και έχοντας ήδη επιλέξει το θεώρημα που επιθυμούμε ν' αποδείξουμε, ανοίγει η εικόνα 4.17 και αρχίζει η απόδειξη του θεωρήματος αυτού. Σ' αυτήν την εικόνα βλέπουμε τις επιλογές "assign" και "apply", με τις οποίες μπορούμε να εφαρμόσουμε κάποιο μετασχηματισμό στο θεώρημα. Πατώντας αρχικά το πλήκτρο "assign" επιλέγουμε τη μεταβλητή επαγωγής και καταχωρούμε κάποια τιμή σε αυτήν (εικόνα 4.18).

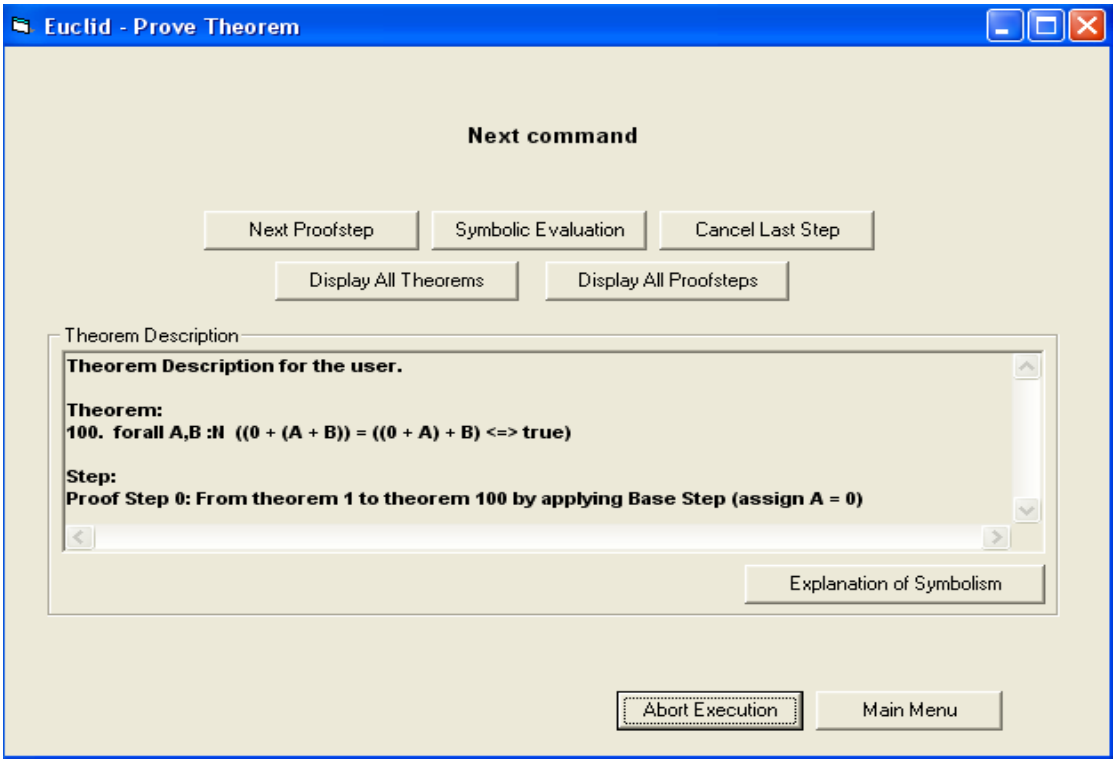


Εικόνα 4.17: *Πλήρες Σενάριο – Έναρξη απόδειξης θεωρήματος .*



Εικόνα 4.18: *Πλήρες Σενάριο – Επιλογή μεταβλητής επαγωγής και καταχώρηση τιμής σ' αυτήν.*

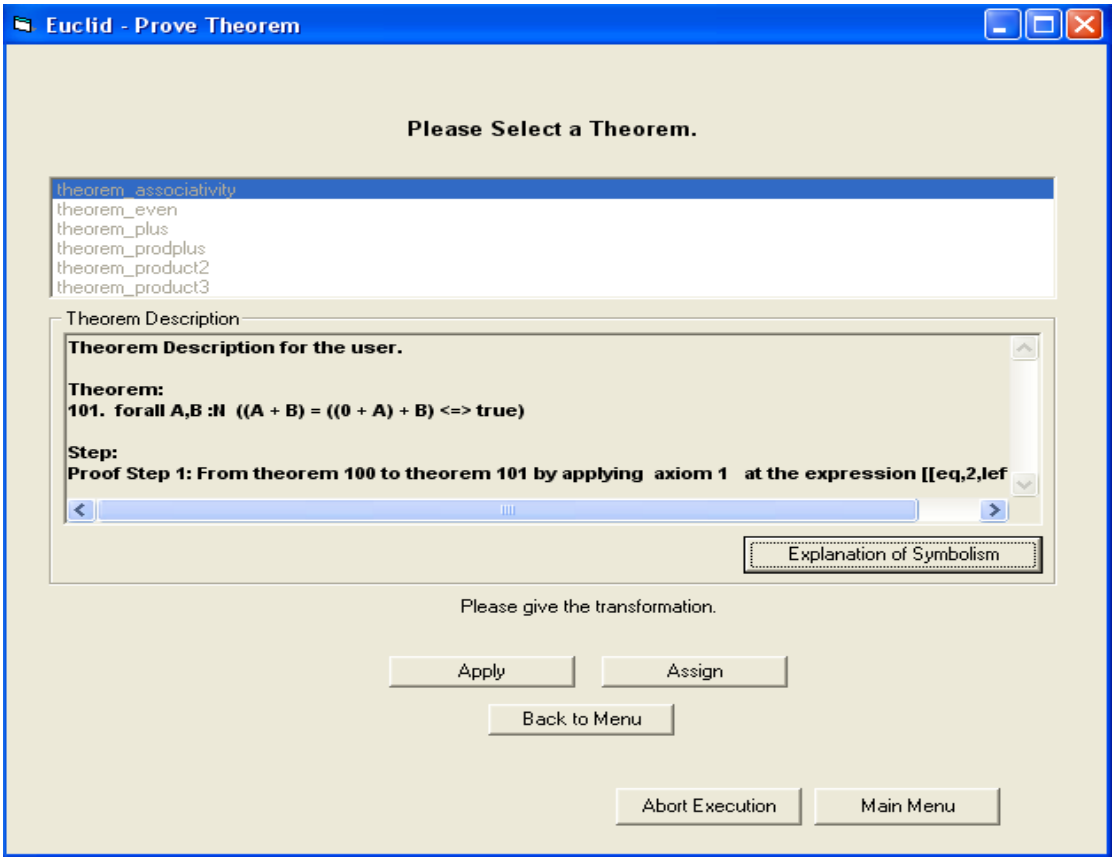
Πατώντας το πλήκτρο "Execute" έχει πραγματοποιηθεί η καταχώρηση της τιμής στην μεταβλητή και ανοίγει η εικόνα 4.19,όπου εμφανίζονται οι επιλογές που έχουμε.



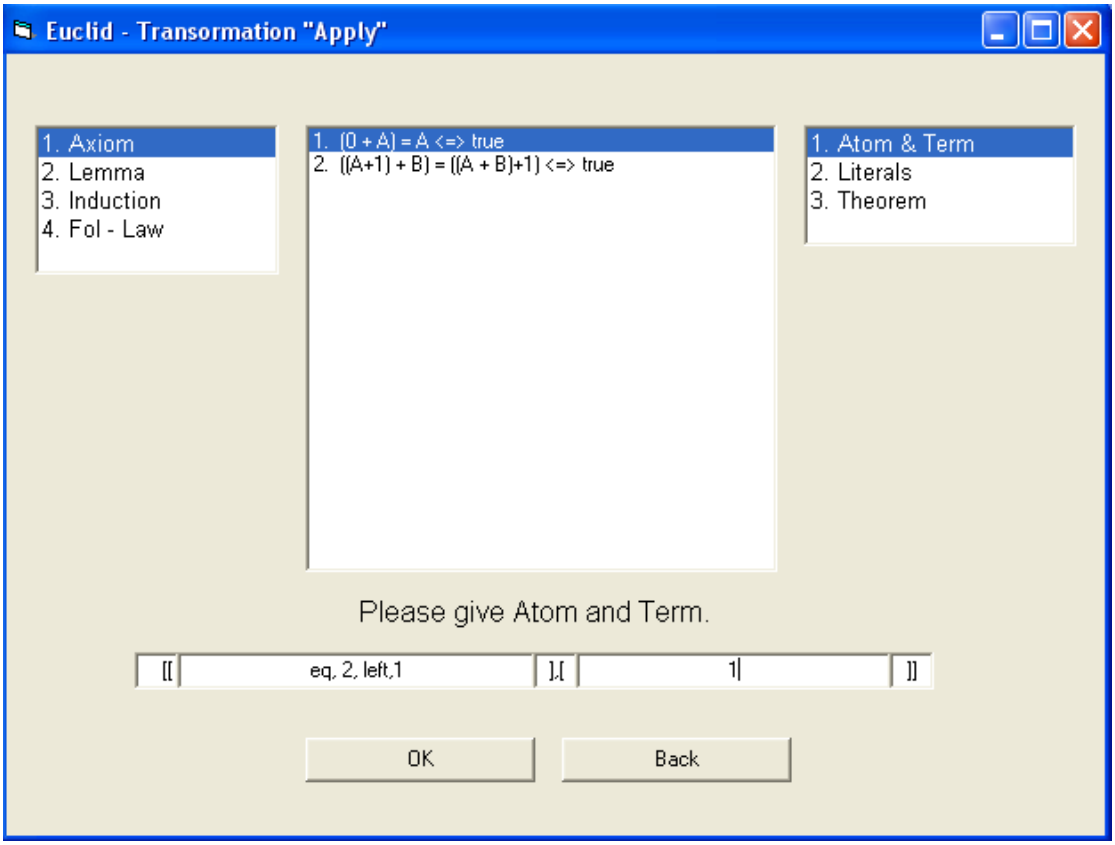
Εικόνα 4.19: *Πλήρες Σενάριο – Επόμενο βήμα απόδειξης.*

Πατώντας το πλήκτρο “Next Proofstep” συνεχίζουμε την απόδειξη με κάποιο καινούργιο μετασχηματισμό, με το “Symbolic Evaluation” ολοκληρώνουμε την απόδειξη του βήματος βάσης ή του βήματος επαγωγής, με το πλήκτρο “Cancel Last Step”ακυρώνουμε τον τελευταίο μετασχηματισμό, ενώ με τα πλήκτρα “Display All Theorems” και “Display All Proofsteps” μπορούμε να δούμε όλα τα θεωρήματα και τα βήματα απόδειξης.

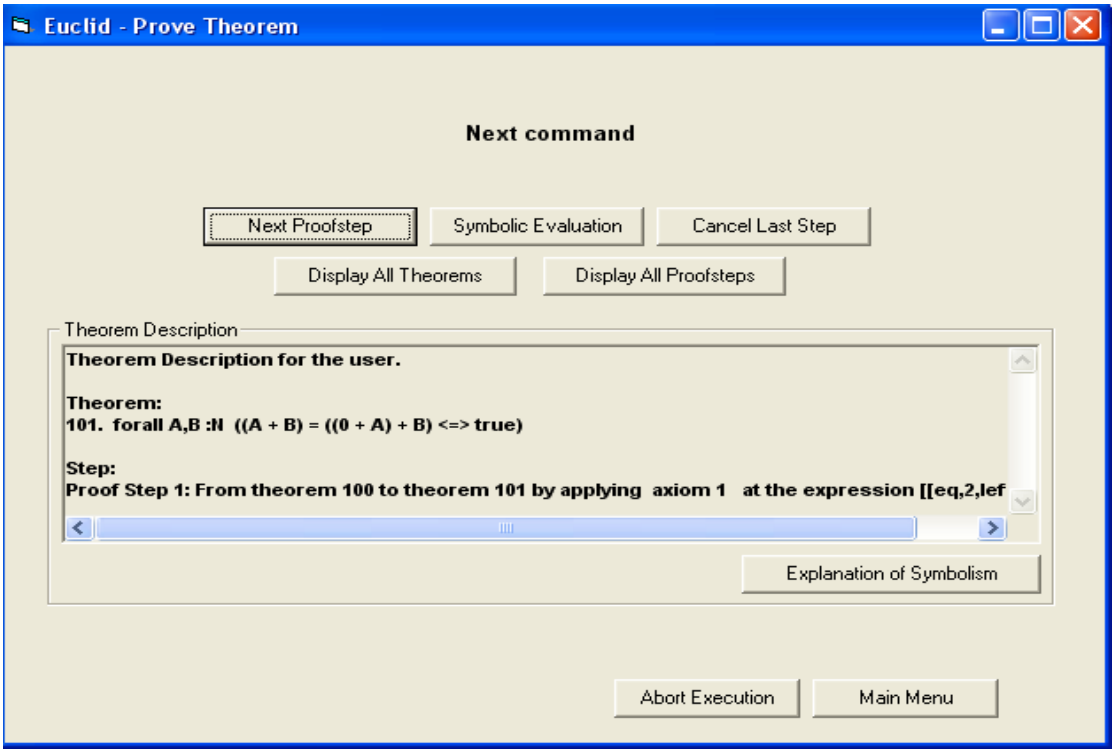
Επιλέγουμε το “Next Proofstep” ώστε να συνεχίσουμε την απόδειξη του θεωρήματος και ανοίγει η εικόνα 4.20. Πατάμε το πλήκτρο “apply”,ώστε να εφαρμόσουμε κάποιο μετασχηματισμό και ανοίγει η εικόνα 4.21.



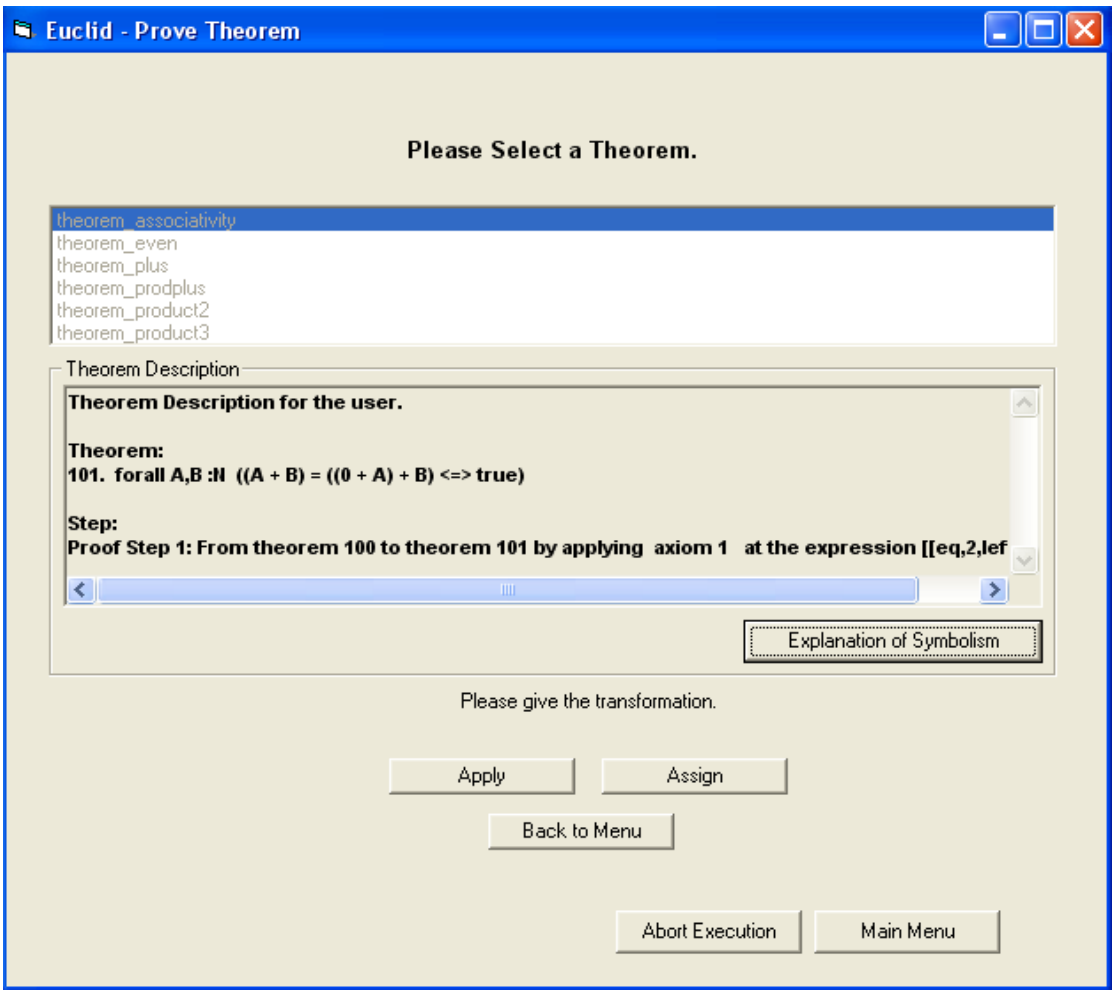
Εικόνα 4.20: *Πλήρες Σενάριο – Επιλογή εκτέλεσης μετασχηματισμού.*



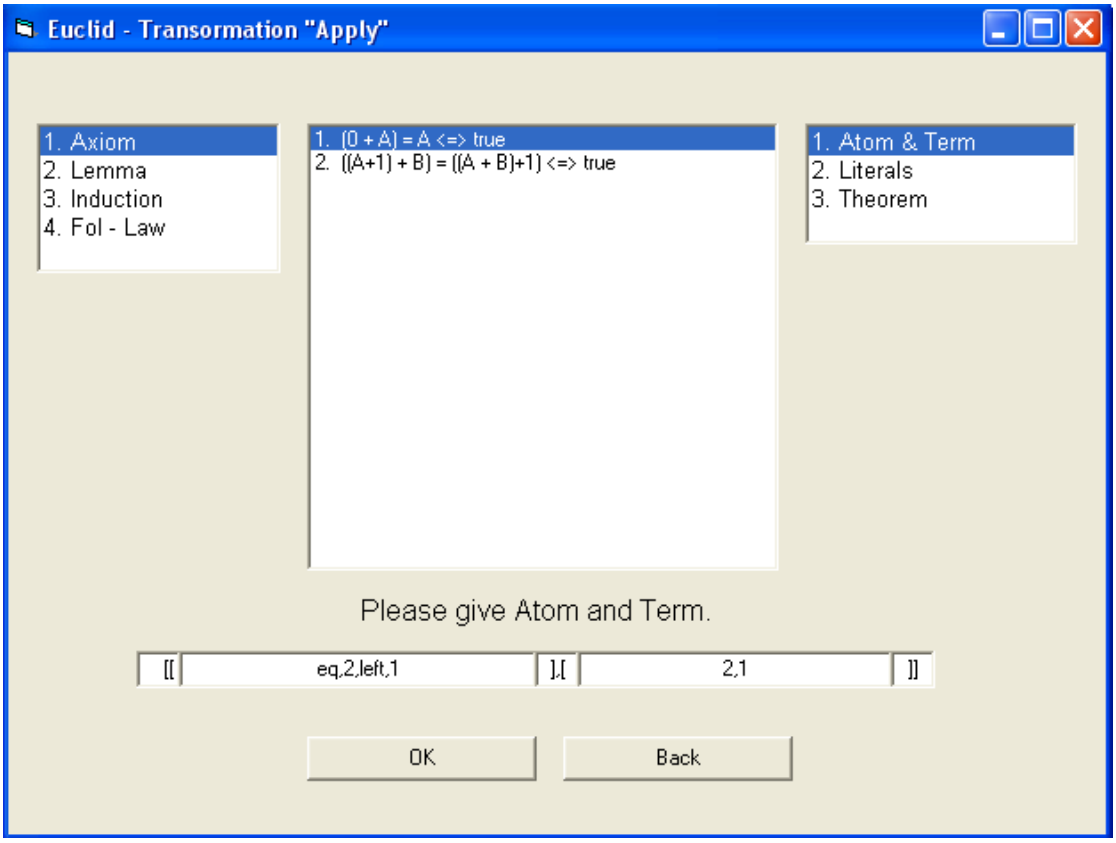
Εικόνα 4.21: *Πλήρες Σενάριο – Μετασχηματισμός "apply".*



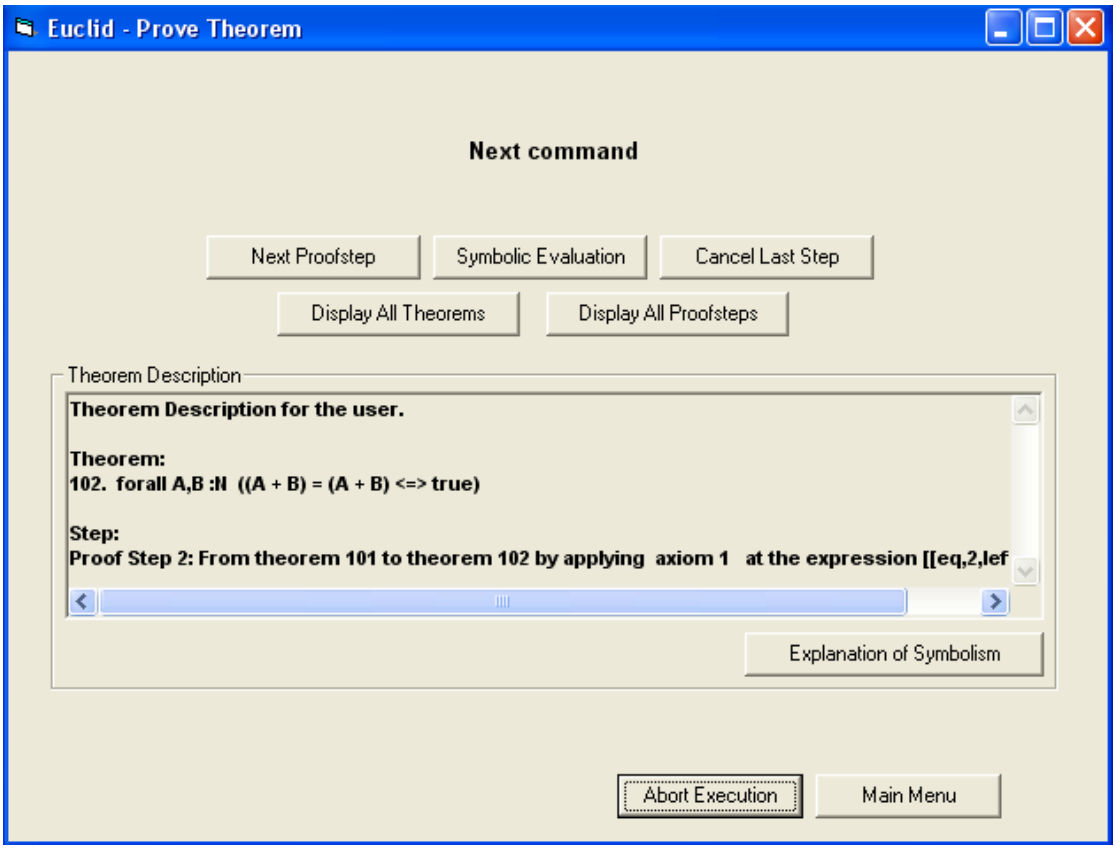
Εικόνα 4.22: *Πλήρες Σενάριο – Επόμενο βήμα απόδειξης.*



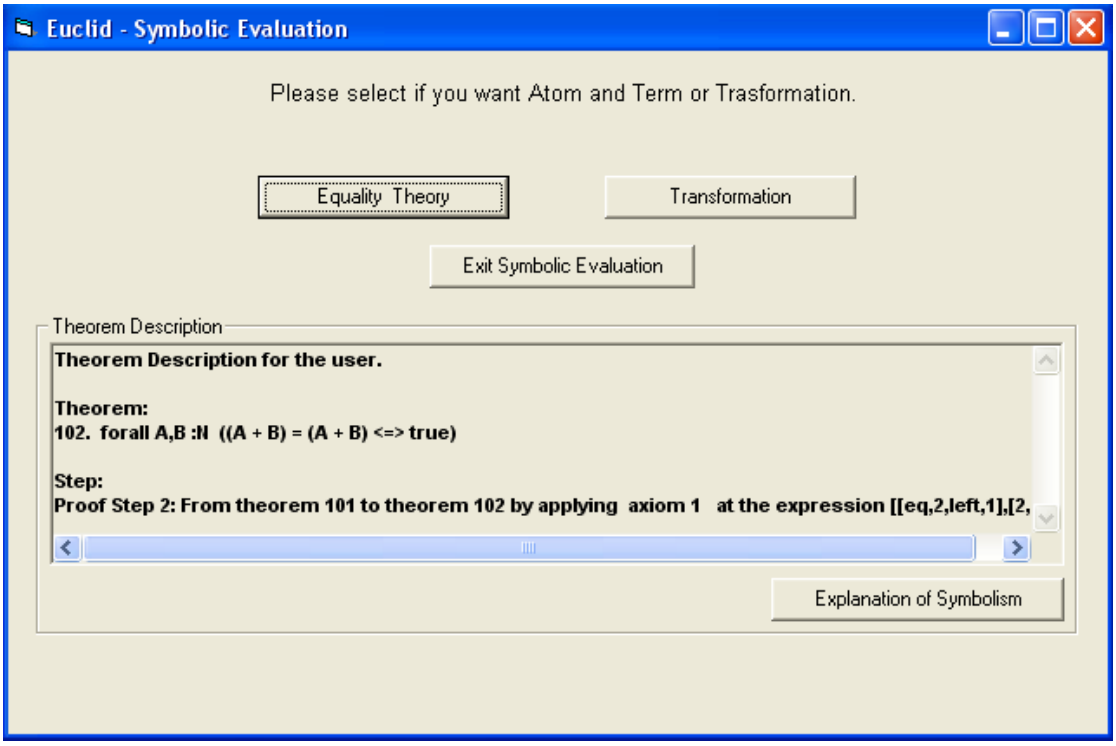
Εικόνα 4.23: *Πλήρες Σενάριο – Επιλογή εκτέλεσης μετασχηματισμού.*



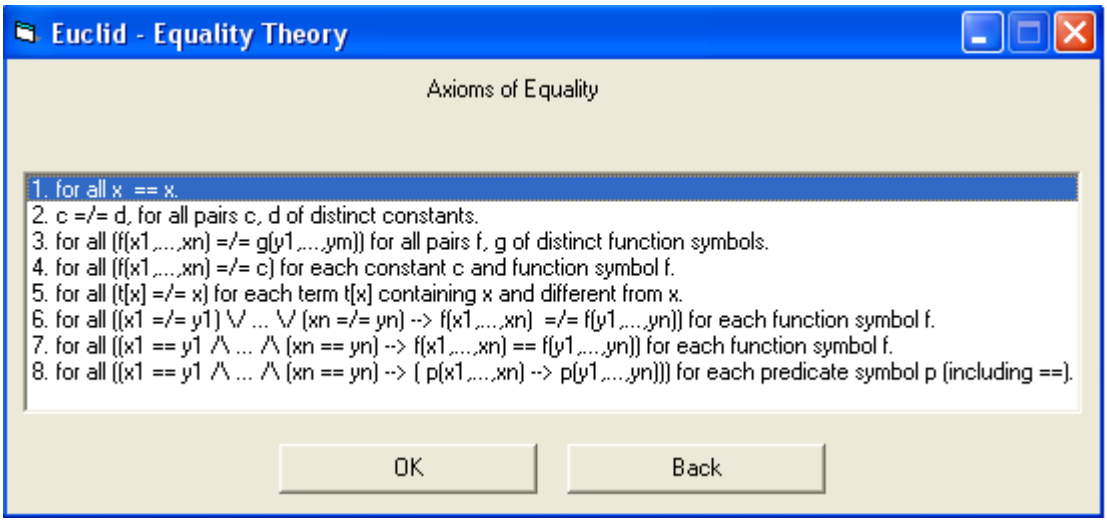
Εικόνα 4.24: *Πλήρες Σενάριο – Μετασχηματισμός "apply".*



Εικόνα 4.25: *Πλήρες Σενάριο – Επόμενο βήμα απόδειξης.*

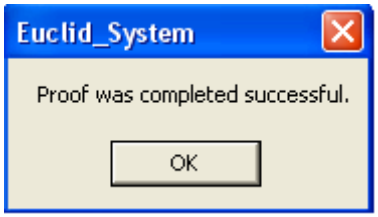


Εικόνα 4.26: *Πλήρες Σενάριο – "Symbolic Evaluation".*

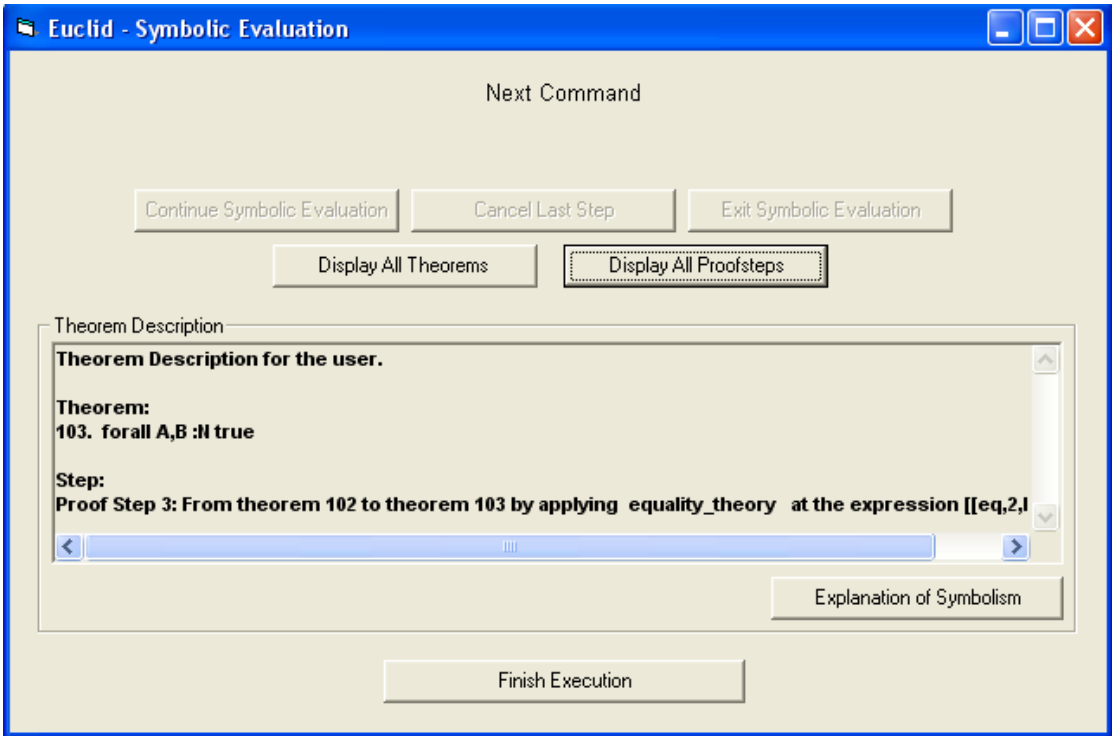


Εικόνα 4.27: *Πλήρες Σενάριο – Αξιώματα της θεωρίας της ισότητας.*

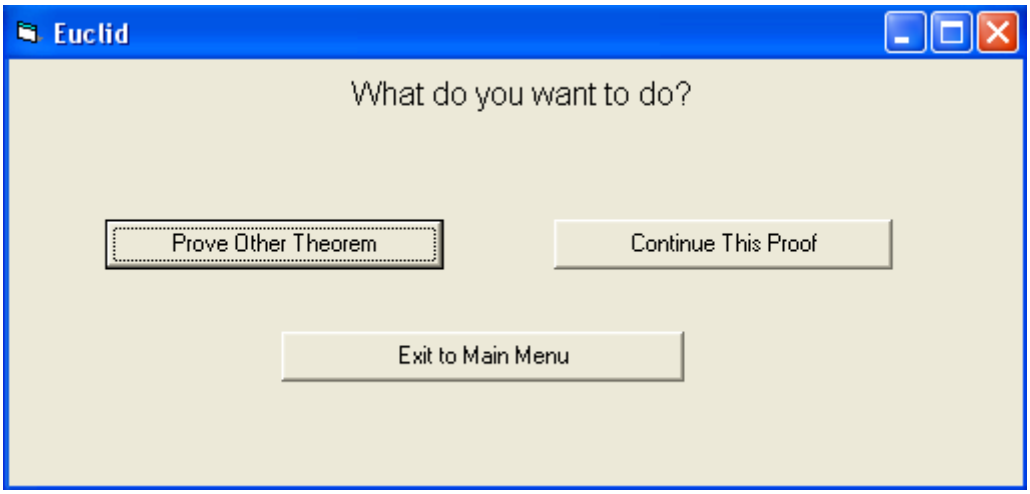
Η Visual Basic επιστρέφει το παρακάτω μήνυμα **"Proof was completed successful"** σε περίπτωση επιτυχούς απόδειξης του βήματος βάσης ή του βήματος επαγωγής του θεωρήματος, εικόνα 4.28:



Εικόνα 4.28: *Πλήρες Σενάριο – Ολοκλήρωση απόδειξης βήματος βάσης ή βήματος επαγωγής.*

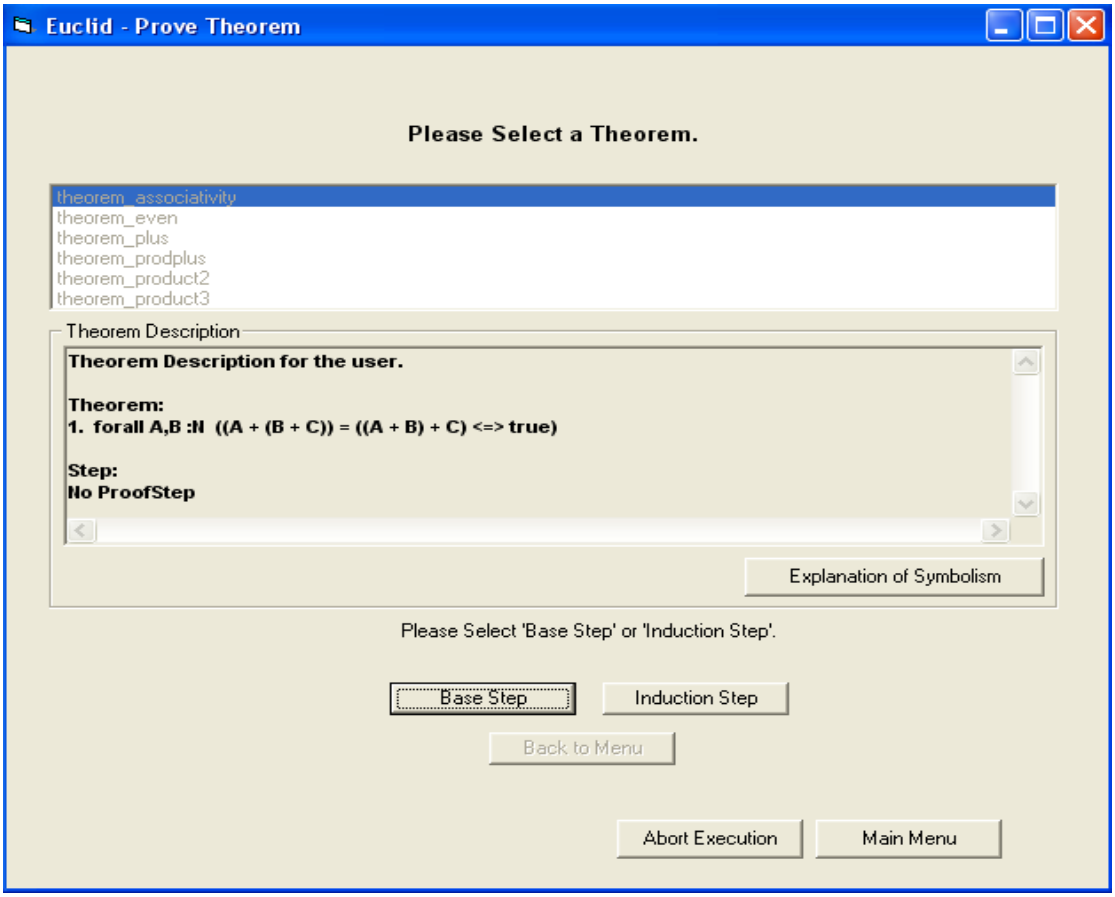


Εικόνα 4.29: *Πλήρες Σενάριο – Εκτύπωση όλων των θεωρημάτων και των βημάτων απόδειξης.*

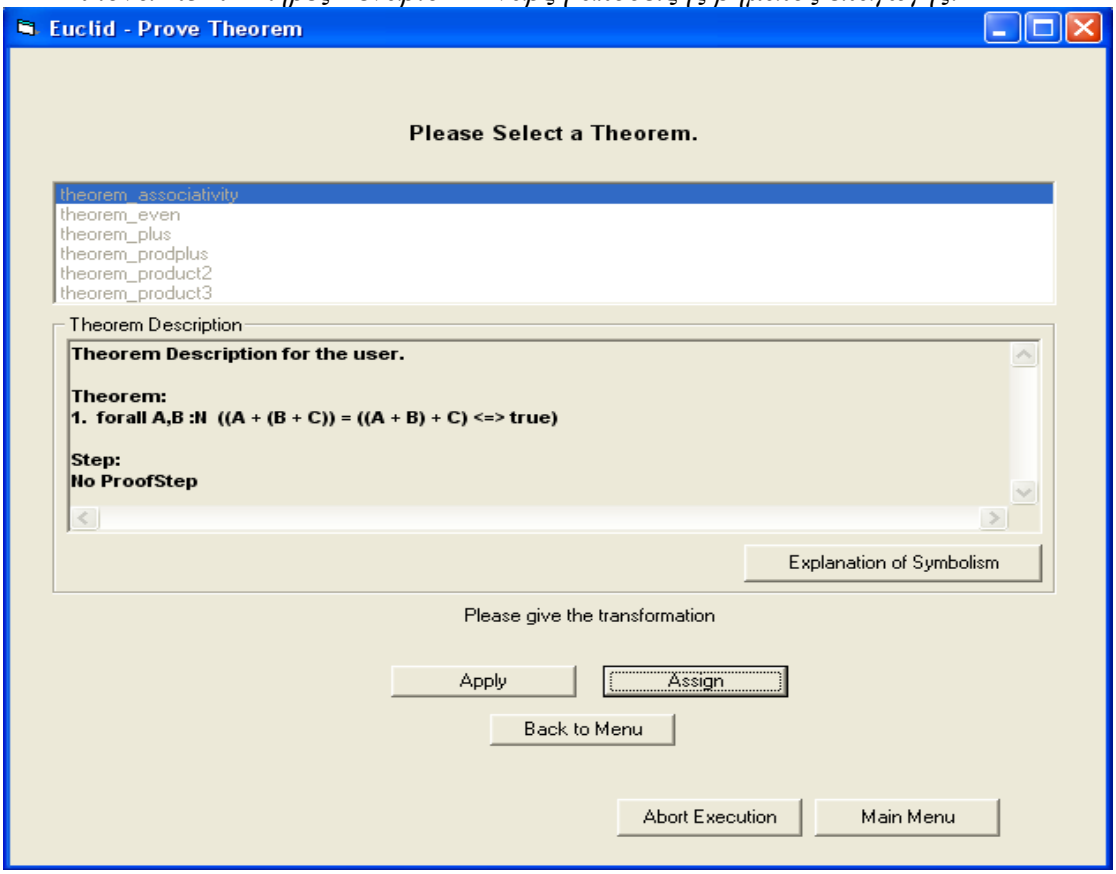


Εικόνα 4.30: *Πλήρες Σενάριο –Επιλογές για συνέχιση απόδειξης του ίδιου ή διαφορετικού θεωρήματος ή για έξοδο στο βασικό μενού.*

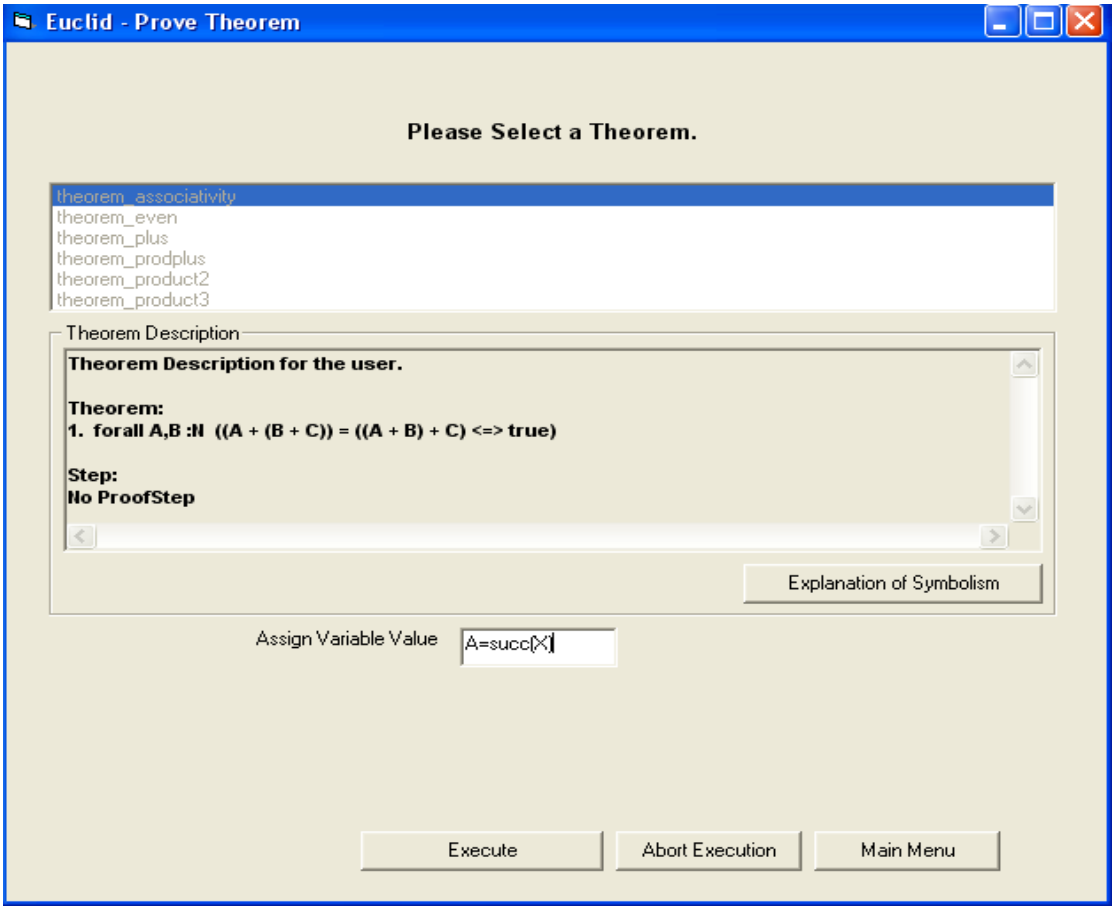
Πατώντας το πλήκτρο “Continue This Proof” ανοίγει η εικόνα 4.31, όπου και πατάμε “Induction Step”, ώστε να αποδείξουμε και το βήμα επαγωγής. Στη συνέχεια ακολουθούμε την ίδια διαδικασία με παραπάνω (αφού πατήσουμε Induction Step) (εικόνα 4.31), αλλάζοντας όμως τη μεταβλητή επαγωγής σε “succ(X)” (εικόνα 4.32) και προσαρμόζοντας κατάλληλα τους υπόλοιπους μετασχηματισμούς.



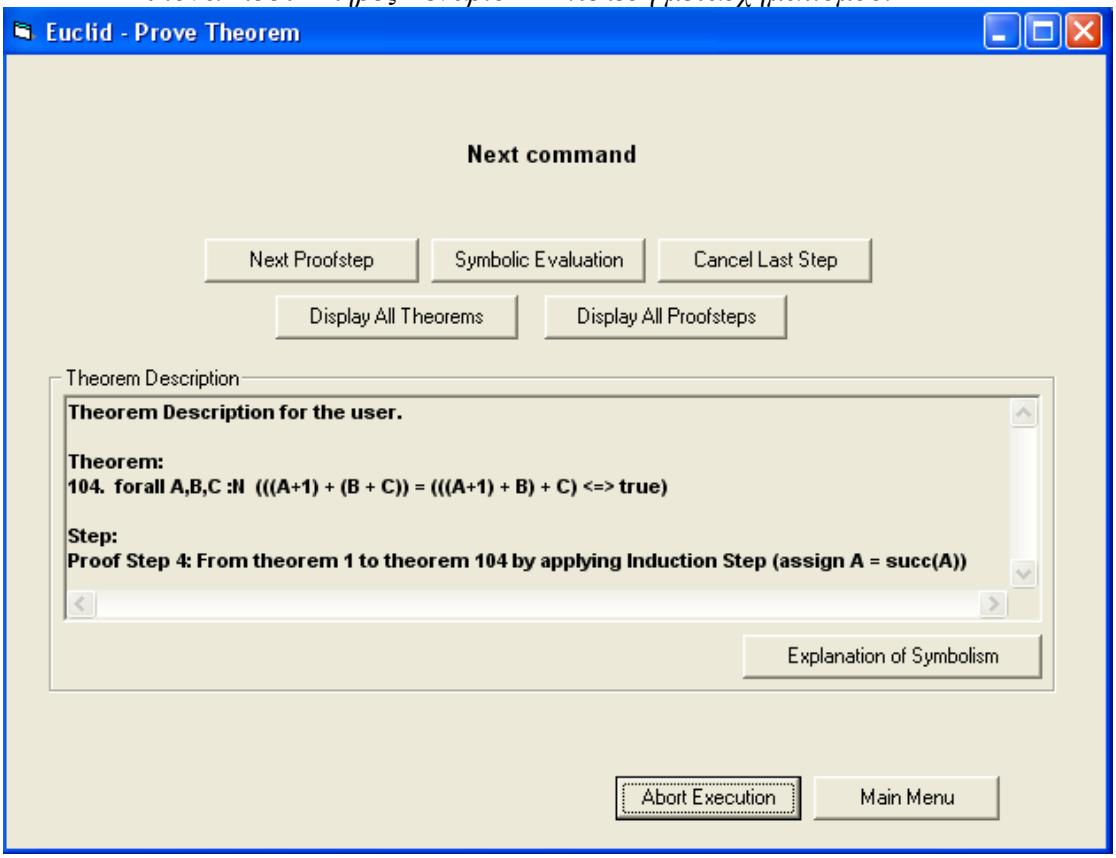
Εικόνα 4.31: *Πλήρες Σενάριο – Έναρξη απόδειξης βήματος επαγωγής.*



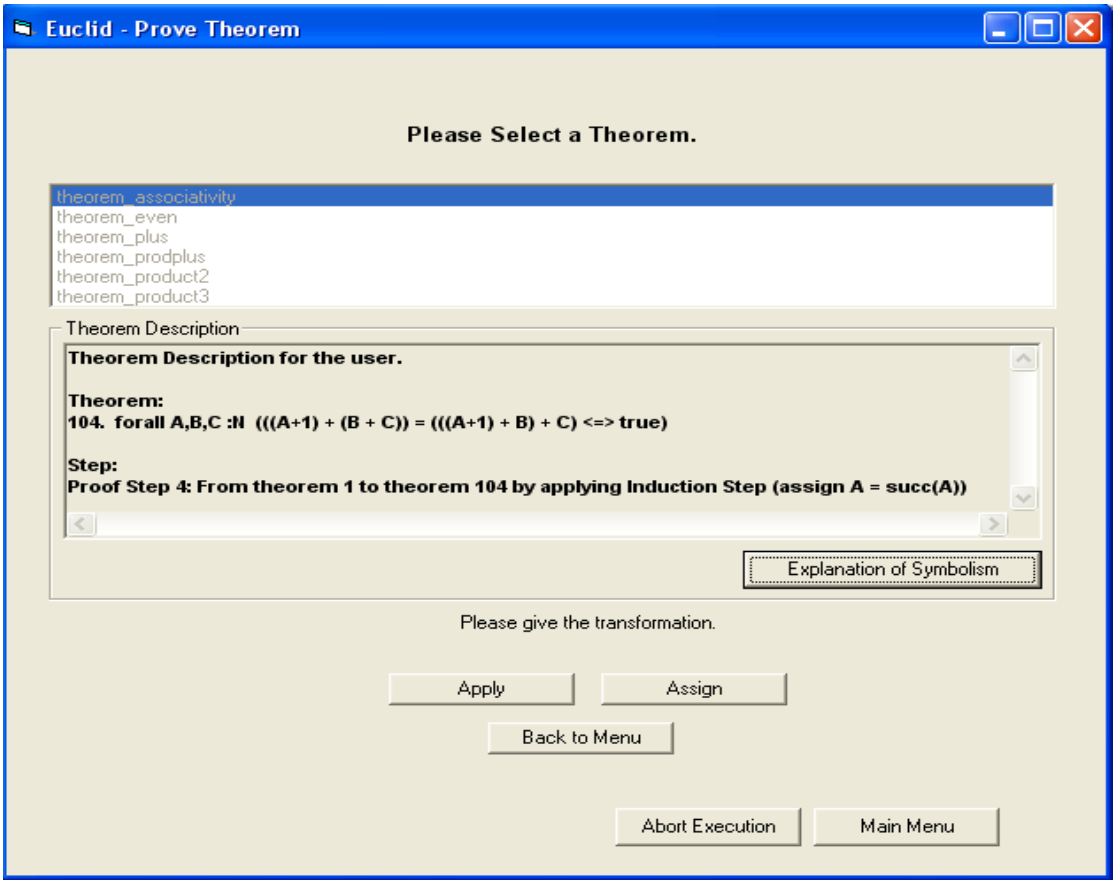
Εικόνα 4.32: *Πλήρες Σενάριο – Επιλογή μεταβλητής επαγωγής και καταχώρηση τιμής σ' αυτήν.*



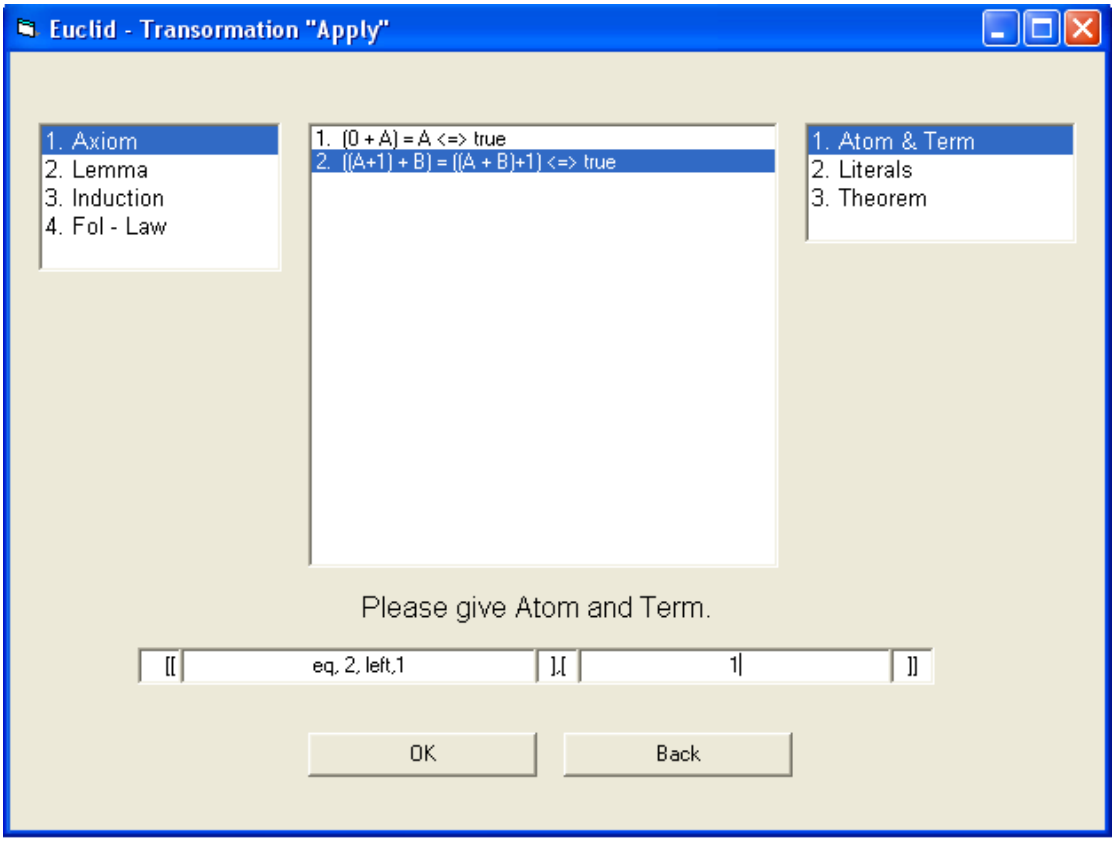
Εικόνα 4.33: *Πλήρες Σενάριο – Εκτέλεση μετασχηματισμού.*



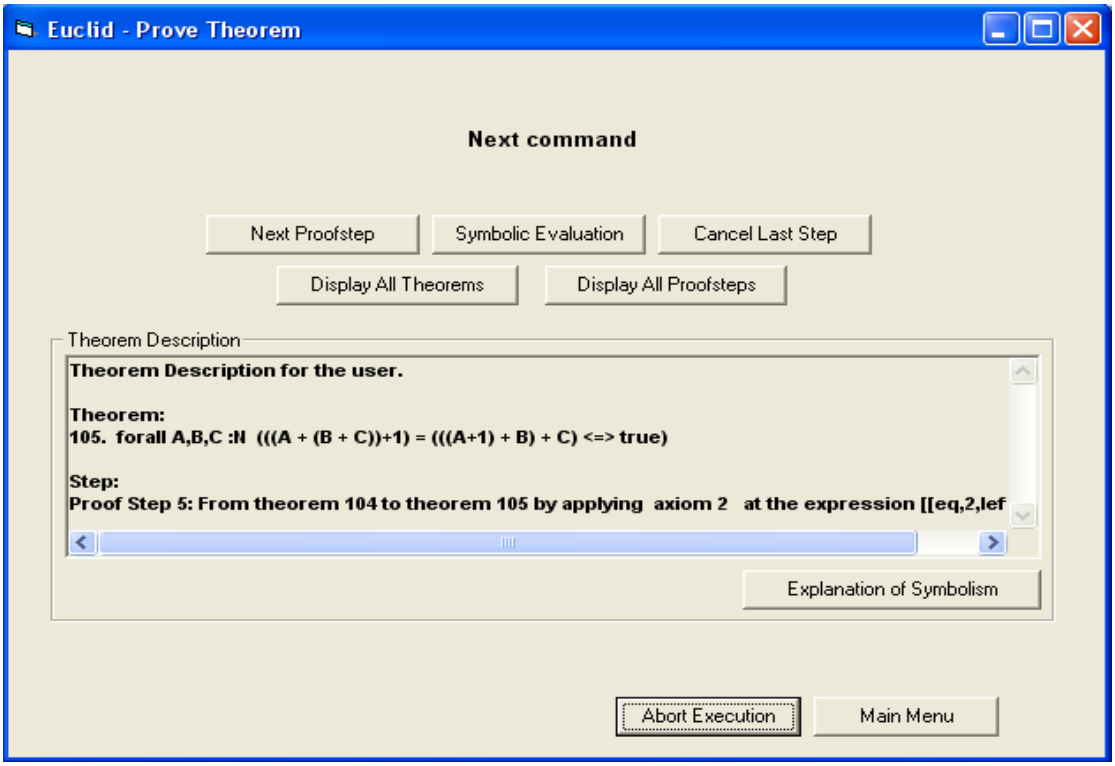
Εικόνα 4.34: *Πλήρες Σενάριο – Επόμενο βήμα απόδειξης.*



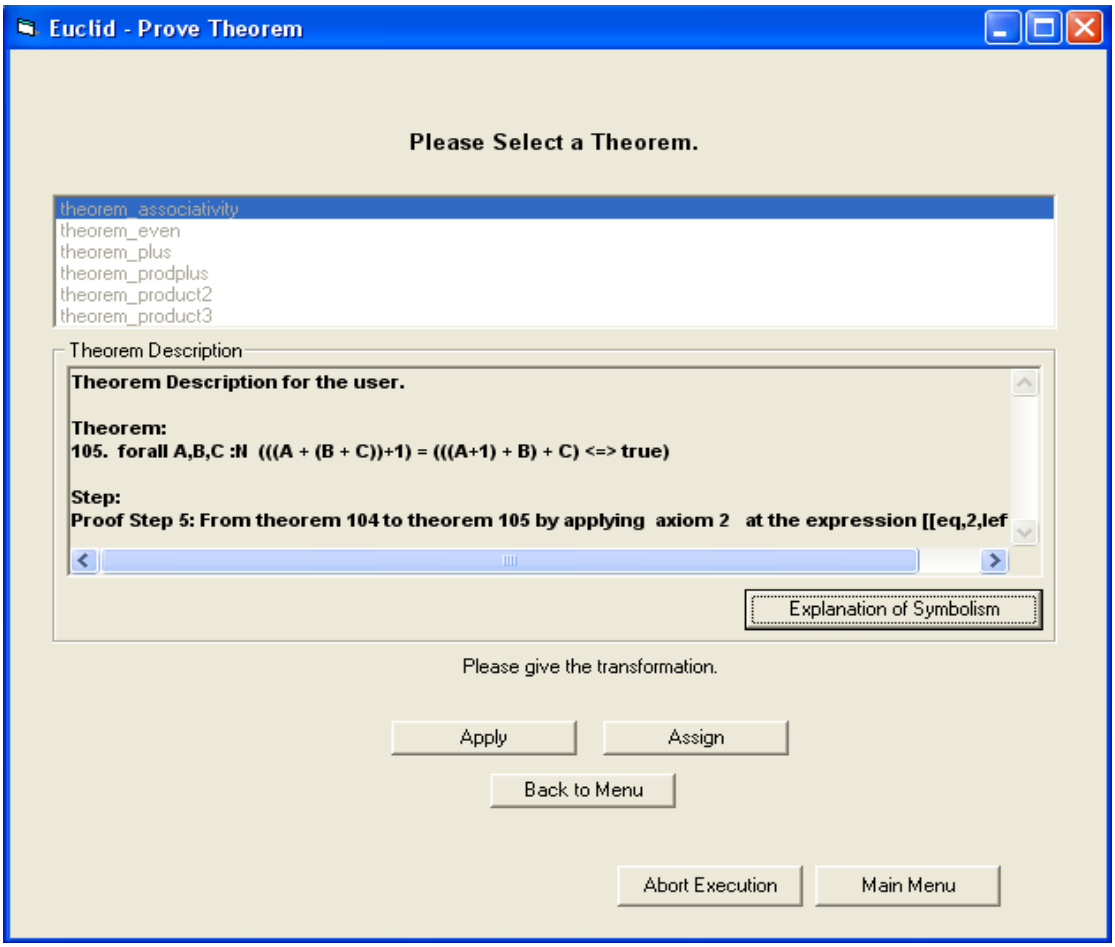
Εικόνα 4.35: *Πλήρες Σενάριο – Επιλογή εκτέλεσης μετασχηματισμού.*



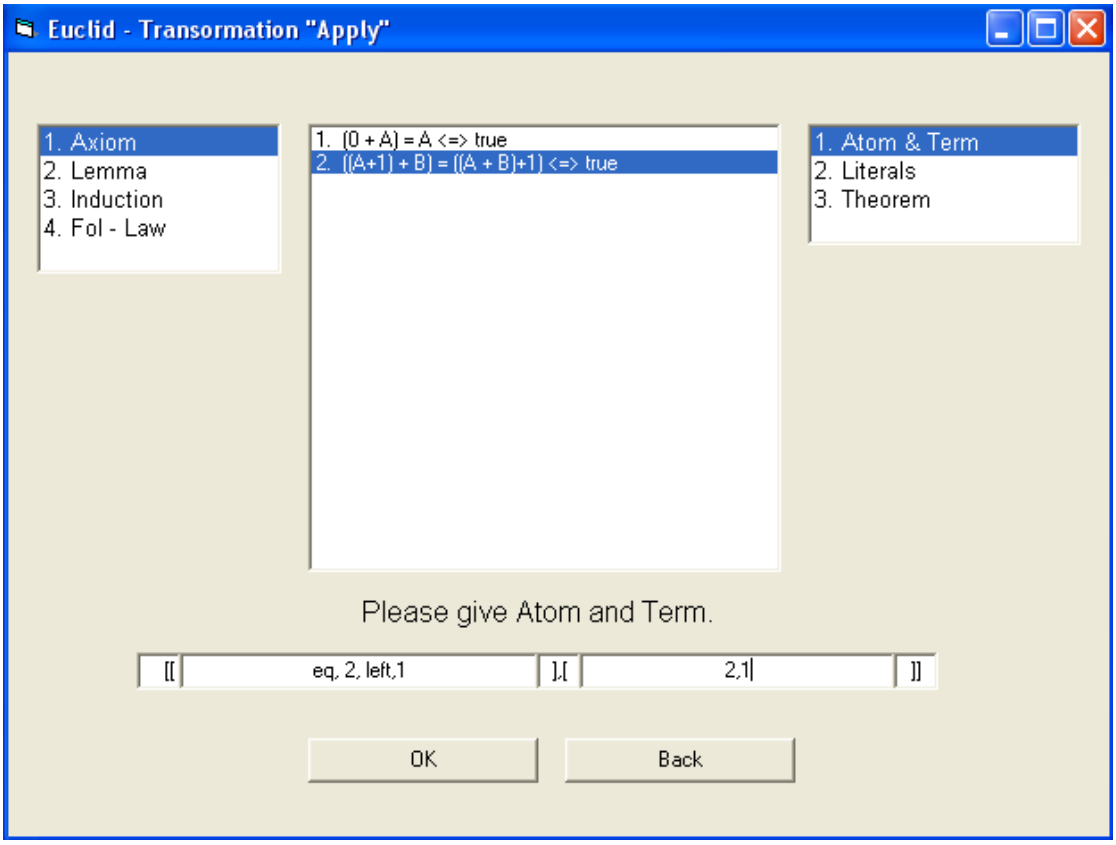
Εικόνα 4.36: *Πλήρες Σενάριο – Μετασχηματισμός "apply".*



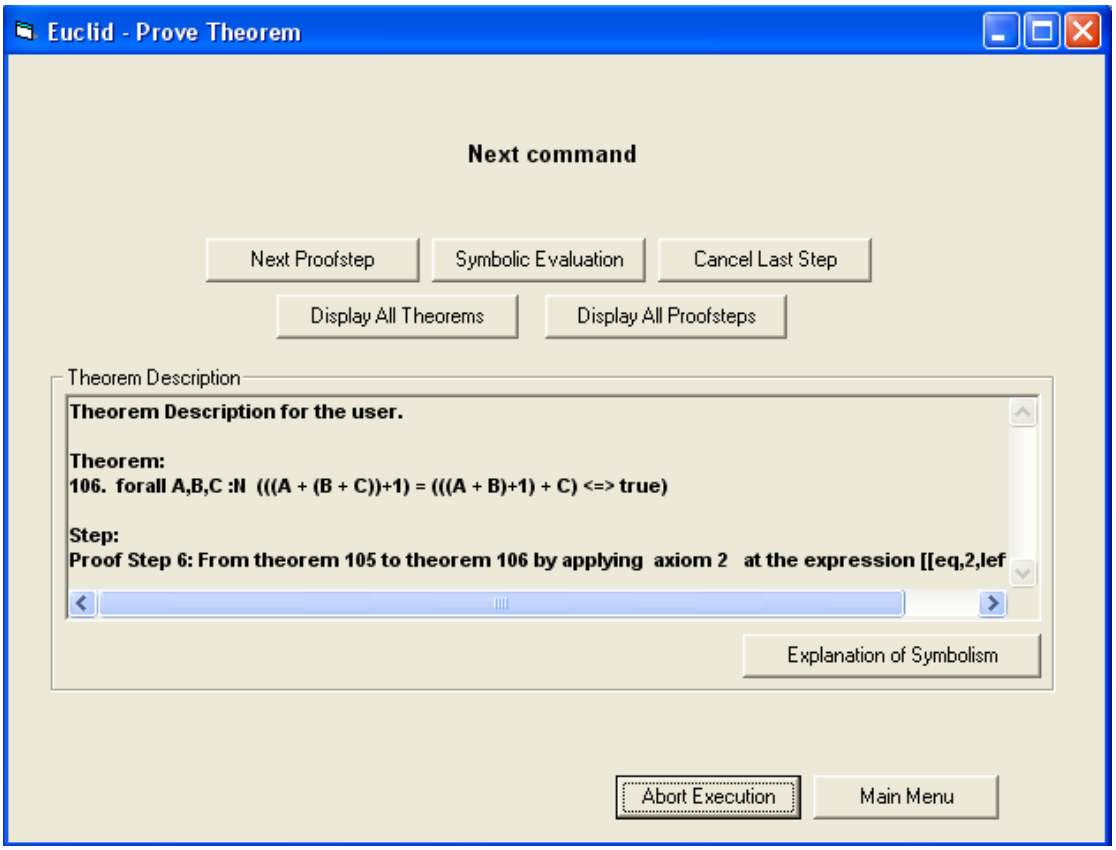
Εικόνα 4.37: *Πλήρες Σενάριο – Επόμενο βήμα απόδειξης.*



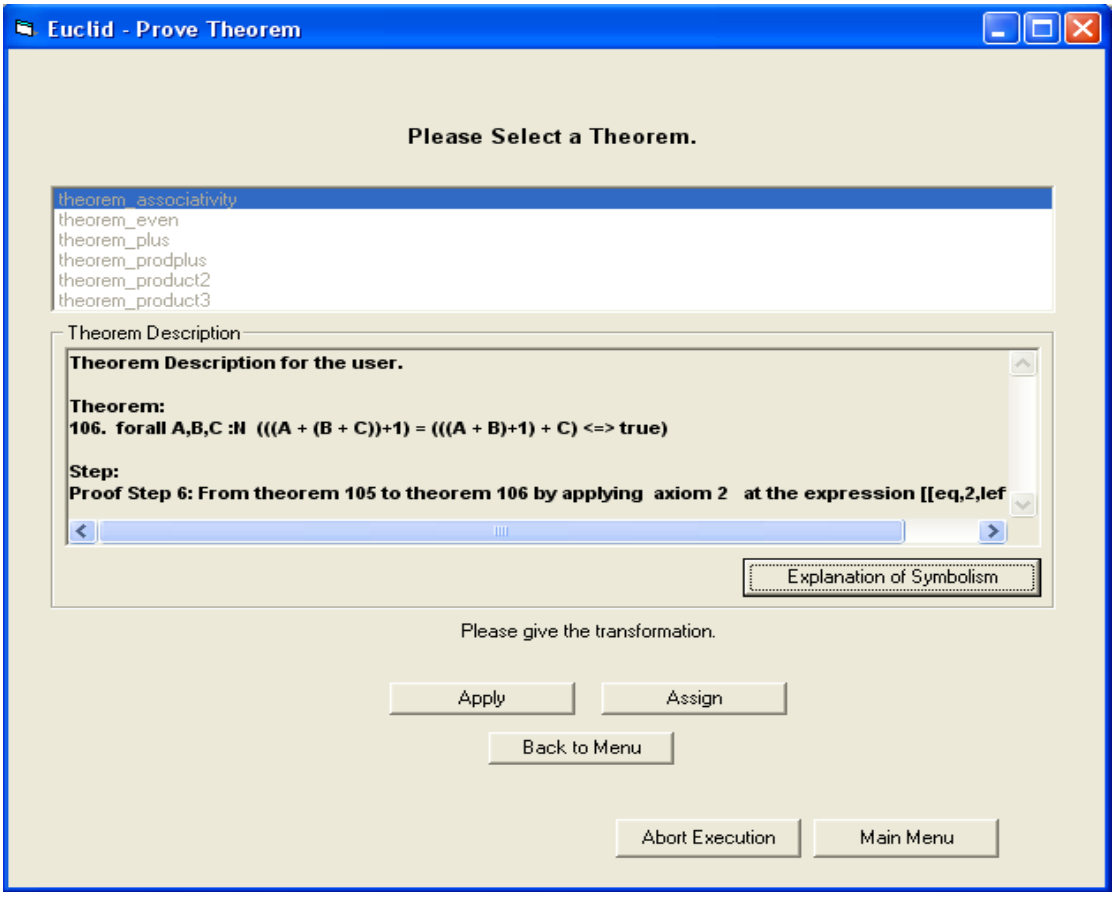
Εικόνα 4.38: *Πλήρες Σενάριο – Επιλογή εκτέλεσης μετασχηματισμού.*



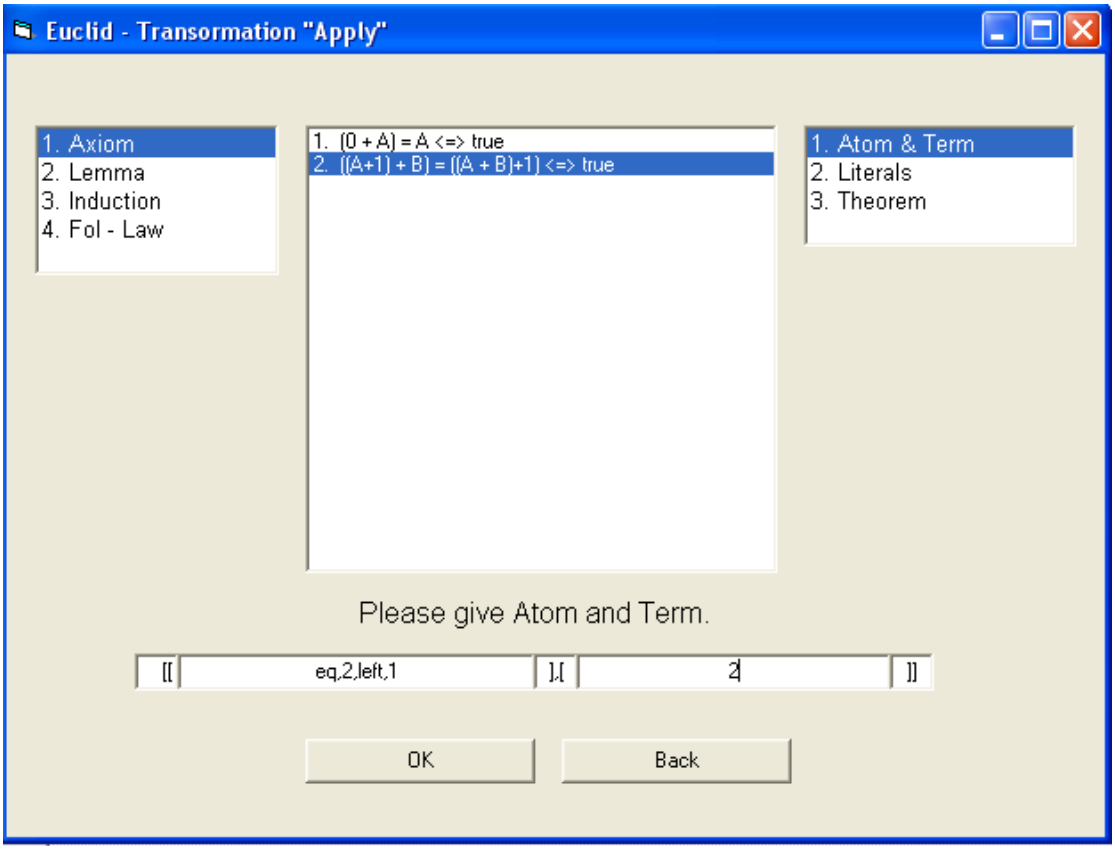
Εικόνα 4.39: *Πλήρες Σενάριο – Μετασχηματισμός "apply".*



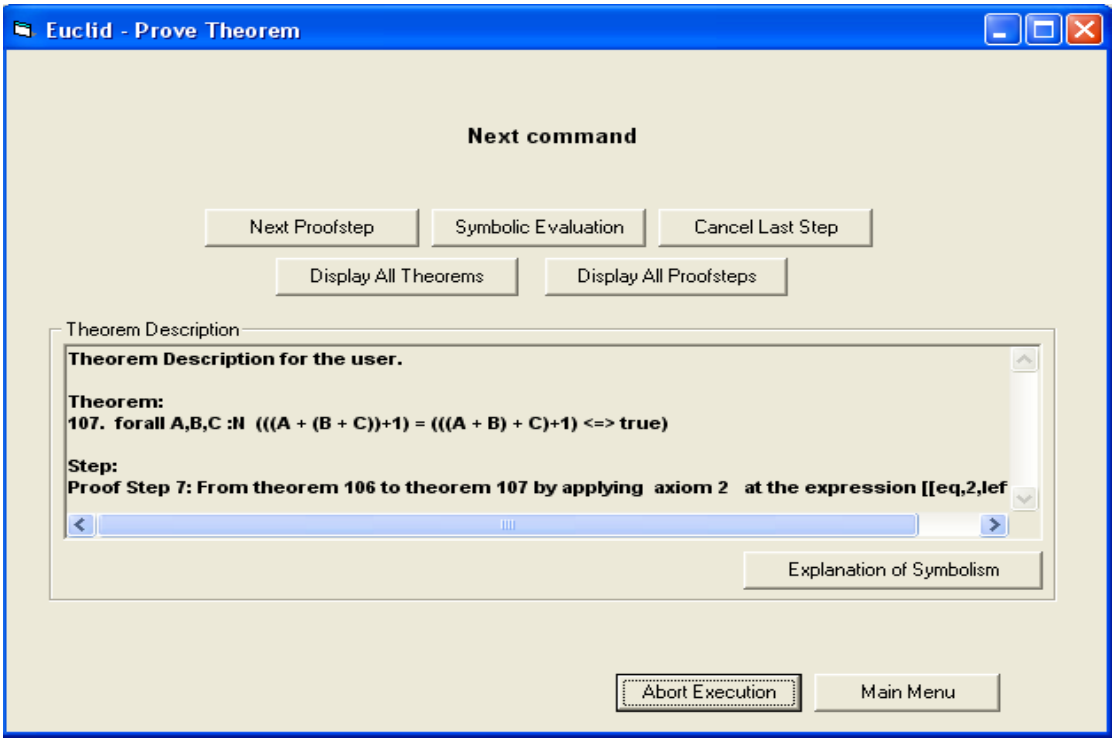
Εικόνα 4.40: *Πλήρες Σενάριο – Επόμενο βήμα απόδειξης.*



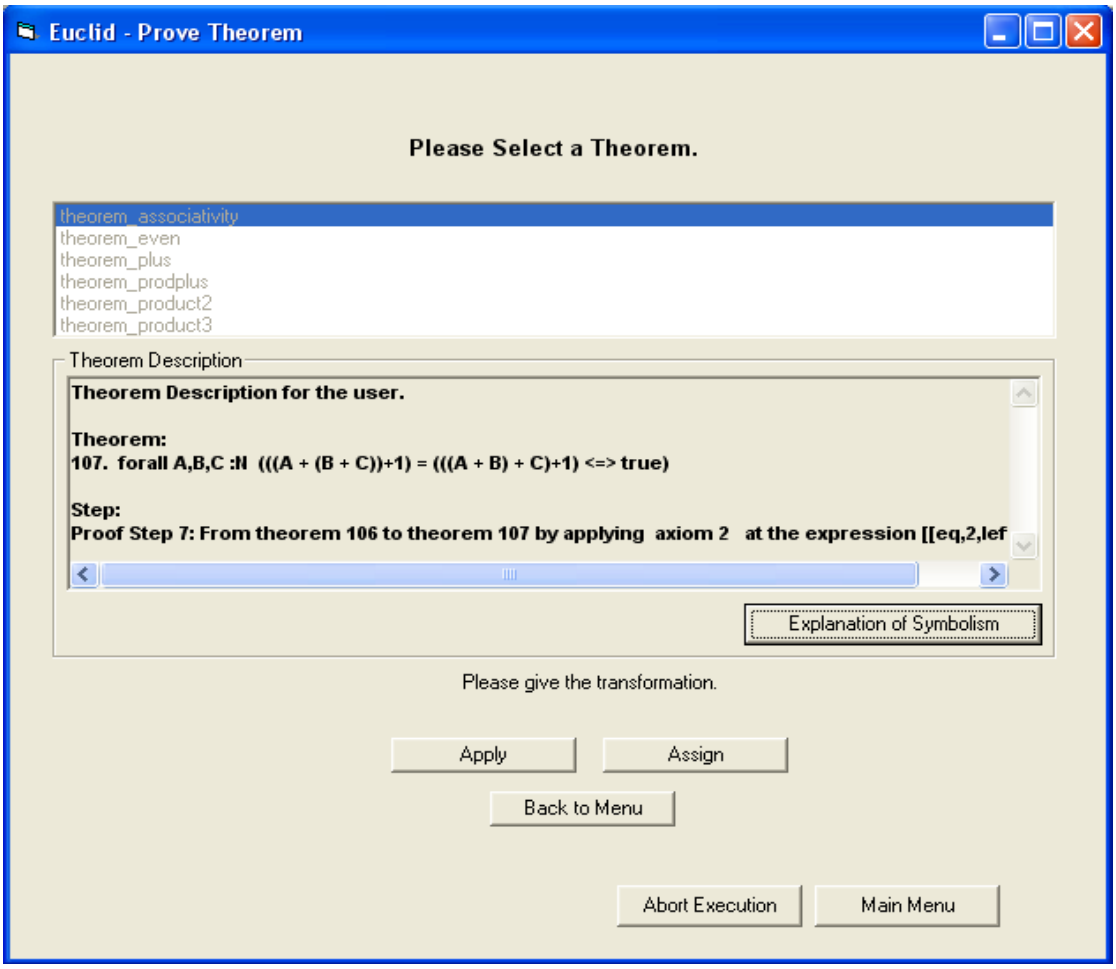
Εικόνα 4.41: *Πλήρες Σενάριο – Επιλογή εκτέλεσης μετασχηματισμού.*



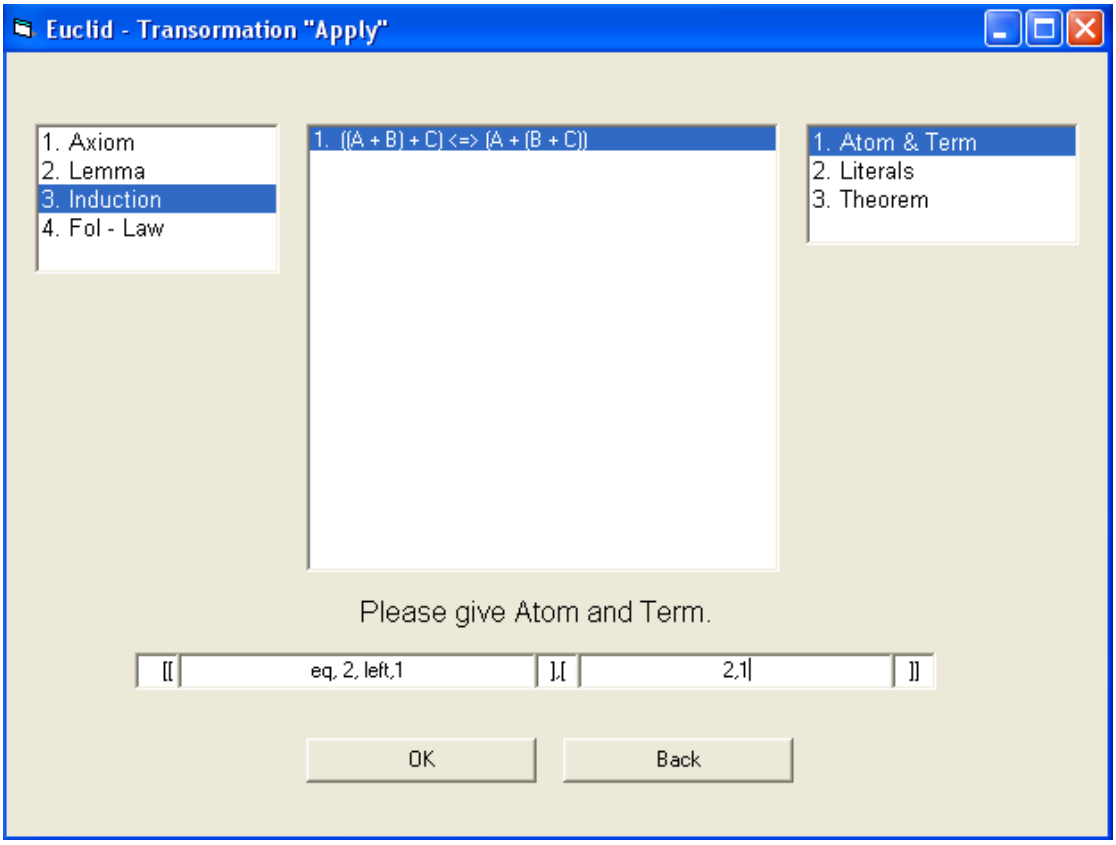
Εικόνα 4.42: *Πλήρες Σενάριο – Μετασχηματισμός "apply".*



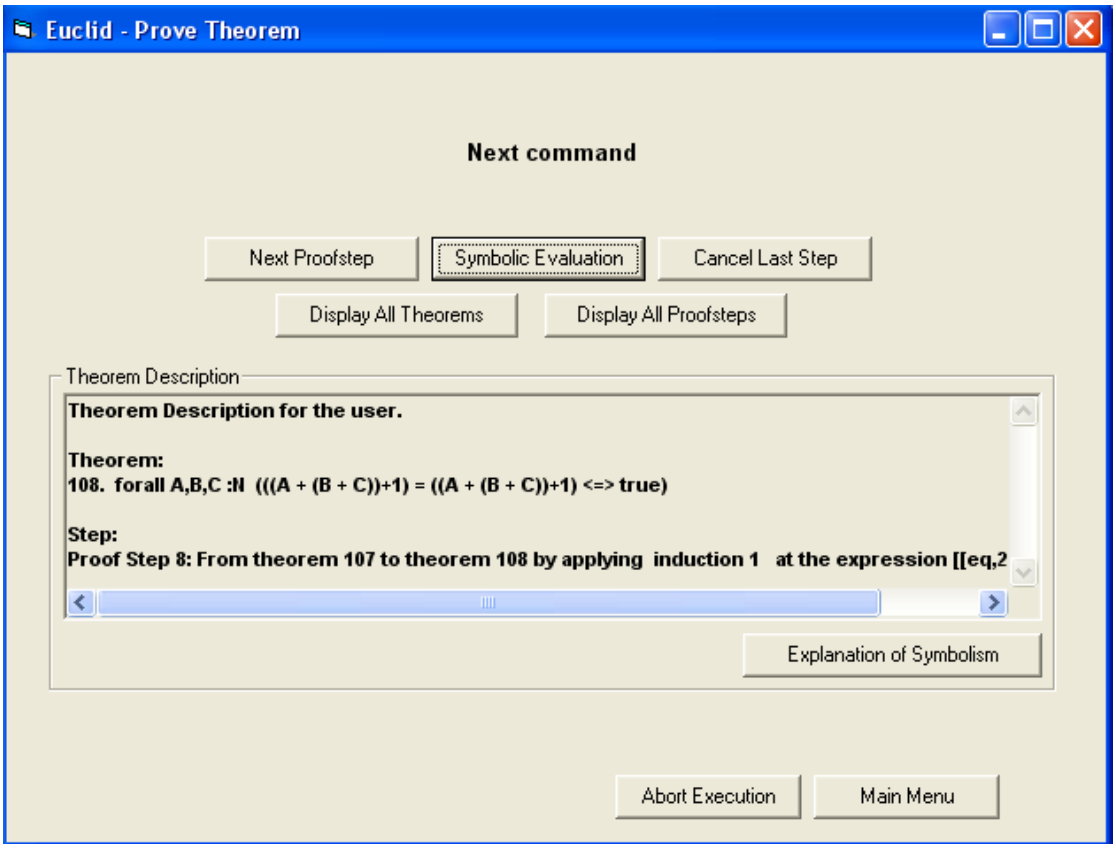
Εικόνα 4.43: *Πλήρες Σενάριο – Επόμενο βήμα απόδειξης.*



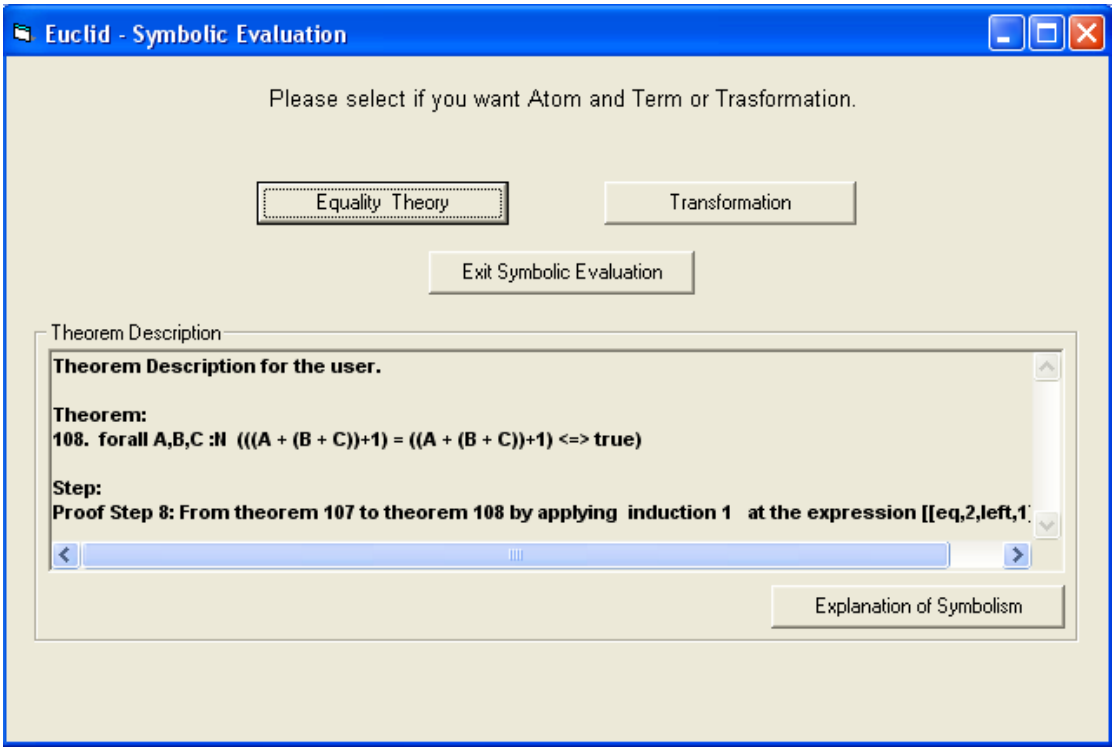
Εικόνα 4.44: *Πλήρες Σενάριο – Επιλογή εκτέλεσης μετασχηματισμού.*



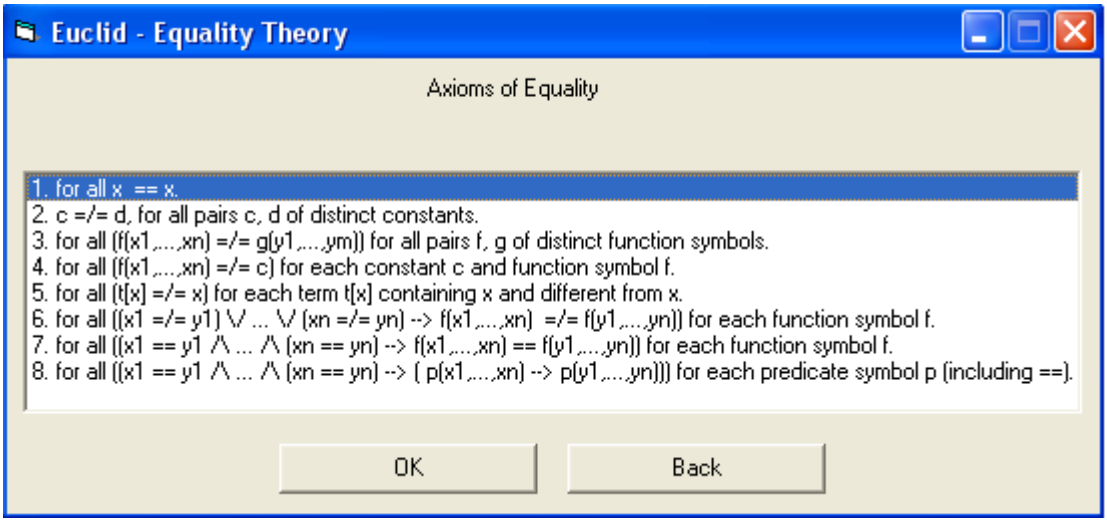
Εικόνα 4.45: *Πλήρες Σενάριο – Μετασχηματισμός "apply".*



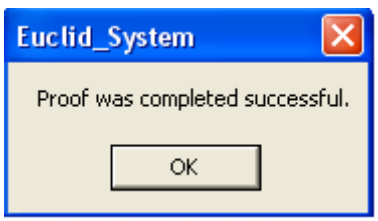
Εικόνα 4.46: *Πλήρες Σενάριο – Επόμενο βήμα απόδειξης.*



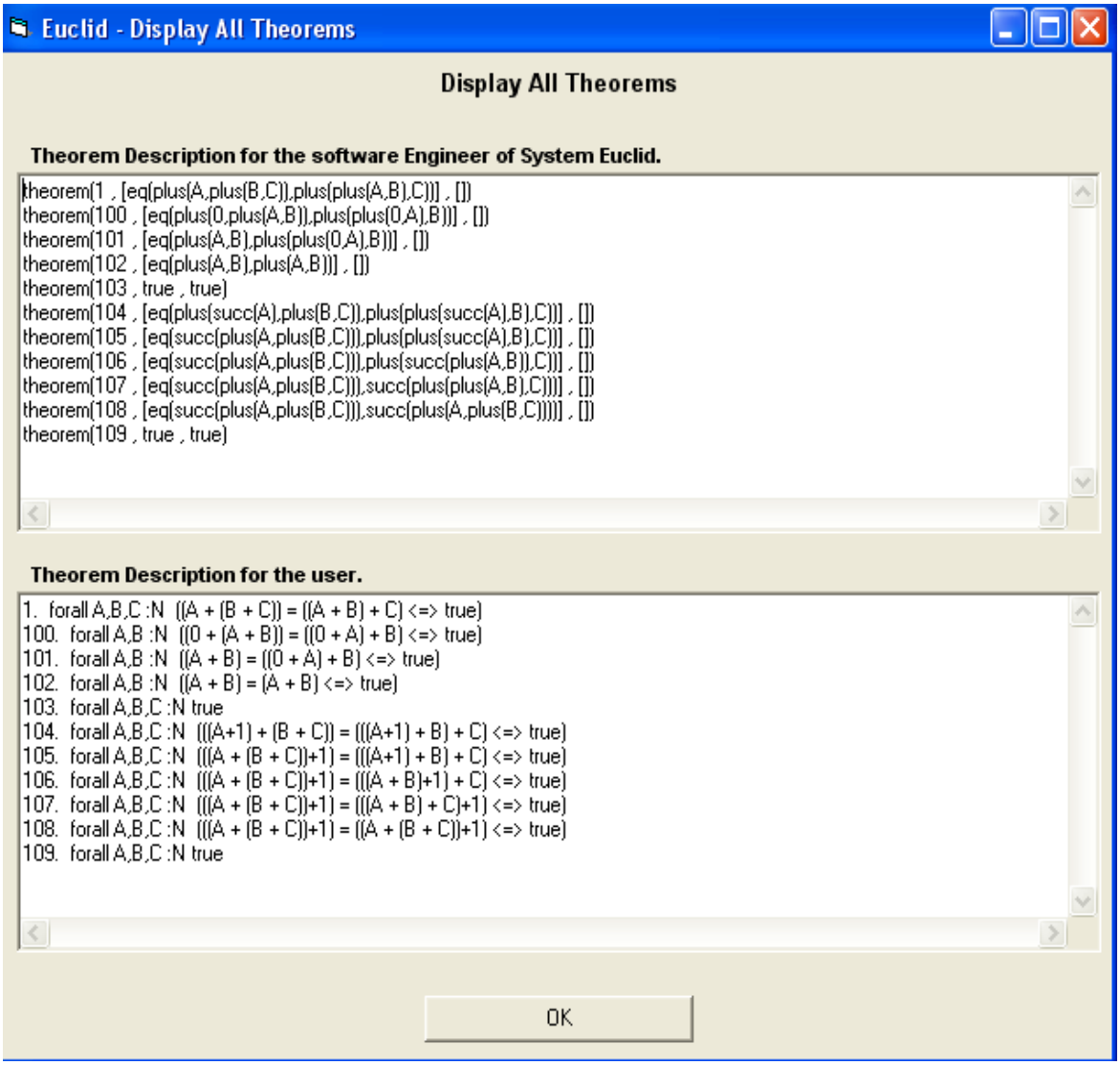
Εικόνα 4.47: *Πλήρες Σενάριο – "Symbolic Evaluation".*



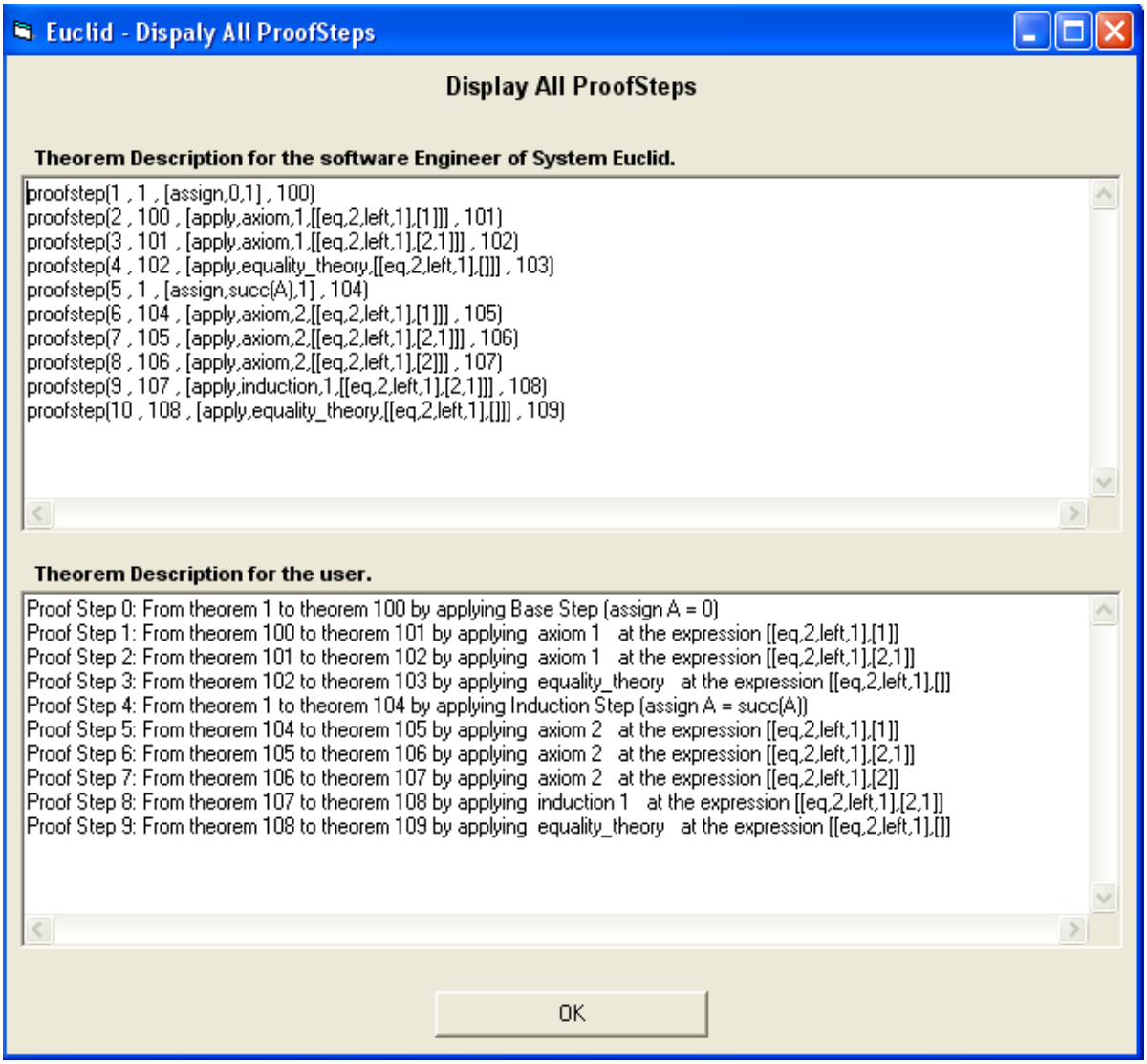
Εικόνα 4.48: *Πλήρες Σενάριο – Αξιώματα της θεωρίας της ισότητας.*



Εικόνα 4.49: *Πλήρες Σενάριο – Ολοκλήρωση απόδειξης βήματος βάσης ή βήματος επαγωγής.*



Εικόνα 4.50: *Πλήρες Σενάριο – Εκτύπωση όλων των θεωρημάτων.*



Εικόνα 4.51: *Πλήρες Σενάριο – Εκτύπωση όλων των βημάτων απόδειξης.*

Με τον ίδιο τρόπο γίνεται η απόδειξη και των υπόλοιπων θεωρημάτων.

5 Συμπεράσματα

Το σύστημα Ευκλείδης είναι ένα διαλογικό σύστημα απόδειξης θεωρημάτων /προτάσεων με επαγωγή. Τα επαγωγικά σύνολα τα οποία υποστηρίζει είναι το σύνολο των φυσικών αριθμών και όλα τα επαγωγικά υποσύνολα των ακέραιων αριθμών. Το σύστημα Ευκλείδης είναι ένα καλό εργαλείο το οποίο μπορεί να χρησιμοποιηθεί για εκπαιδευτικούς σκοπούς από καθηγητές και σπουδαστές. Ο καθηγητής μπορεί να το χρησιμοποιήσει για διδασκαλία της επαγωγής, ενώ ο σπουδαστής για εξάσκηση του, ώστε να εμπεδώσει τη μέθοδο απόδειξης με επαγωγή.

Το σύστημα Ευκλείδης μπορεί να επεκταθεί σε δύο κατευθύνσεις. Μία κατεύθυνση είναι να μπορεί να υποστηρίζει και άλλα επαγωγικά σύνολα εκτός από το σύνολο των φυσικών και υποσύνολα των ακεραίων. Μ' αυτή την επέκταση θα μπορεί ο χρήστης να αποδεικνύει θεωρήματα /προτάσεις και γι' αυτά τα επαγωγικά σύνολα .Δηλαδή, αυξάνεται το πεδίο των θεωρημάτων /προτάσεων που μπορούν ν' αποδειχθούν από το σύστημα. Η άλλη κατεύθυνση είναι η αυτοματοποίησή του. Δηλαδή, η παρέμβαση του χρήστη να ελαχιστοποιηθεί. Το σύστημα ν' αποφασίζει μόνο του ποιο αξίωμα, λήμμα, ή νόμο της λογικής πρώτης τάξης θα εφαρμόζει. Παρέμβαση από το χρήστη να υπάρχει μόνο όταν το σύστημα δε μπορεί να προχωρήσει.

Βιβλιογραφία

Ελληνόγλωσση βιβλιογραφία.

[Κουνάλη 2005]

Κουνάλη Χαρά, *Αυτόματο Σύστημα Μετασχηματισμού Λογικών Προγραμμάτων,* Πτυχιακή εργασία, Τμήμα Εφαρμοσμένης Πληροφορικής και Πολυμέσων, ΤΕΙ Κρήτης, Ηράκλειο 2005.

[Μαρακάκης 2003]

Μαρακάκης Εμμ., *Τεχνητή Νοημοσύνη,* Σημειώσεις Θεωρίας Τεχνητής Νοημοσύνης, ΤΕΙ Κρήτης, Ηράκλειο 2003.

[Μαρακάκης 2004]

Μαρακάκης Εμμ., *Prolog: Προγραμματισμός για Τεχνητή Νοημοσύνη*, Σημειώσεις εργαστηρίου Τεχνητής Νοημοσύνης, ΤΕΙ Κρήτης, Ηράκλειο 2004.

[Μαρακάκης 2003]

Μαρακάκης Εμμ., *Προγραμματισμός ΙΙ,* Σημειώσεις Θεωρίας Προγραμματισμού ΙΙ, Κατασκευή Λογισμικού, ΤΕΙ Κρήτης, Ηράκλειο 2003.

Ξενόγλωσση βιβλιογραφία.

[Bratko 2001]

Bratko Ivan, *Prolog Programming for Artificial Intelligence*, Third edition 2001.

[Cohen 1985]

Cohen Jacques, *Describing Prolog by its Interpretation and Compilation*, Communications of the ACM, December 1985.

[Duffy 1991]

Duffy A. David, *Principles of Automated Theorem Proving*, Scotland 1985.

[Lloyd 1987]

Lloyd J.W., *Foundations of Logic Programming*, Second, Extended Edition 1987.

[Boyer-Moore 1979]

Boyer S. Robert and Moore J. Strother, *A Computational Logic*, Academic Press, 1979.

[Bundy 1988]

Bundy Alan, *The Use of Explicit Plans to Guide Inductive Proofs*, Proceedings of the 9th International Conference on Automated Deduction, 1988.

[Bundy 2001]

Bundy Alan, *The Automation of Proof by Mathematical Induction*, Handbook of Automated Reasoning, 2001.

[Anderson 1979]

Anderson B. Robert, Proving Programs Correct by R. Anderson, 1979.

[Straight- Polimeni 1990]

Polimeni D. Albert and Straight H. Joseph, *Foundations of Discrete Mathematics*, Second Edition, 1990.

[Stanat-McAllister 1977]

Stanat F. Donald and McAllister F. David, *Discrete Mathematics in Computer Science*, 1977.

[SICStus Prolog User's Manual]

By the Intelligent Systems Laboratory Swedish Institute of Computer Science, Swedish Institute of Computer Science, October 2003.

Παραρτήματα

Α Παραδείγματα Απόδειξης με το Σύστημα Ευκλείδης

Α.1 Παραδείγματα με Διεπικοινωνία σε Καταλόγους Επιλογής

Σ' αυτό το τμήμα του παραρτήματος παρουσιάζονται παραδείγματα αποδείξεων με το σύστημα Ευκλείδης, η διεπικοινωνία του οποίου είναι σε μορφή καταλόγων επιλογής. Αυτή η διεπικοινωνία έχει υλοποιηθεί σε Prolog.

Α.1.1 Παράδειγμα 1:Προσεταιριστική Ιδιότητα Φυσικών Αριθμών

```
| ?- prove.

Give the theorem
|: 1.

b for base step
ind for induction step
lth to see theorem
e for end
|: b.

Give the  transformation
|: assign.

 Base  Transf:assignTHEOREM
Theorem Left: [eq(plus(A,plus(B,C)),plus(plus(A,B),C))]
Theorem Right: []

THEOREM VARIABLES
Variables:[A,B,C]

Give variable order e.g. 1,2,3..
|: 1.
Give variable value:0.
**********
THEOREM
write Left Theorem: [eq(plus(0,plus(A,B)),plus(plus(0,A),B))]
write Right Theorem: []

THEOREM VARIABLES
write: [A,B]
```

```
lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
tr for more transformations
c for cancelling last step
s for symbolic_evaluation
e for end
|: tr.

Give the  transformation
|: apply(axiom,1,[[eq, 2, left,1], [1]]).

 Base  Transf:apply(axiom,1,[[eq,2,left,1],[1]])
lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
tr for more transformations
c for cancelling last step
s for symbolic_evaluation
e for end
|: tr.

Give the  transformation
|: apply(axiom,1,[[eq, 2, left,1], [2,1]]).

 Base  Transf:apply(axiom,1,[[eq,2,left,1],[2,1]])
lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
tr for more transformations
c for cancelling last step
s for symbolic_evaluation
e for end
|: s.

Dose Atom and Term or Transformation
|: [[eq,2,left,1],[]].

lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
c for cancelling last step
s for further symbolic evaluation
e for end
|: e.

p  prove other theorem
```

```
e for end
|: p.

Give the theorem
|: 1.

b for base step
ind for induction step
lth to see theorem
e for end
|: ind.

Give the  transformation
|: assign.

 Base  Transf:assignTHEOREM
Theorem Left: [eq(plus(A,plus(B,C)),plus(plus(A,B),C))]
Theorem Right: []

THEOREM VARIABLES
Variables:[A,B,C]

Give variable order e.g. 1,2,3..
|: 1.
Give variable value:succ(X).
**********
THEOREM
write Left Theorem: [eq(plus(succ(A),plus(B,C)),plus(plus(succ(A),B),C))]
write Right Theorem: []

THEOREM VARIABLES
write: [A,B,C]

lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
tr for more transformations
c for cancelling last step
s for symbolic_evaluation
e for end
|: tr.

Give the  transformation
|: apply(axiom,2,[[eq, 2, left,1], [1]]).

 Base  Transf:apply(axiom,2,[[eq,2,left,1],[1]])
lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
```

tr for more transformations
c for cancelling last step
s for symbolic_evaluation
e for end
|: tr.

Give the transformation
|: apply(axiom,2,[[eq, 2, left,1], [2,1]]).

 Base Transf:apply(axiom,2,[[eq,2,left,1],[2,1]])
lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
tr for more transformations
c for cancelling last step
s for symbolic_evaluation
e for end
|: tr.

Give the transformation
|: apply(axiom,2,[[eq, 2, left,1], [2]]).

 Base Transf:apply(axiom,2,[[eq,2,left,1],[2]])
lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
tr for more transformations
c for cancelling last step
s for symbolic_evaluation
e for end
|: tr.

Give the transformation
|: apply(induction,1,[[eq, 2, left,1], [2,1]]).

 Base Transf:apply(induction,1,[[eq,2,left,1],[2,1]])
lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
tr for more transformations
c for cancelling last step
s for symbolic_evaluation
e for end
|: s.

Dose Atom and Term or Transformation
|: [[eq,2,left,1],[]].

```
lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
c for cancelling last step
s for further symbolic evaluation
e for end
|: lth.

Theorem(109,true,true).
lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
c for cancelling last step
s for further symbolic evaluation
e for end
|: lps.

Proofstep(10,108,[apply,equality_theory,[[eq,2,left,1],[]]],109).

lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
c for cancelling last step
s for further symbolic evaluation
e for end
|: allth.

Theorem(1,[eq(plus(A,plus(B,C)),plus(plus(A,B),C))],[]).
Theorem(100,[eq(plus(0,plus(A,B)),plus(plus(0,A),B))],[]).
Theorem(101,[eq(plus(A,B),plus(plus(0,A),B))],[]).
Theorem(102,[eq(plus(A,B),plus(A,B))],[]).
Theorem(103,true,true).
Theorem(104,[eq(plus(succ(A),plus(B,C)),plus(plus(succ(A),B),C))],[]).
Theorem(105,[eq(succ(plus(A,plus(B,C))),plus(plus(succ(A),B),C))],[]).
Theorem(106,[eq(succ(plus(A,plus(B,C))),plus(succ(plus(A,B)),C))],[]).
Theorem(107,[eq(succ(plus(A,plus(B,C))),succ(plus(plus(A,B),C)))],[]).
Theorem(108,[eq(succ(plus(A,plus(B,C))),succ(plus(A,plus(B,C))))],[]).
Theorem(109,true,true).

lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
c for cancelling last step
s for further symbolic evaluation
e for end
|: allps.
```

```
Proofstep(1,1,[assign,0,1],100).
Proofstep(2,100,[apply,axiom,1,[[eq,2,left,1],[1]]],101).
Proofstep(3,101,[apply,axiom,1,[[eq,2,left,1],[2,1]]],102).
Proofstep(4,102,[apply,equality_theory,[[eq,2,left,1],[]]],103).
Proofstep(5,1,[assign,succ(_35047),1],104).
Proofstep(6,104,[apply,axiom,2,[[eq,2,left,1],[1]]],105).
Proofstep(7,105,[apply,axiom,2,[[eq,2,left,1],[2,1]]],106).
Proofstep(8,106,[apply,axiom,2,[[eq,2,left,1],[2]]],107).
Proofstep(9,107,[apply,induction,1,[[eq,2,left,1],[2,1]]],108).
Proofstep(10,108,[apply,equality_theory,[[eq,2,left,1],[]]],109).
lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
c for cancelling last step
s for further symbolic evaluation
e for end
|: e.

p  prove other theorem
e for end
|: e.
yes
```

Α.1.2 Παράδειγμα 2:Αντιμεταθετική Ιδιότητα Φυσικών Αριθμών

```
| ?- prove.

Give the theorem
|: 2.

b for base step
ind for induction step
lth to see theorem
e for end
|: b.

Give the transformation
|: assign.

 Base Transf: assignTHEOREM
Theorem Left: [even (A), even (B)]
Theorem Right: [even (plus (A, B))]

THEOREM VARIABLES
Variables: [A, B, A, B]

Give variable order e.g. 1, 2, 3…
```

|: 1.
Give variable value: 0.
****

lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
tr for more transformations
c for cancelling last step
s for symbolic_evaluation
e for end
|: tr.

Give the transformation
|: apply (axiom, 3, [[even, 1, left, 1], [0]]).

Base Transf: apply (axiom, 3, [[even, 1, left, 1], [0]])
lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
tr for more transformations
c for cancelling last step
s for symbolic_evaluation
e for end
|: tr.

Give the transformation
|: apply (axiom, 1, [[even, 1, right, 1], [1]]).

Base Transf: apply (axiom, 1, [[even, 1, right, 1], [1]])
lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
tr for more transformations
c for cancelling last step
s for symbolic_evaluation
e for end
|: s.

Dose Atom and Term or Transformation
|: apply (fol_law, 1, [left, [1, 2]]).

lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
c for cancelling last step
s for further symbolic evaluation

e for end
|: s.

Dose Atom and Term or Transformation
|: [[even, 1, left, 1], [0]].

lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
c for cancelling last step
s for further symbolic evaluation
e for end
|: allth.

Theorem (2, [even (A), even (B)], [even (plus (A, B))]).
Theorem (100, [even (0), even (A)], [even (plus (0, A))]).
Theorem (101, [true, even (A)], [even (plus (0, A))]).
Theorem (102, [true, even (A)], [even (A)]).
Theorem (103, [even (A)], [even (A)]).
Theorem (104, true, true).

lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
c for cancelling last step
s for further symbolic evaluation
e for end
|: allps.

Proofstep (1, 2, [assign, 0, 1], 100).
Proofstep (2, 100, [apply,axiom,3,[[even,1,left,1],[0]]],101).
Proofstep (3, 101, [apply, axiom, 1, [[even, 1, right, 1], [1]]], 102).
Proofstep (4, 102, [apply, fol_law, 1, [left, [1, 2]]], 103).
Proofstep (5, 103, [apply, equality_theory, [[even, 1, left, 1], [0]]], 104).

lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
c for cancelling last step
s for further symbolic evaluation
e for end
|: e.

p prove other theorem
e for end
|: p.

Give the theorem

```
|: 2.

b for base step
ind for induction step
lth to see theorem
e for end
|: b.

Give the transformation
|: assign.

 Base Transf: assignTHEOREM
Theorem Left: [even (A), even (B)]
Theorem Right: [even (plus (A, B))]

THEOREM VARIABLES
Variables: [A, B, A, B]

Give variable order e.g. 1, 2, 3…
|: 1.
Give variable value:0.
**********

lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
tr for more transformations
c for cancelling last step
s for symbolic_evaluation
e for end
|: c.

lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
tr for more transformations
c for cancelling last step
s for symbolic_evaluation
e for end
|: tr.

Give the transformation
|: assign.

 Base Transf: assignTHEOREM
Theorem Left: [even (A), even (B)]
Theorem Right: [even (plus (A, B))]

THEOREM VARIABLES
```

Variables: [A, B, A, B]

Give variable order e.g. 1, 2, 3...
|: 1.
Give variable value: succ(0).
****

lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
tr for more transformations
c for cancelling last step
s for symbolic_evaluation
e for end
|: tr.

Give the transformation
|: apply (axiom, 4, [left, [1]]).

Base Transf: apply (axiom, 4, [left, [1]])

lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
tr for more transformations
c for cancelling last step
s for symbolic_evaluation
e for end
|: s.

Dose Atom and Term or Transformation
|: apply (fol_law, 3, [[left, [1]], [right, [1]]]).

lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
c for cancelling last step
s for further symbolic evaluation
e for end
|: allth.

Theorem (2, [even (A), even (B)], [even (plus (A, B))]).
Theorem (100, [even (0), even (A)], [even (plus (0, A))]).
Theorem (101, [true, even (A)], [even (plus (0, A))]).
Theorem (102, [true, even (A)], [even (A)]).
Theorem (103, [even (A)], [even (A)]).
Theorem (104, true, true).
Theorem(105, [even(succ(0)),even(A)],[even(plus(succ(0),A))]).

```
Theorem (106, [even (A), false], [even (plus (succ (0), A))]).
Theorem (107, [false], [even (plus (succ (0), A))]).
Theorem (108, true, true).

lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
c for cancelling last step
s for further symbolic evaluation
e for end
|: allps.

Proofstep (1, 2, [assign, 0, 1], 100).
Proofstep (2, 100, [apply, axiom, 3, [[even, 1, left, 1], [0]]], 101).
Proofstep (3, 101, [apply, axiom, 1, [[even, 1, right, 1], [1]]], 102).
Proofstep (4, 102, [apply, fol_law, 1, [left, [1, 2]]], 103).
Proofstep (5, 103, [apply, equality_theory, [[even, 1, left, 1], [0]]], 104).
Proofstep (6, 2, [assign, succ (0), 1], 105).
Proofstep (7, 105, [apply, axiom, 4, [left, [1]]], 106).
Proofstep (8, 106, [apply, fol_law, 2, [left, [1, 2]]], 107).
Proofstep (9, 107, [apply, fol_law, 3, [[left, [1]], [right, [1]]]], 108).

lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
c for cancelling last step
s for further symbolic evaluation
e for end
|:  e.

p  prove other theorem
e for end
|: p.

Give the theorem
|: 2.

b for base step
ind for induction step
lth to see theorem
e for end
|: ind.

Give the transformation
|: assign.

 Base Transf: assignTHEOREM
Theorem Left: [even (A), even (B)]
Theorem Right: [even (plus (A, B))]
```

THEOREM VARIABLES
Variables: [A, B, A, B]

Give variable order e.g. 1, 2, 3…
|: 1.
Give variable value: succ(X).
****

lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
tr for more transformations
c for cancelling last step
s for symbolic_evaluation
e for end
|: tr.

Give the transformation
|: apply (axiom, 5, [[even, 1, left, 1], [0]]).

Base Transf: apply (axiom, 5, [[even, 1, left, 1], [0]])
lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
tr for more transformations
c for cancelling last step
s for symbolic_evaluation
e for end
|: tr.

Give the transformation
|: apply (axiom, 2, [[even, 1, right, 1], [1]]).

Base Transf: apply (axiom, 2, [[even, 1, right, 1], [1]])
lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
tr for more transformations
c for cancelling last step
s for symbolic_evaluation
e for end
|: tr.

Give the transformation
|: apply (axiom, 2, [[even, 1, right, 1], [1, 1]]).

Base Transf: apply (axiom, 2, [[even, 1, right, 1], [1, 1]])

```
lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
tr for more transformations
c for cancelling last step
s for symbolic_evaluation
e for end
|: tr.

Give the transformation
|: apply (axiom, 5, [[even, 1, right, 1], [0]]).

 Base Transf: apply (axiom, 5, [[even, 1, right, 1], [0]])
lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
tr for more transformations
c for cancelling last step
s for symbolic_evaluation
e for end
|: tr.

Give the transformation
|: apply (induction, 2, [left, [1, 2]]).

 Base Transf: apply (induction, 2, [left, [1, 2]])
lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
tr for more transformations
c for cancelling last step
s for symbolic_evaluation
e for end
|: s.

Dose Atom and Term or Transformation
|:  [[even, 1, left, 1], [0]].
lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
c for cancelling last step
s for further symbolic evaluation
e for end
|: allth.

Theorem (2, [even (A), even (B)], [even (plus (A, B))]).
Theorem (100, [even (0), even (A)], [even (plus (0, A))]).
```

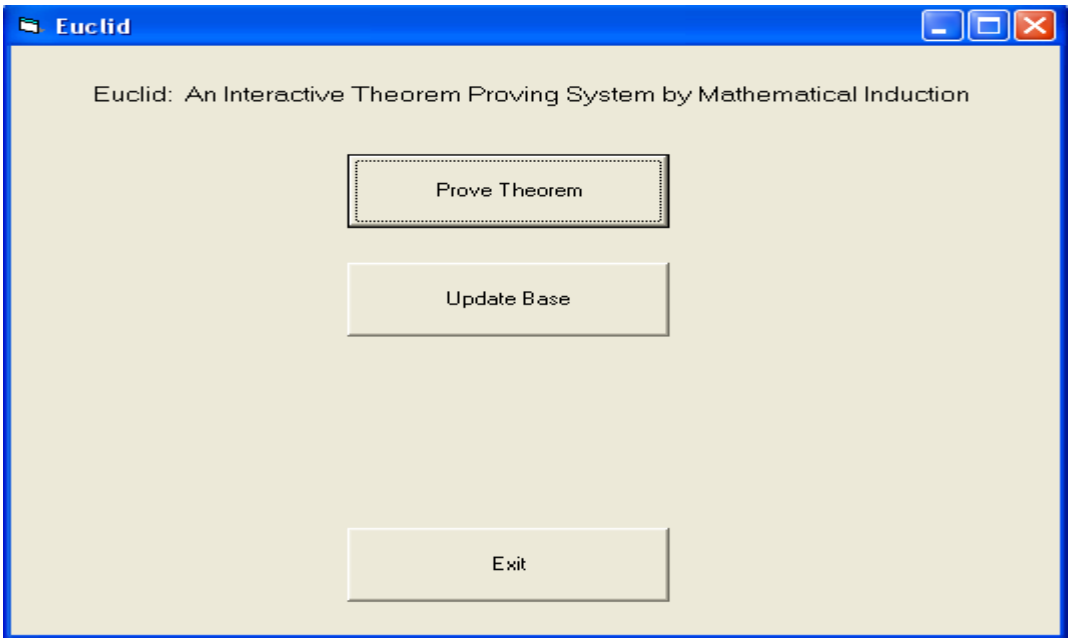
```
Theorem (101, [true, even (A)], [even (plus (0, A))]).
Theorem (102, [true, even (A)], [even (A)]).
Theorem (103, [even (A)], [even (A)]).
Theorem (104, true, true).
Theorem (105, [even (succ (0)), even (A)], [even (plus (succ (0), A))]).
Theorem (106, [even (A), false], [even (plus (succ (0), A))]).
Theorem (107, [false], [even (plus (succ (0), A))]).
Theorem (108, true, true).
Theorem (109, [even (succ (A)), even (B)], [even (plus (succ (A), B))]).
Theorem (110, [even (A), even (B)], [even (plus (succ (succ (A)), B))]).
Theorem (111, [even (A), even (B)], [even (succ (plus (succ (A), B)))]).
Theorem (112, [even (A), even (B)], [even (succ (succ (plus (A, B))))]).
Theorem (113, [even (A), even (B)], [even (plus (A, B))]).
Theorem (114, [even (plus (A, B))], [even (plus (A, B))]).
Theorem (115, true, true).

lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
c for cancelling last step
s for further symbolic evaluation
e for end
|: allps.

Proofstep (1, 2, [assign, 0, 1], 100).
Proofstep (2, 100, [apply, axiom, 3, [[even, 1, left, 1], [0]]], 101).
Proofstep (3, 101, [apply, axiom, 1, [[even, 1, right, 1], [1]]], 102).
Proofstep (4, 102, [apply, fol_law, 1, [left, [1, 2]]], 103).
Proofstep (5, 103, [apply, equality_theory, [[even, 1, left, 1], [0]]], 104).
Proofstep (6, 2, [assign, succ (0), 1], 105).
Proofstep (7, 105, [apply, axiom, 4, [left, [1]]], 106).
Proofstep (8, 106, [apply, fol_law, 2, [left, [1, 2]]], 107).
Proofstep (9, 107, [apply, fol_law, 3, [[left, [1]], [right, [1]]]], 108).
Proofstep (10, 2, [assign, succ (X), 1], 109).
Proofstep (11, 109, [apply, axiom, 5, [[even, 1, left, 1], [0]]], 110).
Proofstep (12, 110, [apply, axiom, 2, [[even, 1, right, 1], [1]]], 111).
Proofstep (13, 111, [apply, axiom, 2, [[even, 1, right, 1], [1, 1]]], 112).
Proofstep (14, 112, [apply, axiom, 5, [[even, 1, right, 1], [0]]], 113).
Proofstep (15, 113, [apply, induction, 2, [left, [1, 2]]], 114).
Proofstep (16, 114, [apply, equality_theory, [[even, 1, left, 1], [0]]], 115).

lth to see last theorem
lps to see last proofstep
allth to see all theorems
allps to see all proofsteps
c for cancelling last step
s for further symbolic evaluation
e for end
|: e.
```

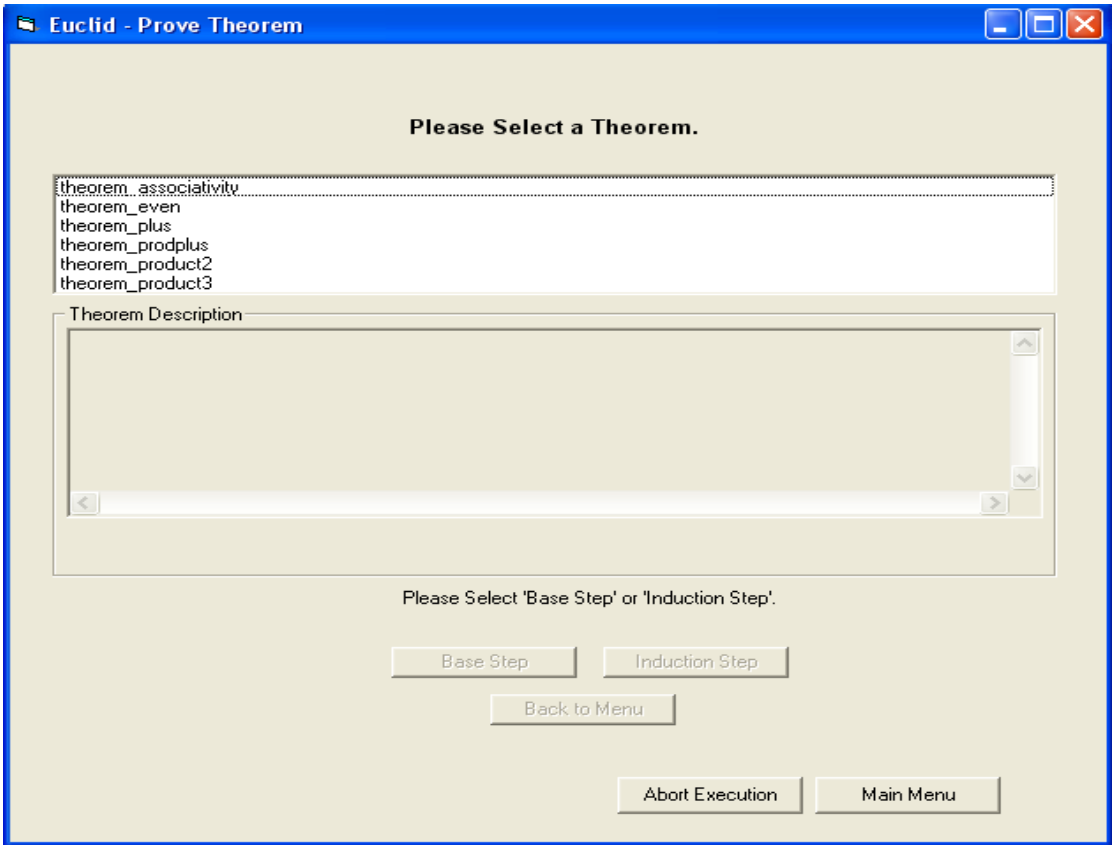
p prove other theorem
e for end
|: e.
yes

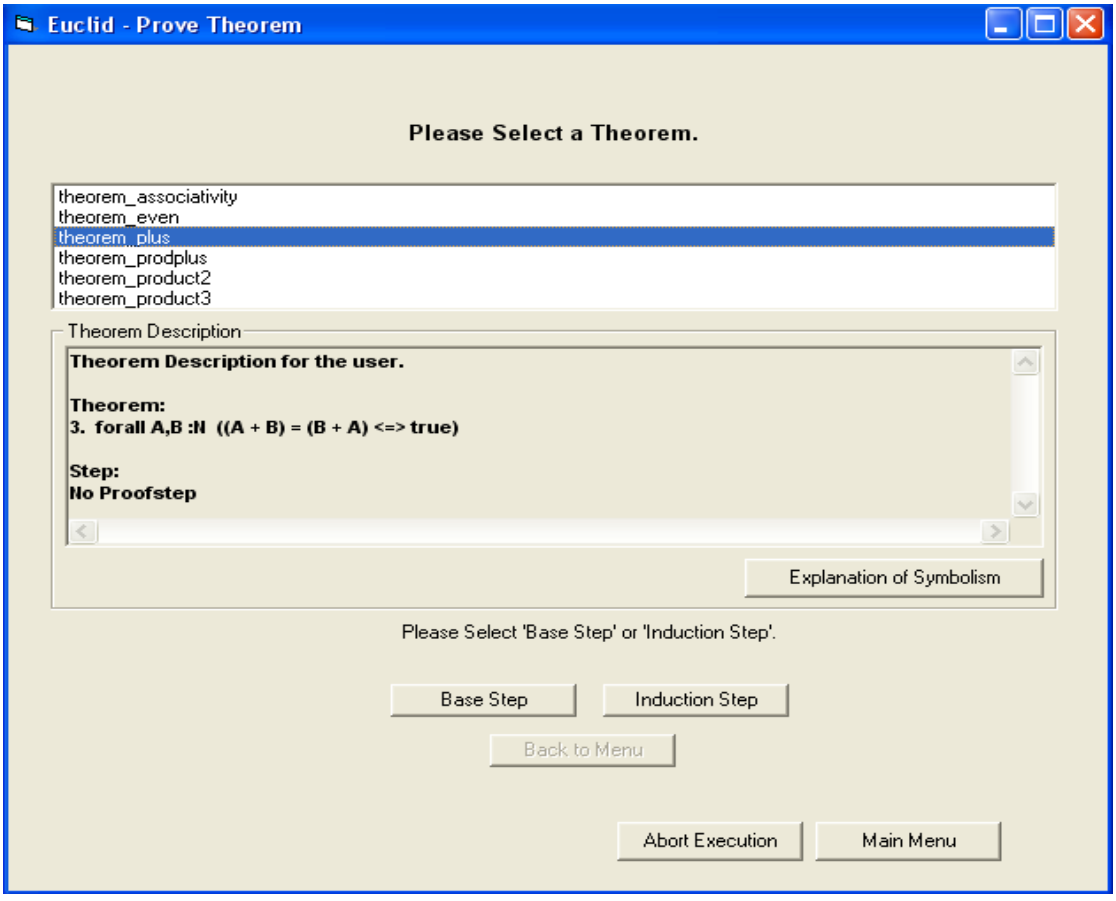
Α.2 Παράδειγμα με Διεπικοινωνία σε Παραθυρικό Περιβάλλον

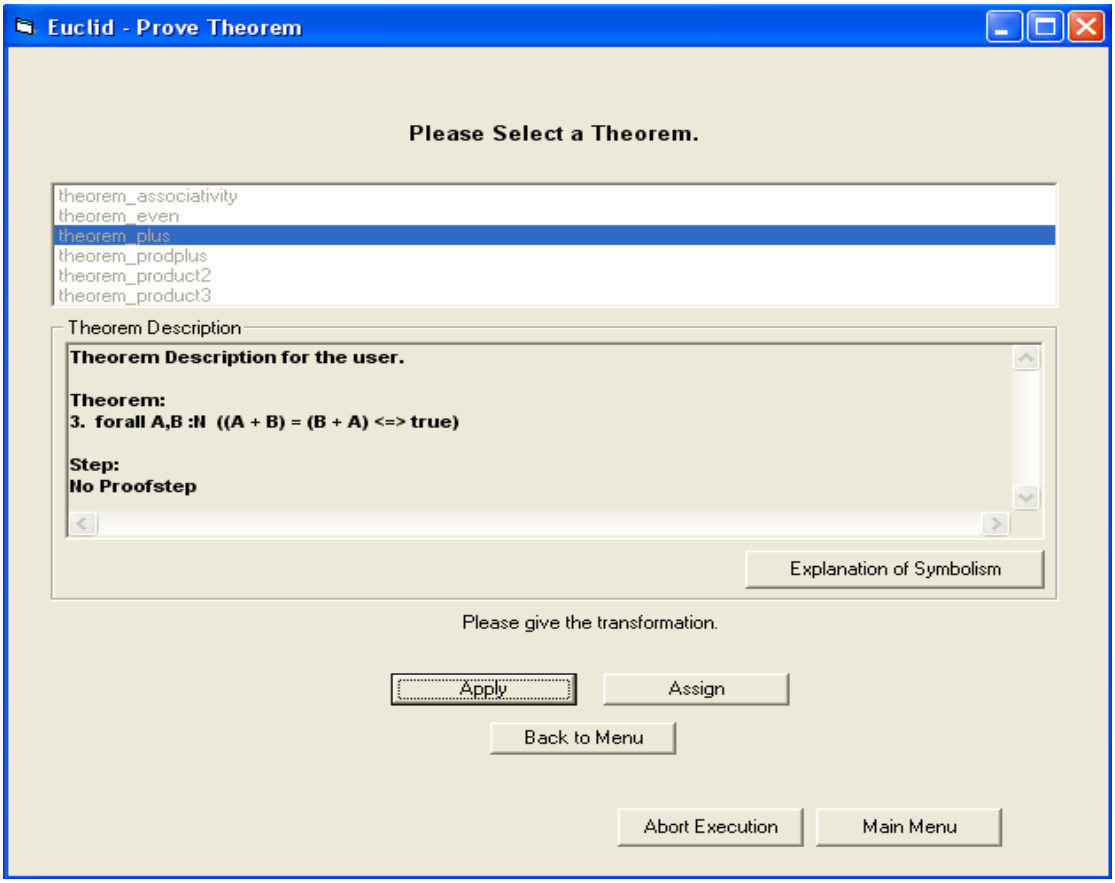
Σ' αυτό το τμήμα του παραρτήματος παρουσιάζονται παραδείγματα αποδείξεων με το σύστημα Ευκλείδης, η διεπικοινωνία του οποίου είναι σε παραθυρικό περιβάλλον. Αυτή η διεπικοινωνία έχει υλοποιηθεί σε Visual Basic.

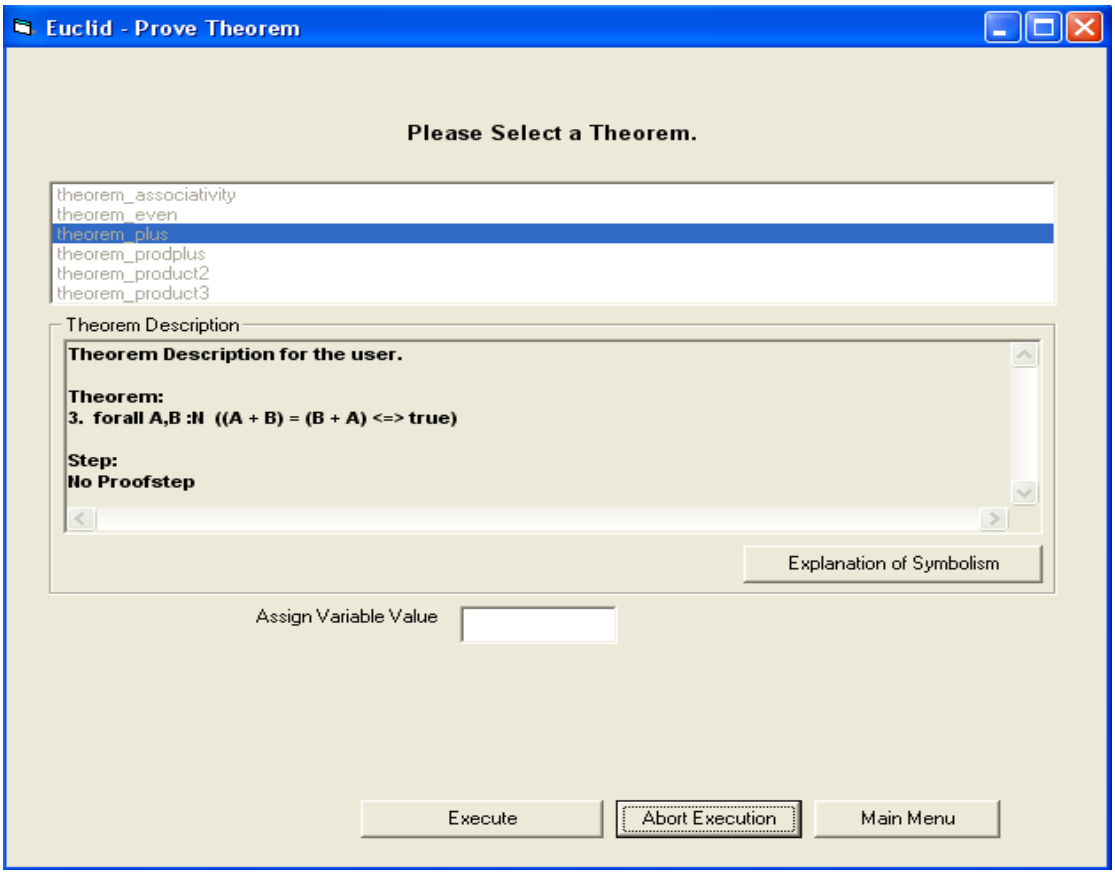
Α.2.1 Παράδειγμα 1: Αντιμεταθετική Ιδιότητα Φυσικών Αριθμών

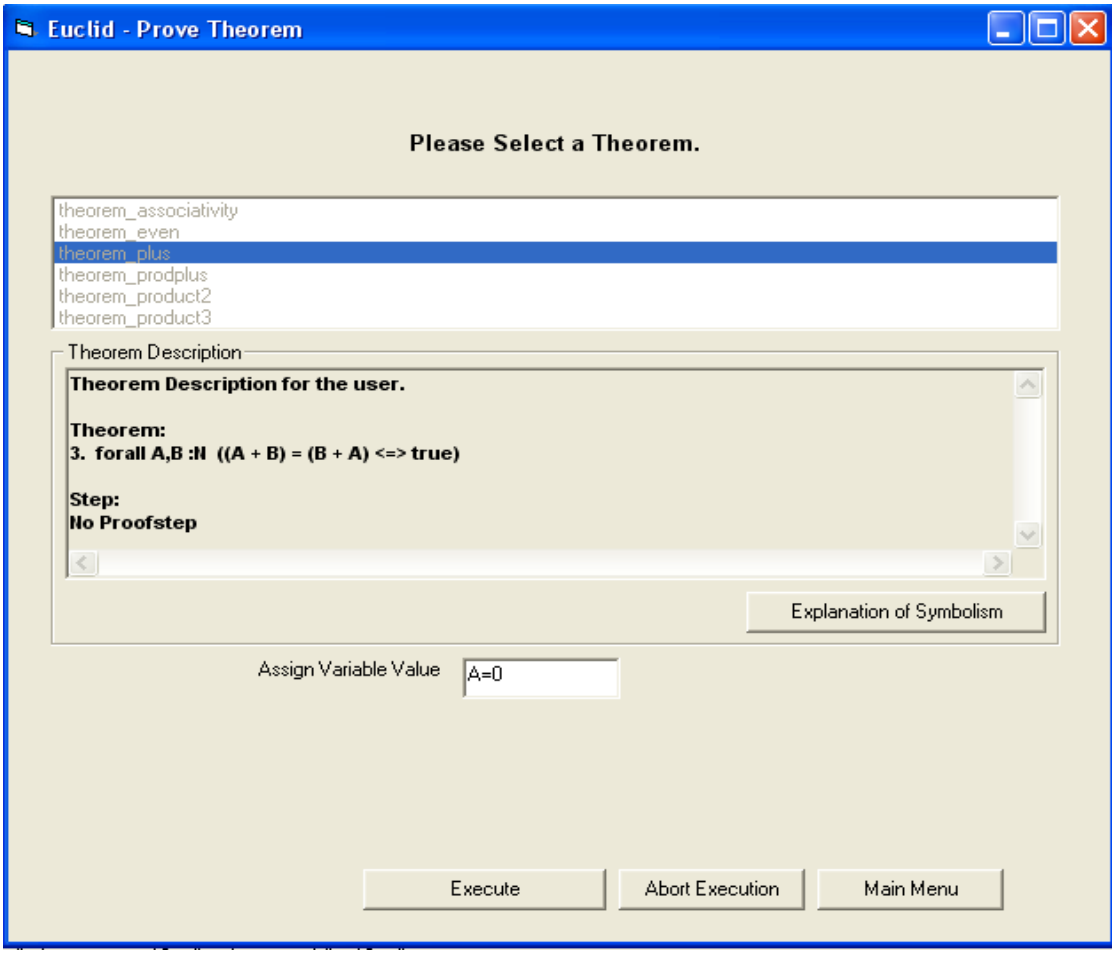


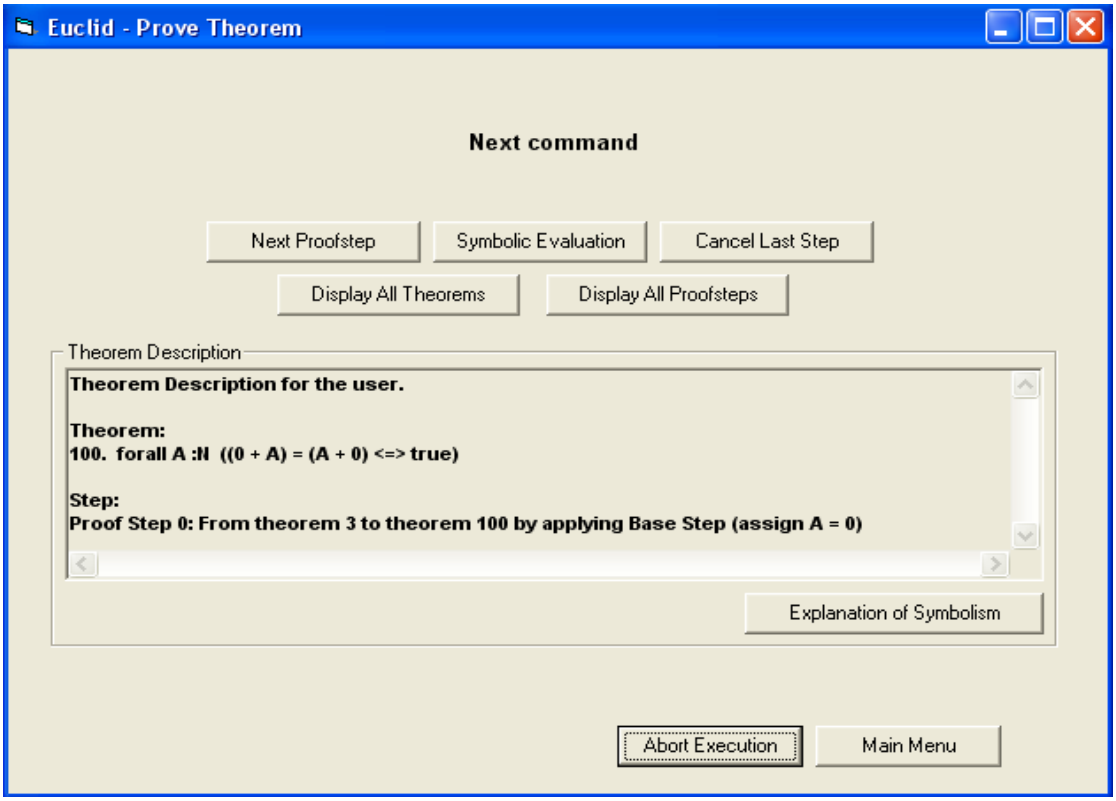

Euclid - Prove Theorem
Please Select a Theorem.
theorem_associativity
theorem_even
theorem_plus
theorem_prodplus
theorem_product2
theorem_product3
Theorem Description
Please Select 'Base Step' or 'Induction Step'.
Base Step
Induction Step
Back to Menu
Abort Execution
Main Menu

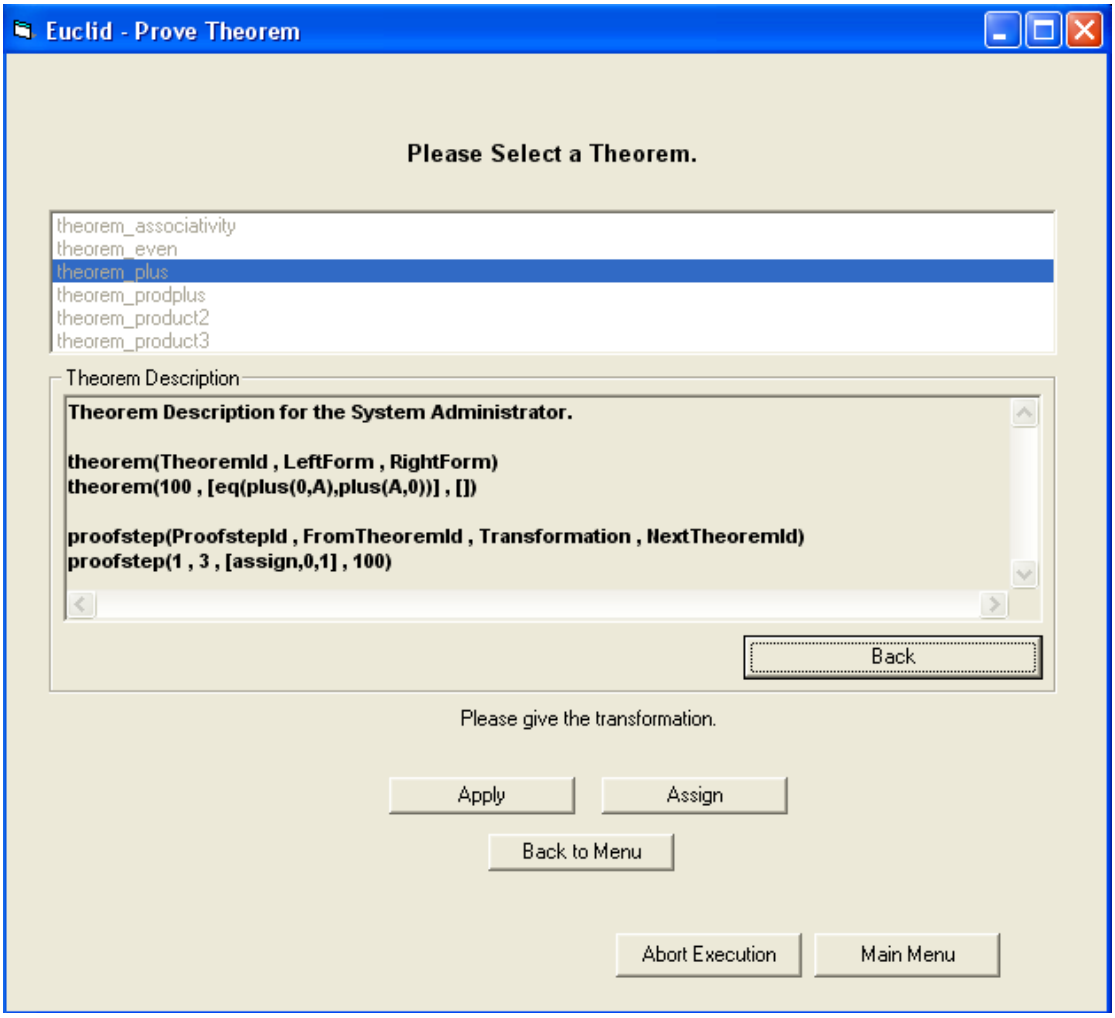

Euclid - Prove Theorem
Please Select a Theorem.
theorem_associativity
theorem_even
theorem_plus
theorem_prodplus
theorem_product2
theorem_product3
Theorem Description
Theorem Description for the user.
Theorem:
3. forall A,B :N ((A + B) = (B + A) <=> true)
Step:
No Proofstep
Explanation of Symbolism
Please Select 'Base Step' or 'Induction Step'.
Base Step
Induction Step
Back to Menu
Abort Execution
Main Menu

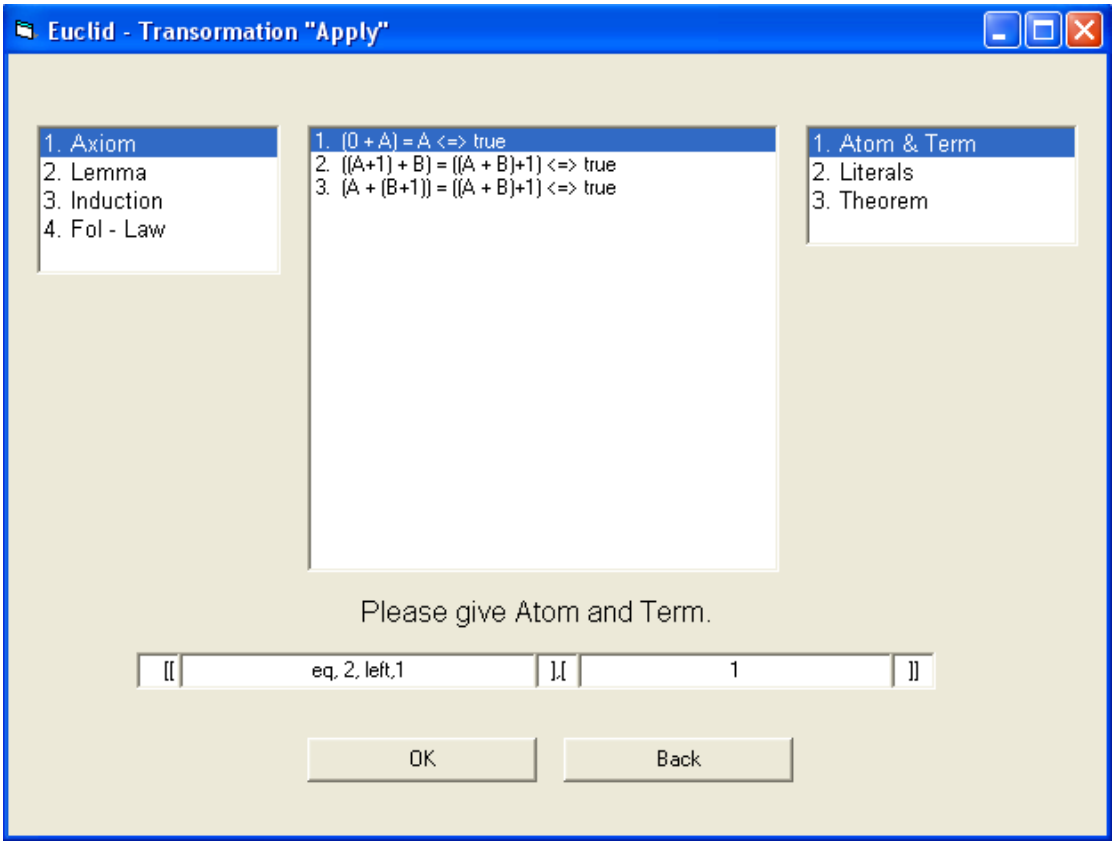

Euclid - Prove Theorem
Please Select a Theorem.
theorem_associativity
theorem_even
theorem_plus
theorem_prodplus
theorem_product2
theorem_product3
Theorem Description
Theorem Description for the user.
Theorem:
3. forall A,B :N ((A + B) = (B + A) <=> true)
Step:
No Proofstep
Explanation of Symbolism
Please give the transformation.
Apply
Assign
Back to Menu
Abort Execution
Main Menu

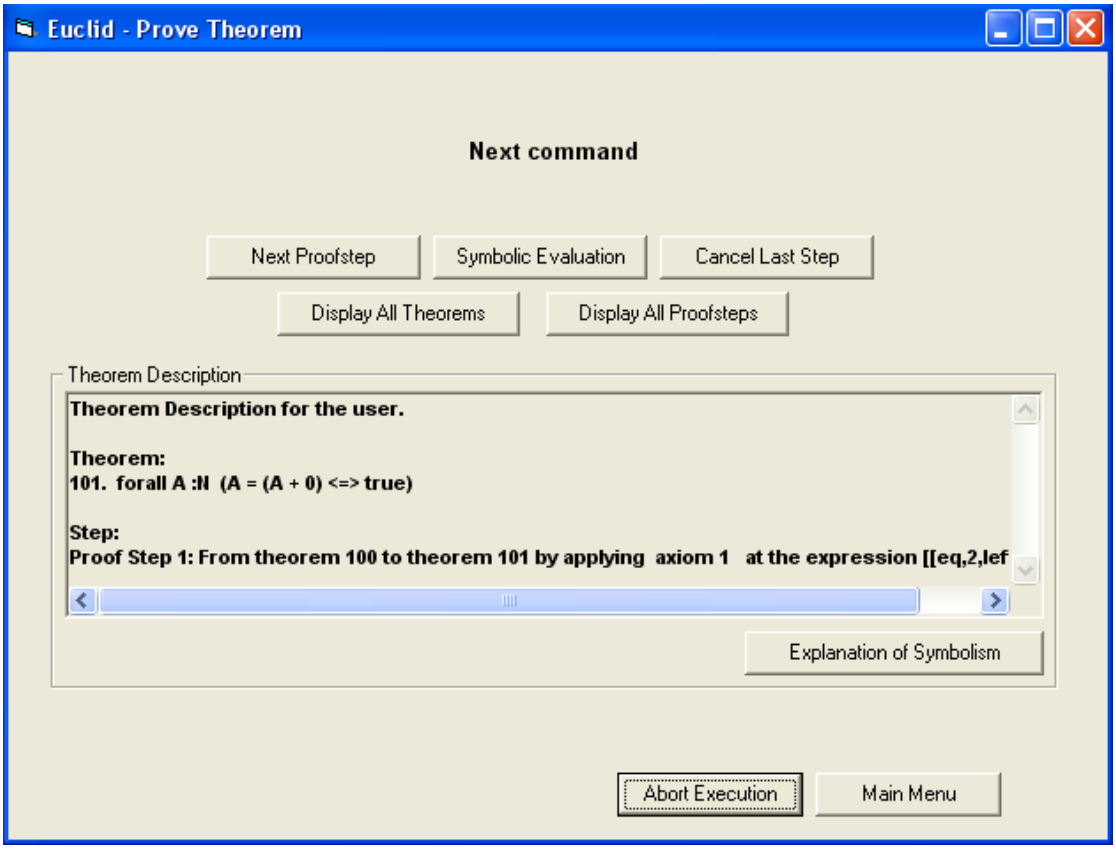

Euclid - Prove Theorem
Please Select a Theorem.
theorem_associativity
theorem_even
theorem_plus
theorem_prodplus
theorem_product2
theorem_product3
Theorem Description
Theorem Description for the user.
Theorem:
3. forall A,B :N ((A + B) = (B + A) <=> true)
Step:
No Proofstep
Explanation of Symbolism
Assign Variable Value
Execute
Abort Execution
Main Menu

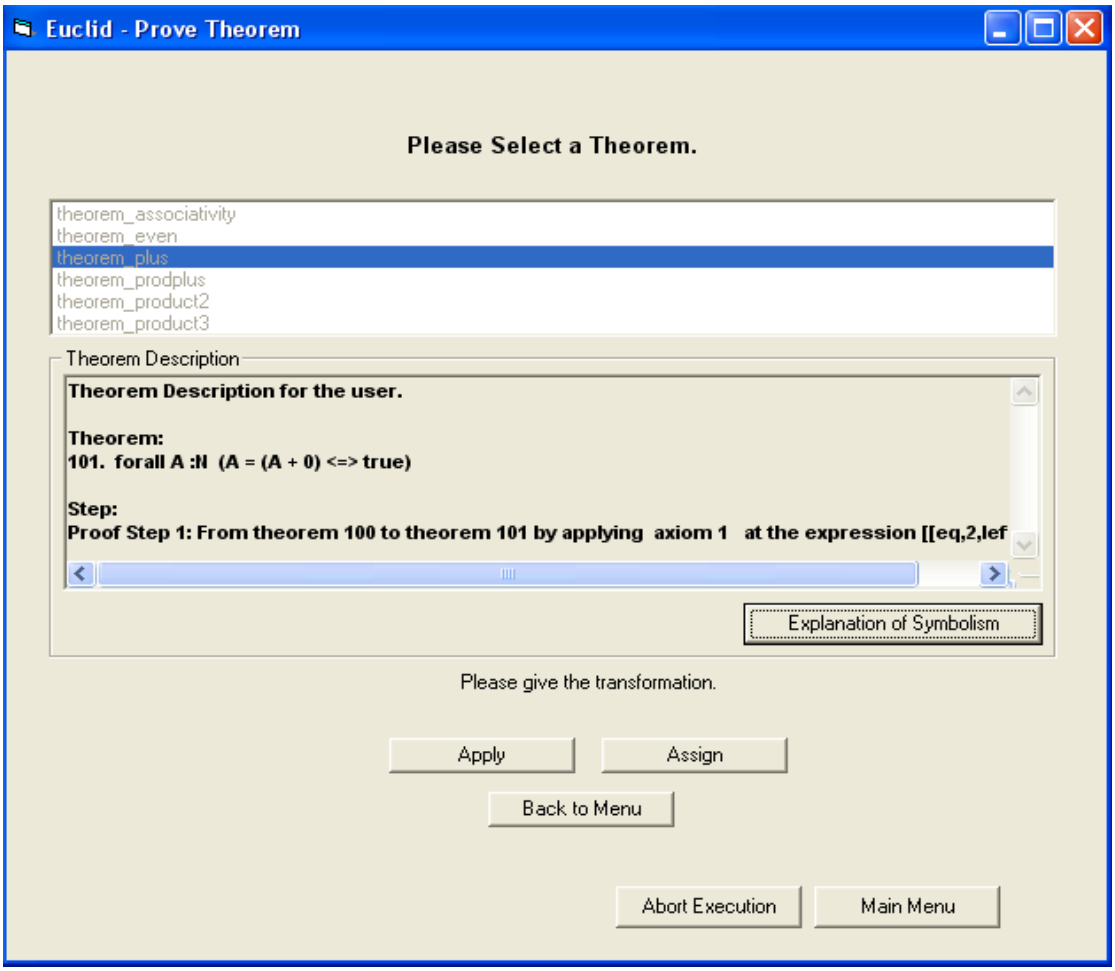

Euclid - Prove Theorem
Please Select a Theorem.
theorem_associativity
theorem_even
theorem_plus
theorem_prodplus
theorem_product2
theorem_product3
Theorem Description
Theorem Description for the user.
Theorem:
3. forall A,B :N ((A + B) = (B + A) <=> true)
Step:
No Proofstep
Explanation of Symbolism
Assign Variable Value
A=0
Execute
Abort Execution
Main Menu

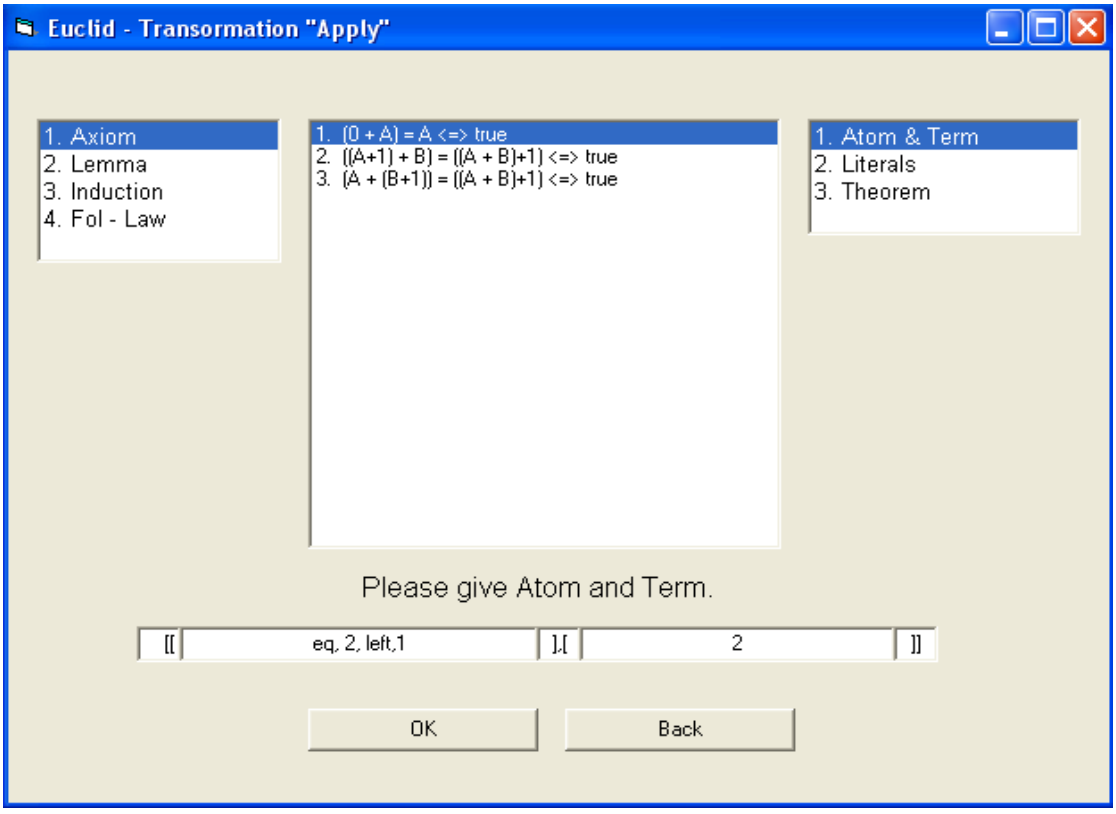

Euclid - Prove Theorem
Next command
Next Proofstep
Symbolic Evaluation
Cancel Last Step
Display All Theorems
Display All Proofsteps
Theorem Description
Theorem Description for the user.
Theorem:
100. forall A :N ((0 + A) = (A + 0) <=> true)
Step:
Proof Step 0: From theorem 3 to theorem 100 by applying Base Step (assign A = 0)
Explanation of Symbolism
Abort Execution
Main Menu

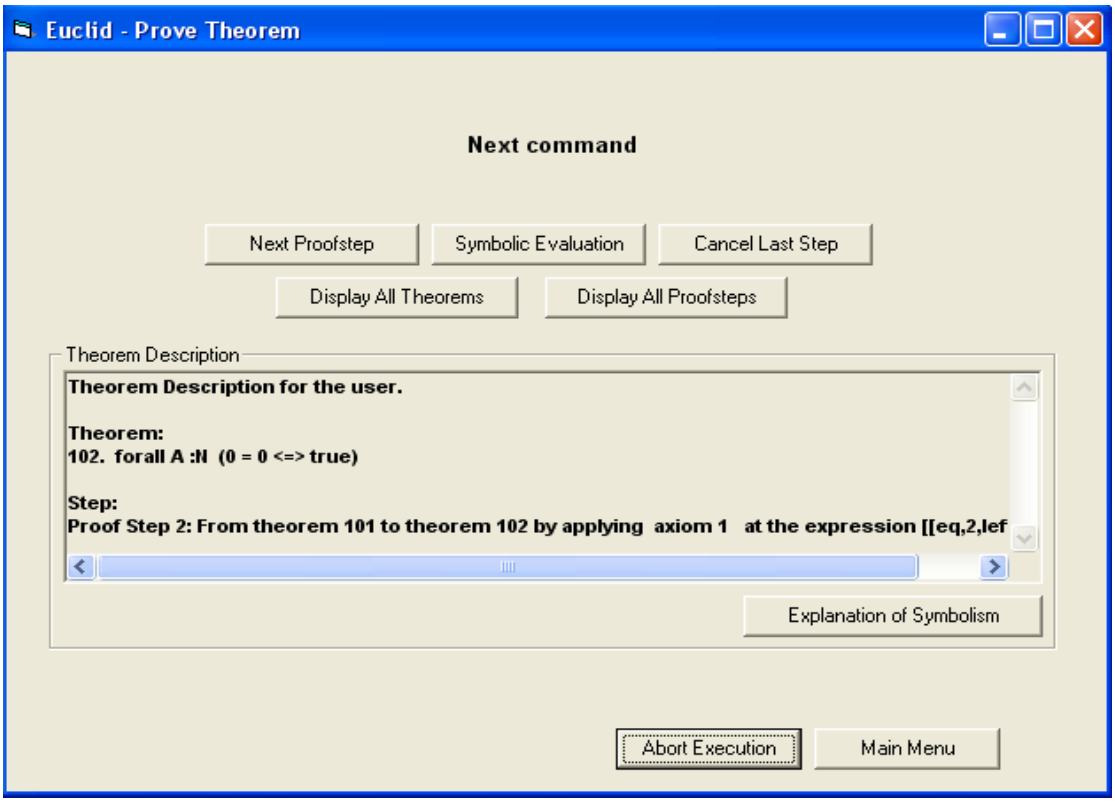

Euclid - Prove Theorem
Please Select a Theorem.
theorem_associativity
theorem_even
theorem_plus
theorem_prodplus
theorem_product2
theorem_product3
Theorem Description
Theorem Description for the System Administrator.
theorem(TheoremId , LeftForm , RightForm)
theorem(100 , [eq(plus(0,A),plus(A,0))] , [])
proofstep(ProofstepId , FromTheoremId , Transformation , NextTheoremId)
proofstep(1 , 3 , [assign,0,1] , 100)
Back
Please give the transformation.
Apply
Assign
Back to Menu
Abort Execution
Main Menu

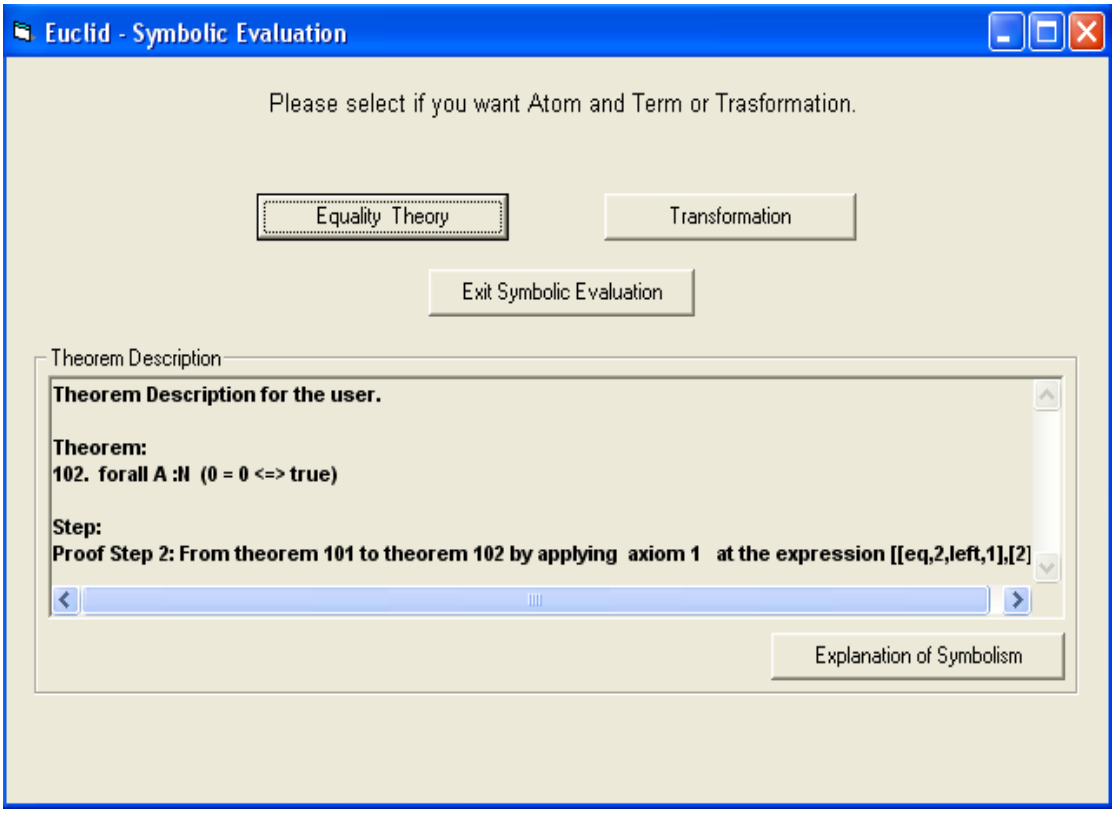

Euclid - Transormation "Apply"
1. Axiom
2. Lemma
3. Induction
4. Fol - Law
1. (0 + A) = A <=> true
2. ((A+1) + B) = ((A + B)+1) <=> true
3. (A + (B+1)) = ((A + B)+1) <=> true
1. Atom & Term
2. Literals
3. Theorem
Please give Atom and Term.
[[
eq, 2, left,1
],[
1
]]
OK
Back

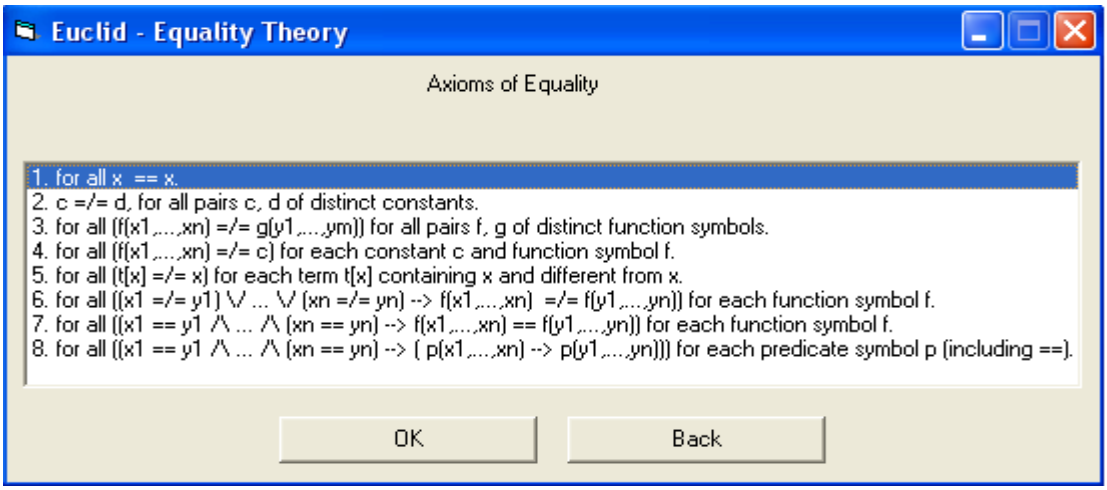

Euclid - Prove Theorem
Next command
Next Proofstep
Symbolic Evaluation
Cancel Last Step
Display All Theorems
Display All Proofsteps
Theorem Description
Theorem Description for the user.
Theorem:
101. forall A :N (A = (A + 0) <=> true)
Step:
Proof Step 1: From theorem 100 to theorem 101 by applying axiom 1 at the expression [[eq,2,lef
Explanation of Symbolism
Abort Execution
Main Menu

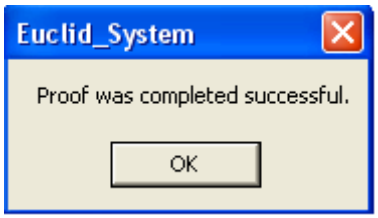

Euclid - Prove Theorem
Please Select a Theorem.
theorem_associativity
theorem_even
theorem_plus
theorem_prodplus
theorem_product2
theorem_product3
Theorem Description
Theorem Description for the user.
Theorem:
101. forall A :N (A = (A + 0) <=> true)
Step:
Proof Step 1: From theorem 100 to theorem 101 by applying axiom 1 at the expression [[eq,2,lef
Explanation of Symbolism
Please give the transformation.
Apply
Assign
Back to Menu
Abort Execution
Main Menu

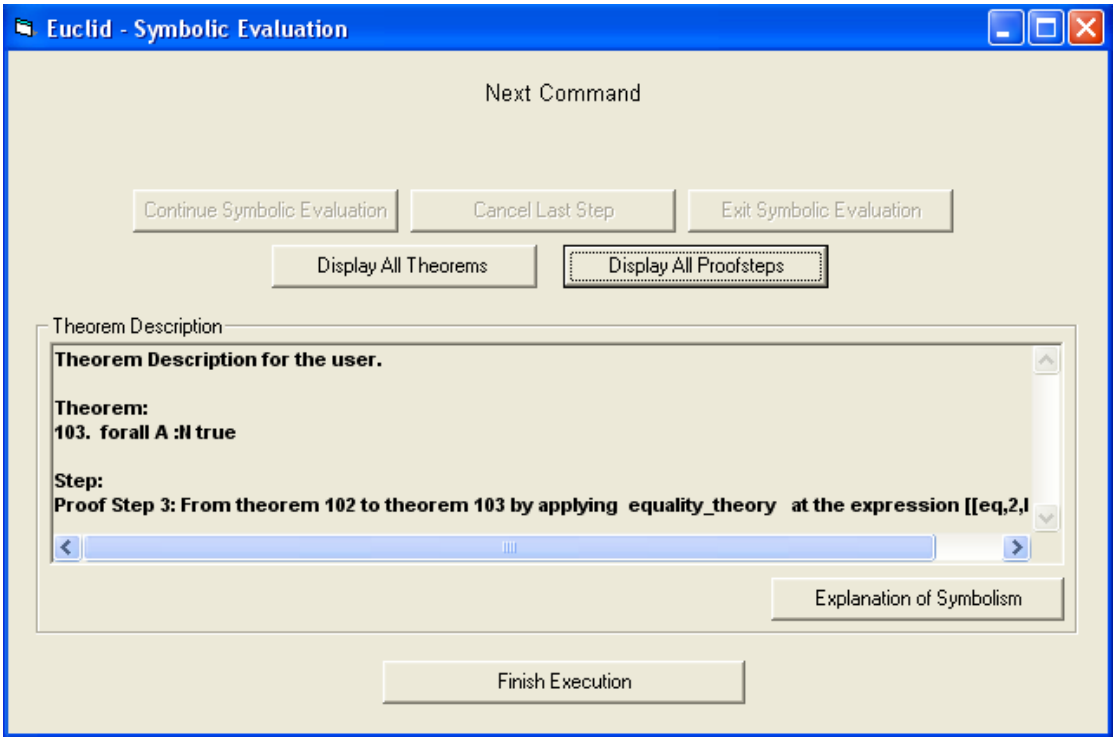

Euclid - Transormation "Apply"
1. Axiom
2. Lemma
3. Induction
4. Fol - Law
1. (0 + A) = A <=> true
2. ((A+1) + B) = ((A + B)+1) <=> true
3. (A + (B+1)) = ((A + B)+1) <=> true
1. Atom & Term
2. Literals
3. Theorem
Please give Atom and Term.
[[
eq, 2, left,1
],[
2
]]
OK
Back

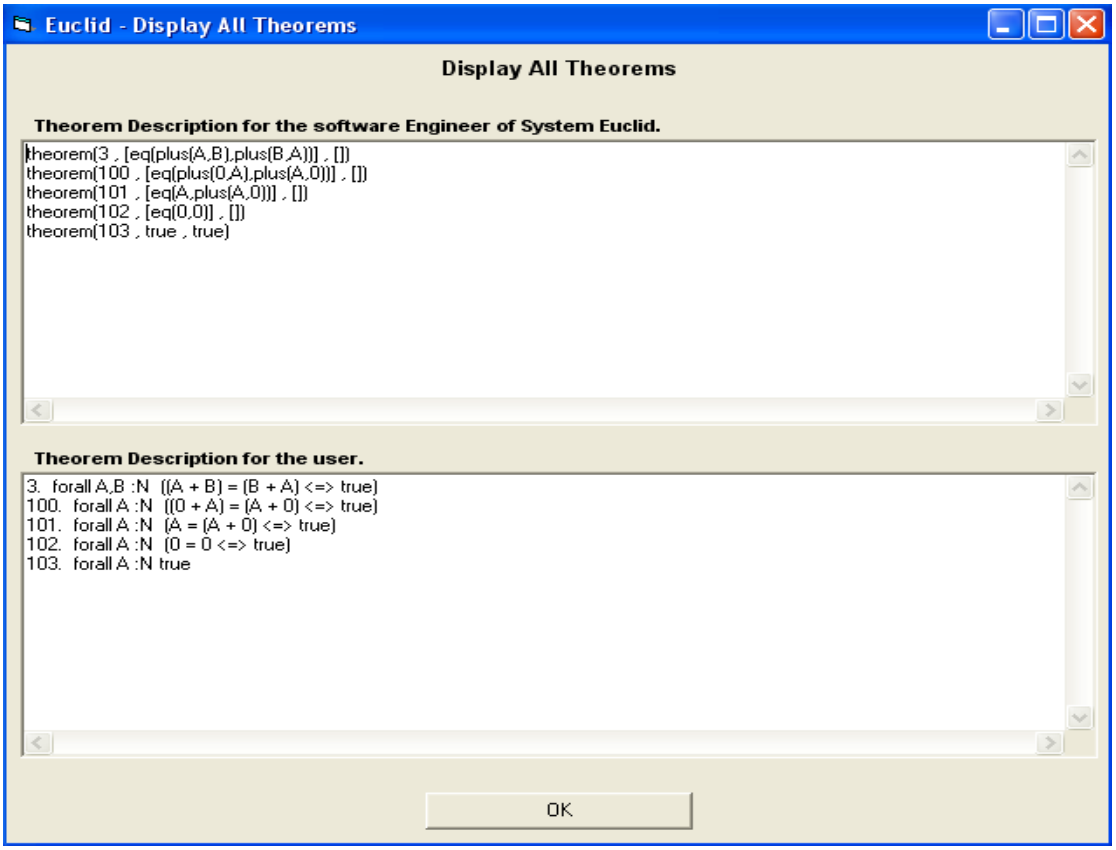

Euclid - Prove Theorem
Next command
Next Proofstep
Symbolic Evaluation
Cancel Last Step
Display All Theorems
Display All Proofsteps
Theorem Description
Theorem Description for the user.
Theorem:
102. forall A :N (0 = 0 <=> true)
Step:
Proof Step 2: From theorem 101 to theorem 102 by applying axiom 1 at the expression [[eq,2,lef
Explanation of Symbolism
Abort Execution
Main Menu

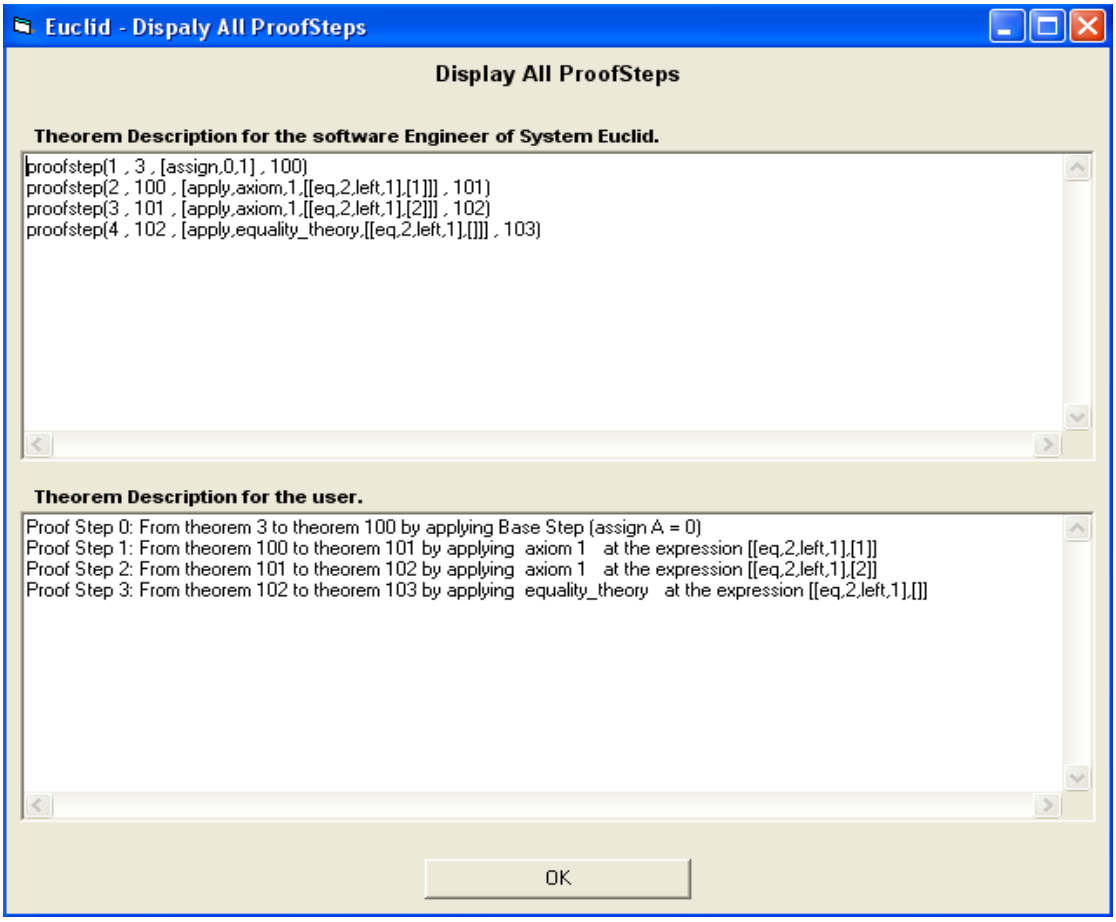

Euclid - Symbolic Evaluation
Please select if you want Atom and Term or Trasformation.
Equality Theory
Transformation
Exit Symbolic Evaluation
Theorem Description
Theorem Description for the user.
Theorem:
102. forall A :N (0 = 0 <=> true)
Step:
Proof Step 2: From theorem 101 to theorem 102 by applying axiom 1 at the expression [[eq,2,left,1],[2]
Explanation of Symbolism

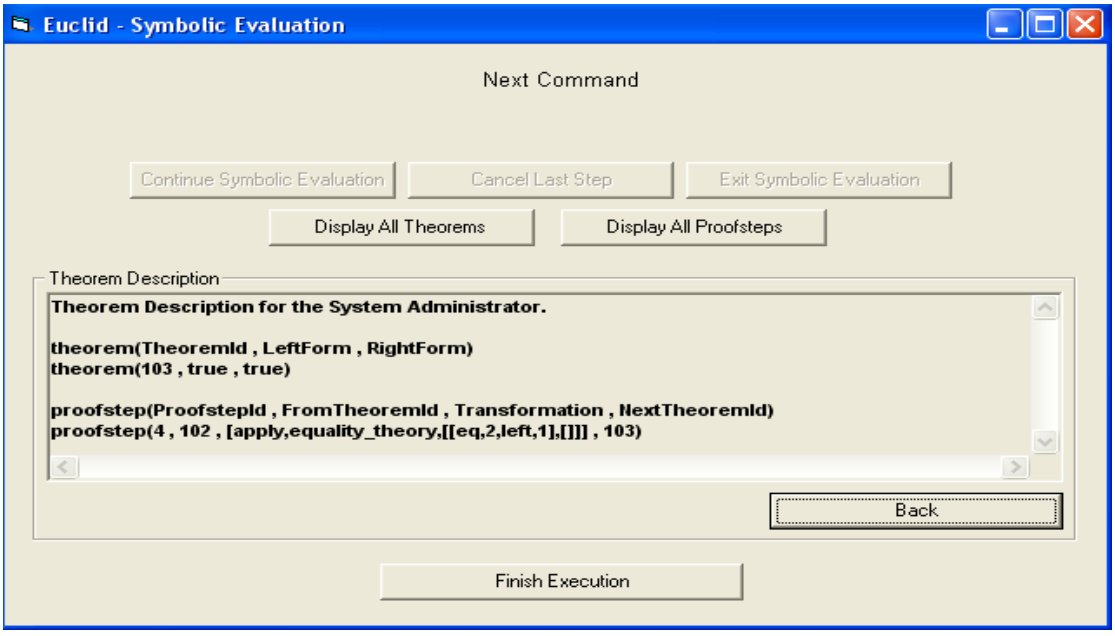

Euclid - Equality Theory
Axioms of Equality
1. for all x == x.
2. c =/= d, for all pairs c, d of distinct constants.
3. for all (f(x1,...,xn) =/= g(y1,...,ym)) for all pairs f, g of distinct function symbols.
4. for all (f(x1,...,xn) =/= c) for each constant c and function symbol f.
5. for all (t[x] =/= x) for each term t[x] containing x and different from x.
6. for all ((x1 =/= y1) \/ ... \/ (xn =/= yn) --> f(x1,...,xn) =/= f(y1,...,yn)) for each function symbol f.
7. for all ((x1 == y1 /\ ... /\ (xn == yn) --> f(x1,...,xn) == f(y1,...,yn)) for each function symbol f.
8. for all ((x1 == y1 /\ ... /\ (xn == yn) --> (p(x1,...,xn) --> p(y1,...,yn))) for each predicate symbol p (including ==).
OK
Back

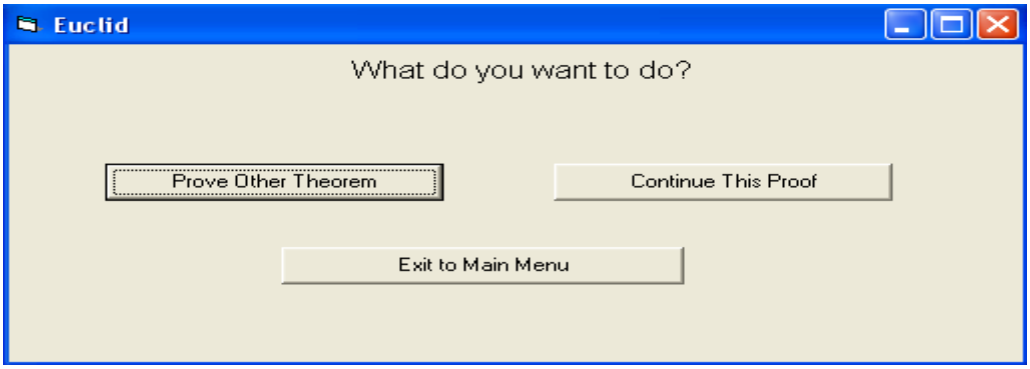

Euclid_System
Proof was completed successful.
OK

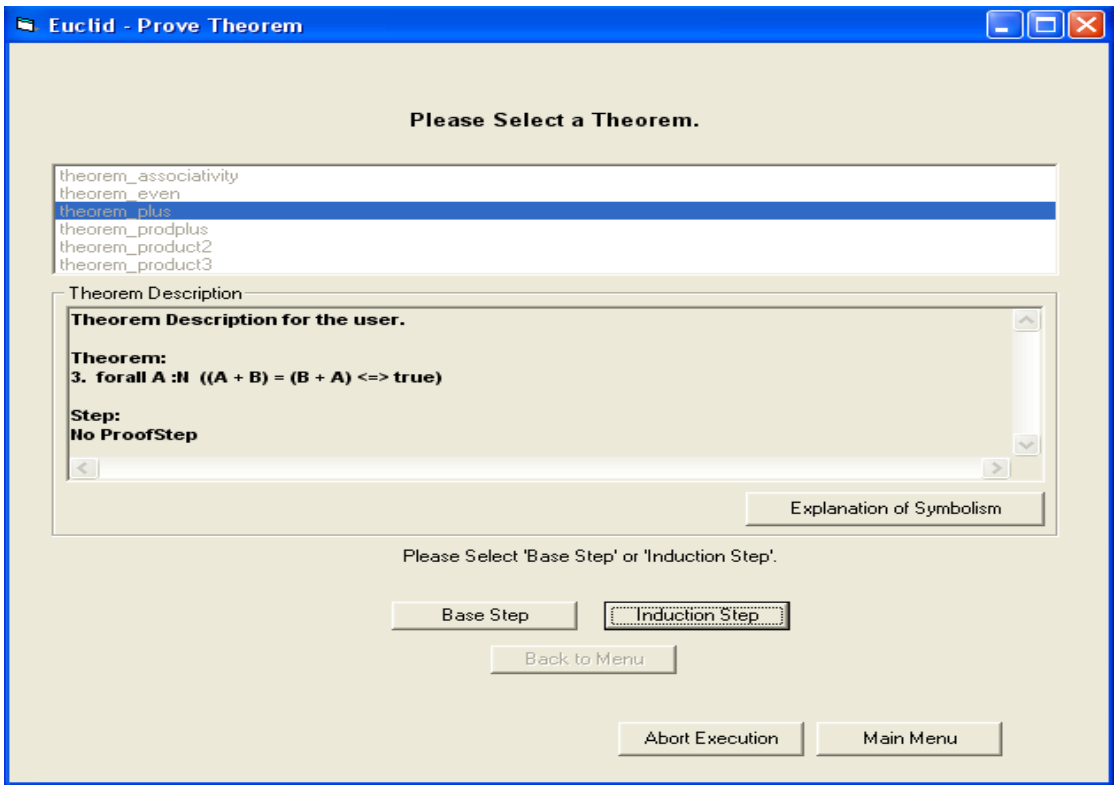

Euclid - Symbolic Evaluation
Next Command
Continue Symbolic Evaluation
Cancel Last Step
Exit Symbolic Evaluation
Display All Theorems
Display All Proofsteps
Theorem Description
Theorem Description for the user.
Theorem:
103. forall A :N true
Step:
Proof Step 3: From theorem 102 to theorem 103 by applying equality_theory at the expression [[eq,2,I
Explanation of Symbolism
Finish Execution

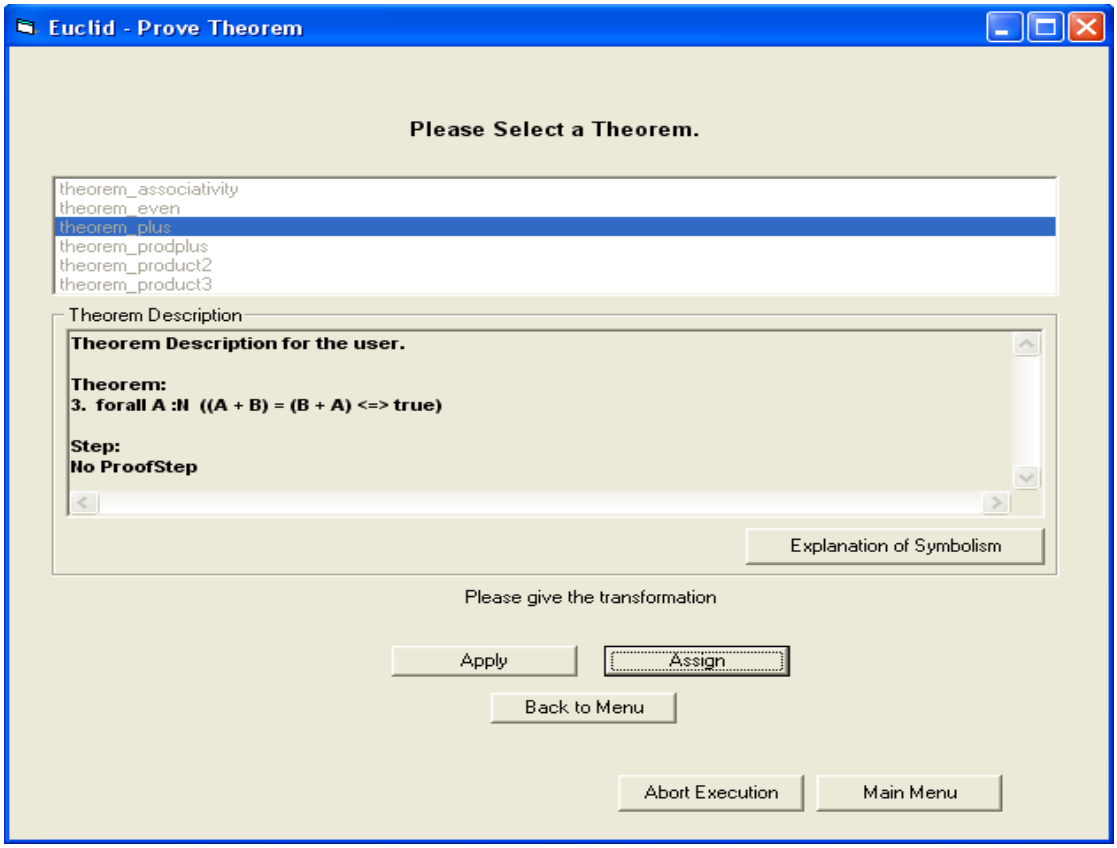

Euclid - Display All Theorems
Display All Theorems
Theorem Description for the software Engineer of System Euclid.
theorem(3 , [eq(plus(A,B),plus(B,A))] , [])
theorem(100 , [eq(plus(0,A),plus(A,0))] , [])
theorem(101 , [eq(A,plus(A,0))] , [])
theorem(102 , [eq(0,0)] , [])
theorem(103 , true , true)
Theorem Description for the user.
3. forall A,B :N ((A + B) = (B + A) <=> true)
100. forall A :N ((0 + A) = (A + 0) <=> true)
101. forall A :N (A = (A + 0) <=> true)
102. forall A :N (0 = 0 <=> true)
103. forall A :N true
OK

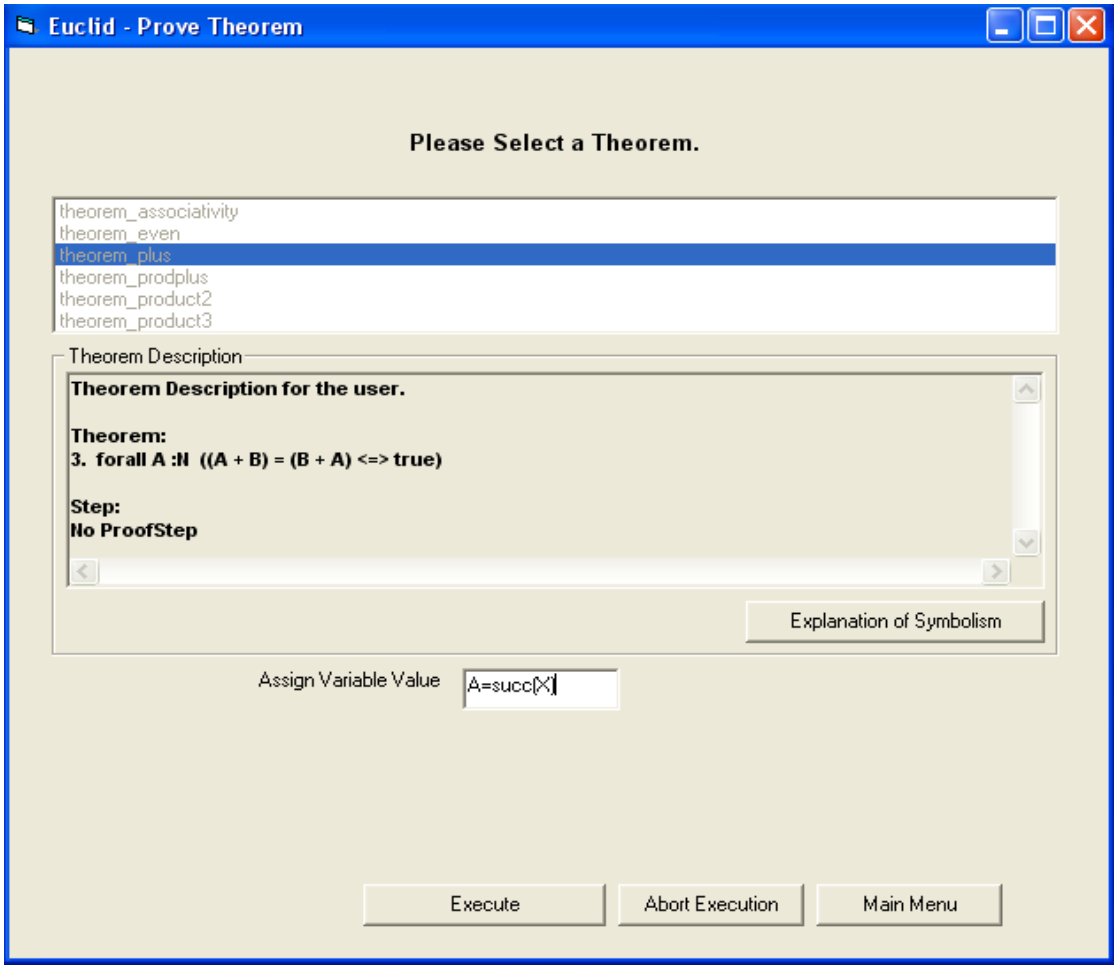

Euclid - Dispaly All ProofSteps
Display All ProofSteps
Theorem Description for the software Engineer of System Euclid.
proofstep(1 , 3 , [assign,0,1] , 100)
proofstep(2 , 100 , [apply,axiom,1,[[eq,2,left,1],[1]]] , 101)
proofstep(3 , 101 , [apply,axiom,1,[[eq,2,left,1],[2]]] , 102)
proofstep(4 , 102 , [apply,equality_theory,[[eq,2,left,1],[]]] , 103)
Theorem Description for the user.
Proof Step 0: From theorem 3 to theorem 100 by applying Base Step (assign A = 0)
Proof Step 1: From theorem 100 to theorem 101 by applying axiom 1 at the expression [[eq,2,left,1],[1]]
Proof Step 2: From theorem 101 to theorem 102 by applying axiom 1 at the expression [[eq,2,left,1],[2]]
Proof Step 3: From theorem 102 to theorem 103 by applying equality_theory at the expression [[eq,2,left,1],[]]
OK

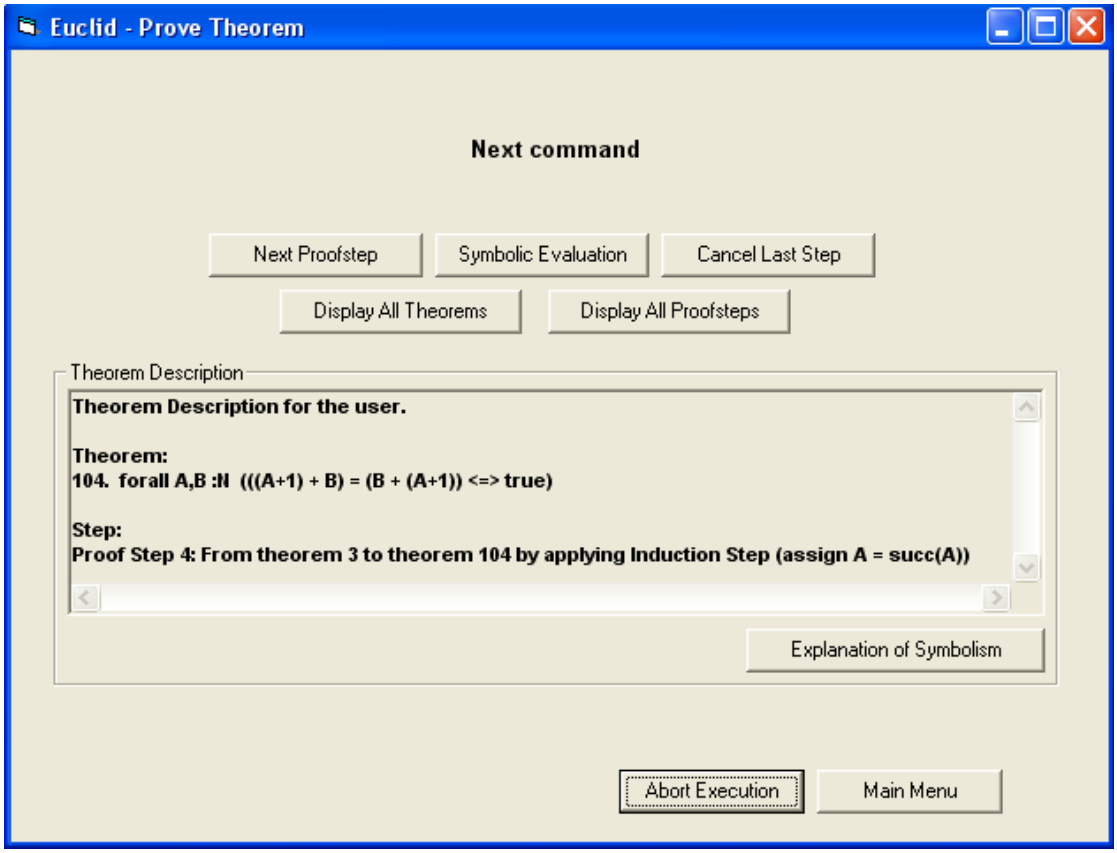

Euclid - Symbolic Evaluation
Next Command
Continue Symbolic Evaluation
Cancel Last Step
Exit Symbolic Evaluation
Display All Theorems
Display All Proofsteps
Theorem Description
Theorem Description for the System Administrator.
theorem(TheoremId , LeftForm , RightForm)
theorem(103 , true , true)
proofstep(ProofstepId , FromTheoremId , Transformation , NextTheoremId)
proofstep(4 , 102 , [apply,equality_theory,[[eq,2,left,1],[]]] , 103)
Back
Finish Execution

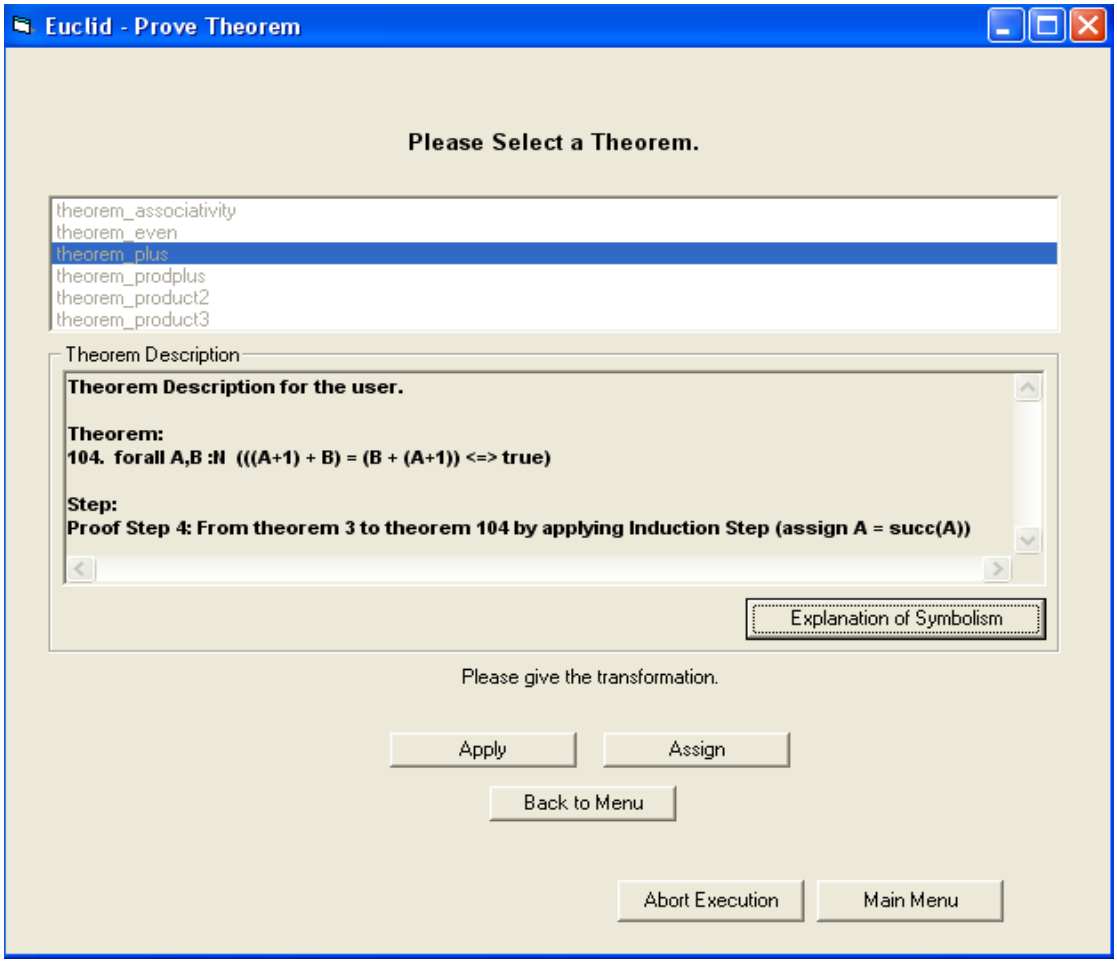

Euclid
What do you want to do?
Prove Other Theorem
Continue This Proof
Exit to Main Menu

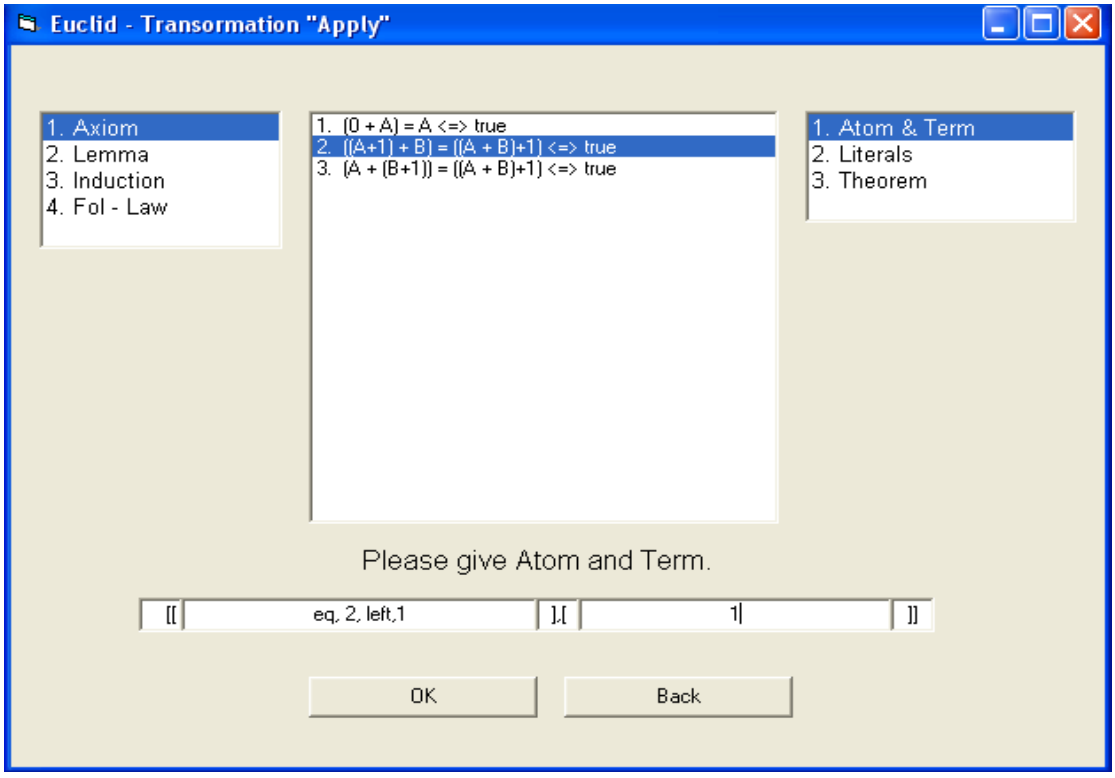

Euclid - Prove Theorem
Please Select a Theorem.
theorem_associativity
theorem_even
theorem_plus
theorem_prodplus
theorem_product2
theorem_product3
Theorem Description
Theorem Description for the user.
Theorem:
3. forall A :N ((A + B) = (B + A) <=> true)
Step:
No ProofStep
Explanation of Symbolism
Please Select 'Base Step' or 'Induction Step'.
Base Step
Induction Step
Back to Menu
Abort Execution
Main Menu

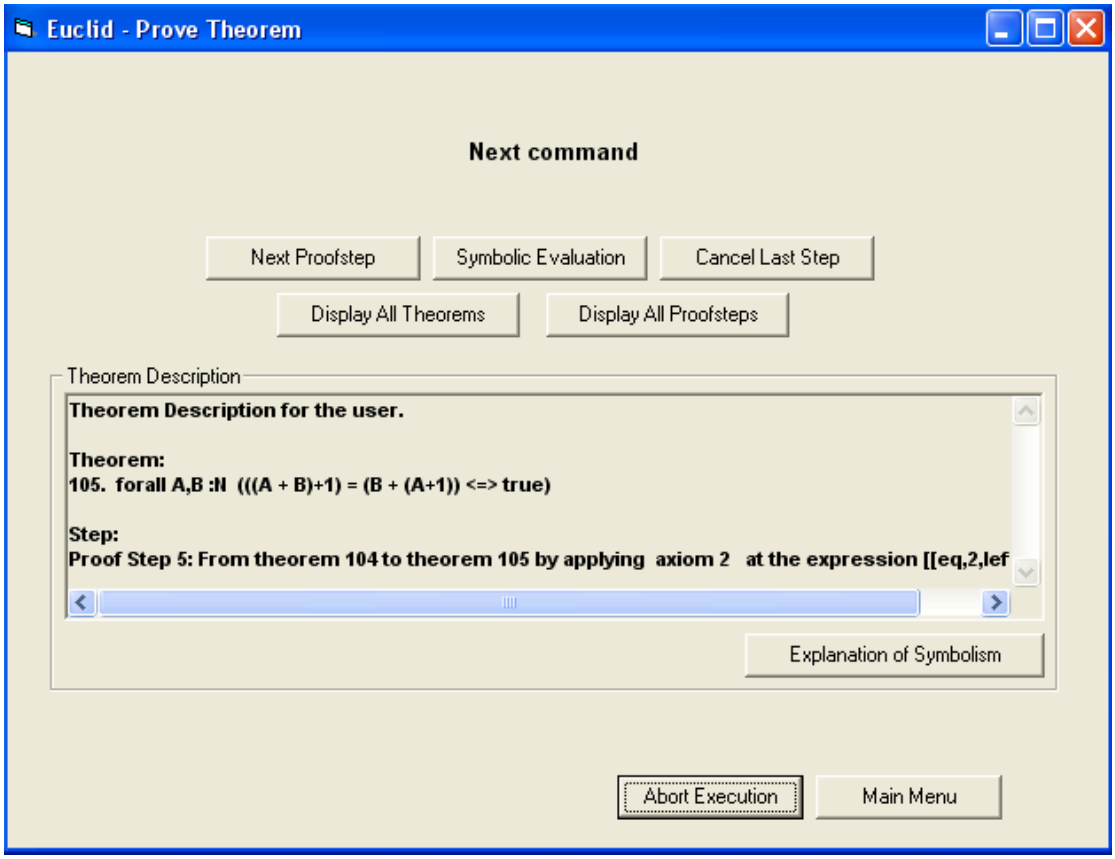

Euclid - Prove Theorem
Please Select a Theorem.
theorem_associativity
theorem_even
theorem_plus
theorem_prodplus
theorem_product2
theorem_product3
Theorem Description
Theorem Description for the user.
Theorem:
3. forall A :N ((A + B) = (B + A) <=> true)
Step:
No ProofStep
Explanation of Symbolism
Please give the transformation
Apply
Assign
Back to Menu
Abort Execution
Main Menu

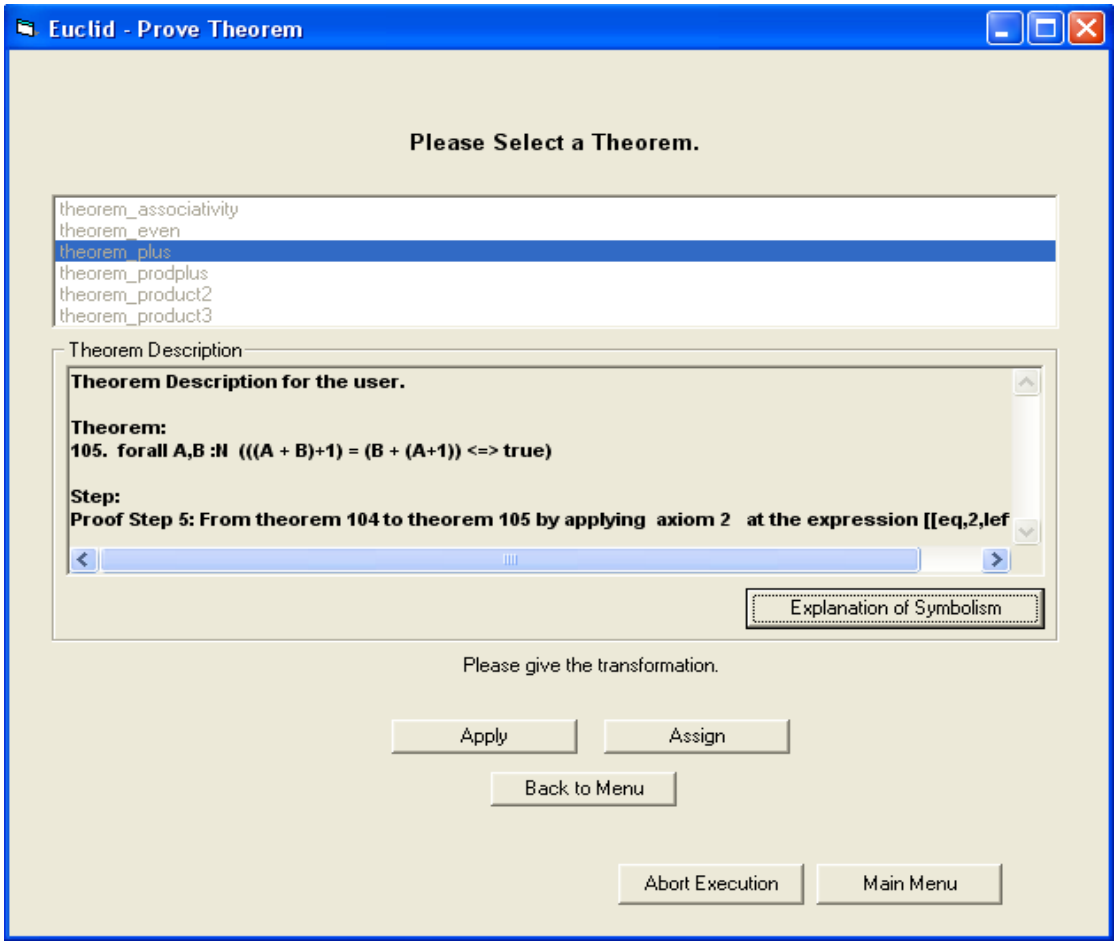

Euclid - Prove Theorem
Please Select a Theorem.
theorem_associativity
theorem_even
theorem_plus
theorem_prodplus
theorem_product2
theorem_product3
Theorem Description
Theorem Description for the user.
Theorem:
3. forall A :N ((A + B) = (B + A) <=> true)
Step:
No ProofStep
Explanation of Symbolism
Assign Variable Value
A=succ(X)
Execute
Abort Execution
Main Menu

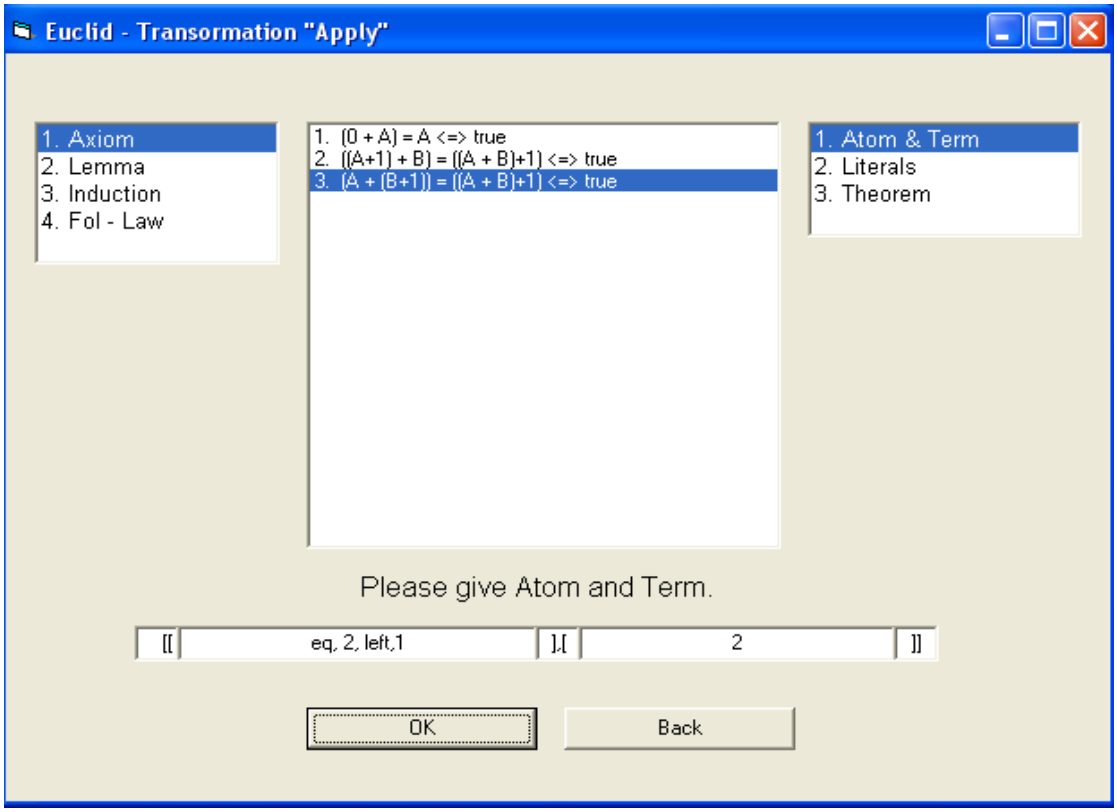

Euclid - Prove Theorem
Next command
Next Proofstep
Symbolic Evaluation
Cancel Last Step
Display All Theorems
Display All Proofsteps
Theorem Description
Theorem Description for the user.
Theorem:
104. forall A,B :N (((A+1) + B) = (B + (A+1)) <=> true)
Step:
Proof Step 4: From theorem 3 to theorem 104 by applying Induction Step (assign A = succ(A))
Explanation of Symbolism
Abort Execution
Main Menu

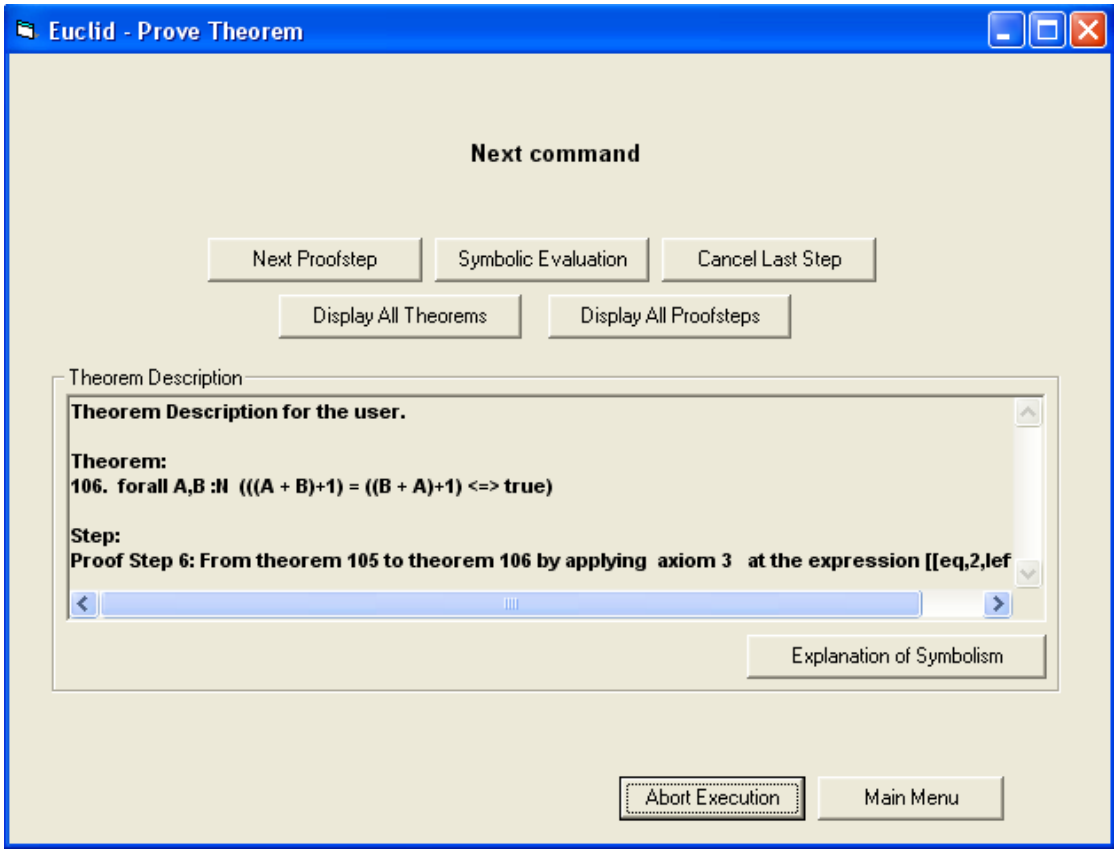

Euclid - Prove Theorem
Please Select a Theorem.
theorem_associativity
theorem_even
theorem_plus
theorem_prodplus
theorem_product2
theorem_product3
Theorem Description
Theorem Description for the user.
Theorem:
104. forall A,B :N (((A+1) + B) = (B + (A+1)) <=> true)
Step:
Proof Step 4: From theorem 3 to theorem 104 by applying Induction Step (assign A = succ(A))
Explanation of Symbolism
Please give the transformation.
Apply
Assign
Back to Menu
Abort Execution
Main Menu

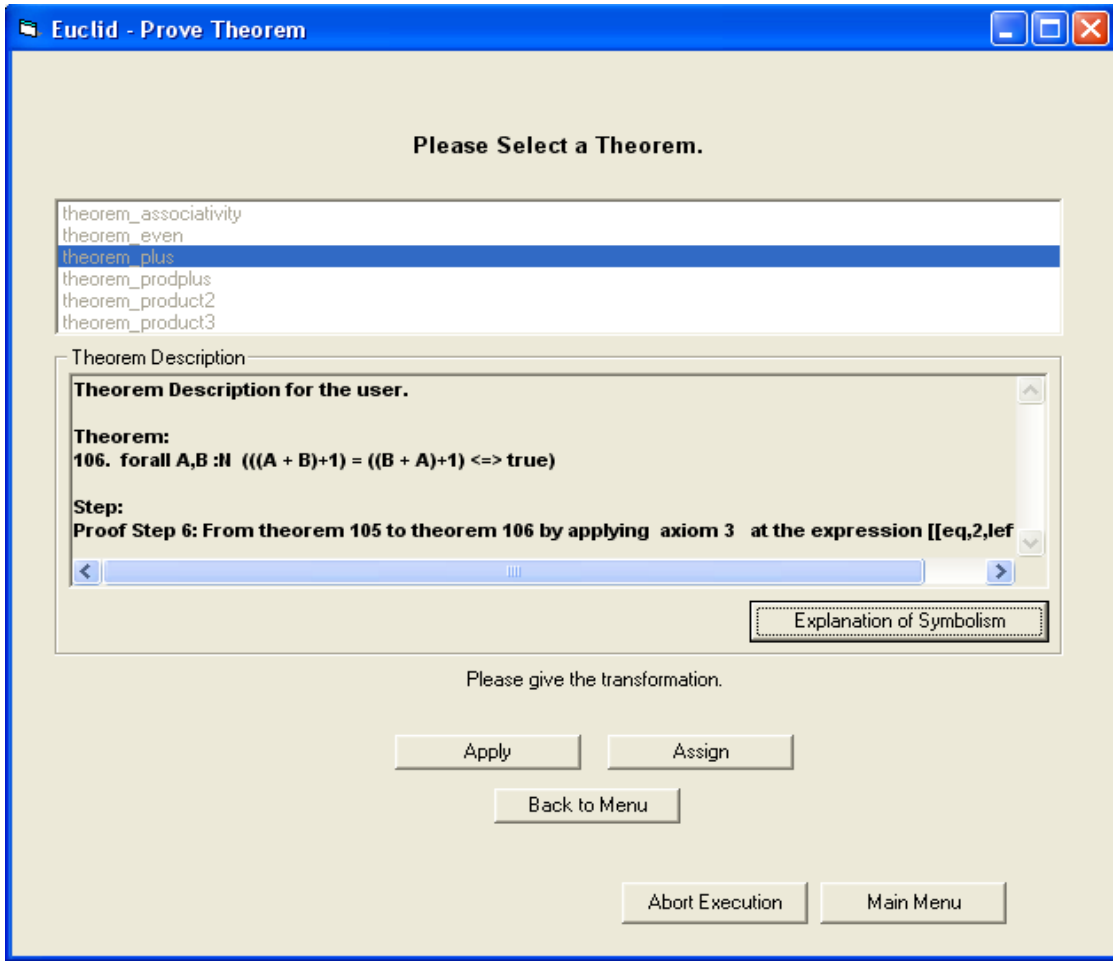

Euclid - Transormation "Apply"
1. Axiom
2. Lemma
3. Induction
4. Fol - Law
1. (0 + A) = A <=> true
2. ((A+1) + B) = ((A + B)+1) <=> true
3. (A + (B+1)) = ((A + B)+1) <=> true
1. Atom & Term
2. Literals
3. Theorem
Please give Atom and Term.
[[
eq, 2, left,1
],[
1
]]
OK
Back

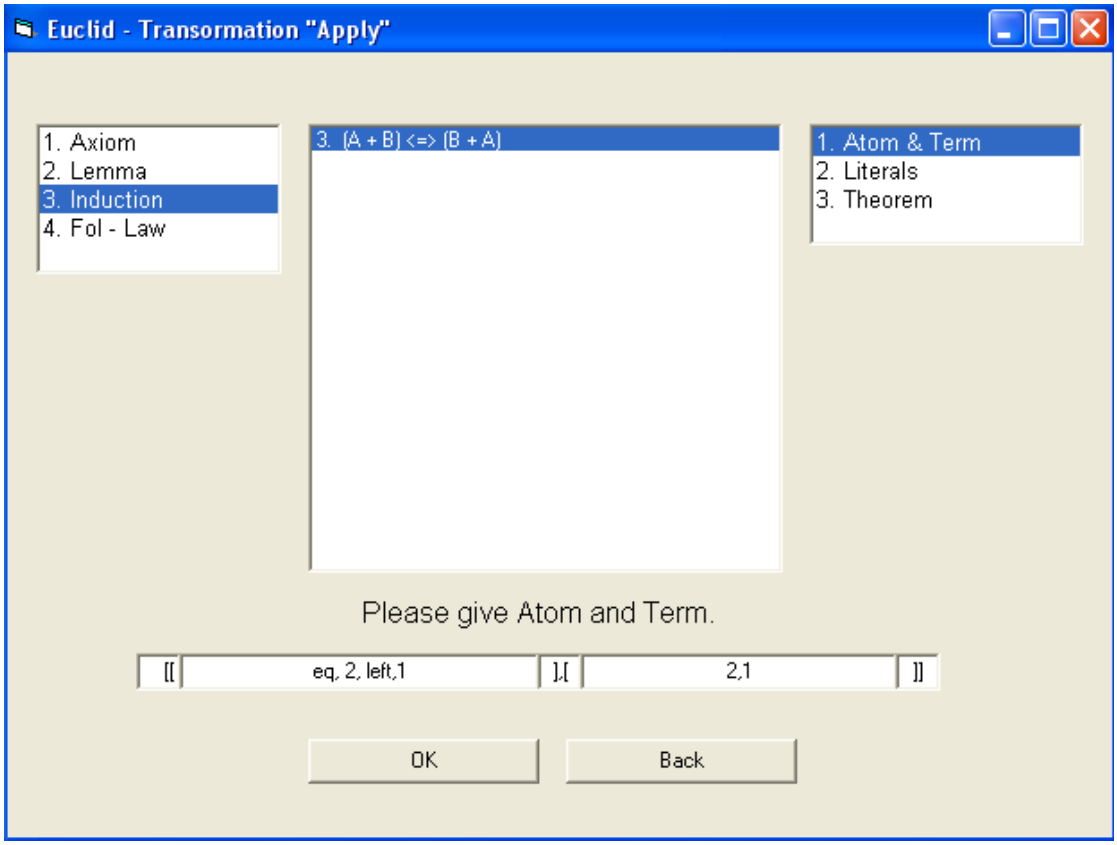

Euclid - Prove Theorem
Next command
Next Proofstep
Symbolic Evaluation
Cancel Last Step
Display All Theorems
Display All Proofsteps
Theorem Description
Theorem Description for the user.
Theorem:
105. forall A,B :N (((A + B)+1) = (B + (A+1)) <=> true)
Step:
Proof Step 5: From theorem 104 to theorem 105 by applying axiom 2 at the expression [[eq,2,lef
Explanation of Symbolism
Abort Execution
Main Menu

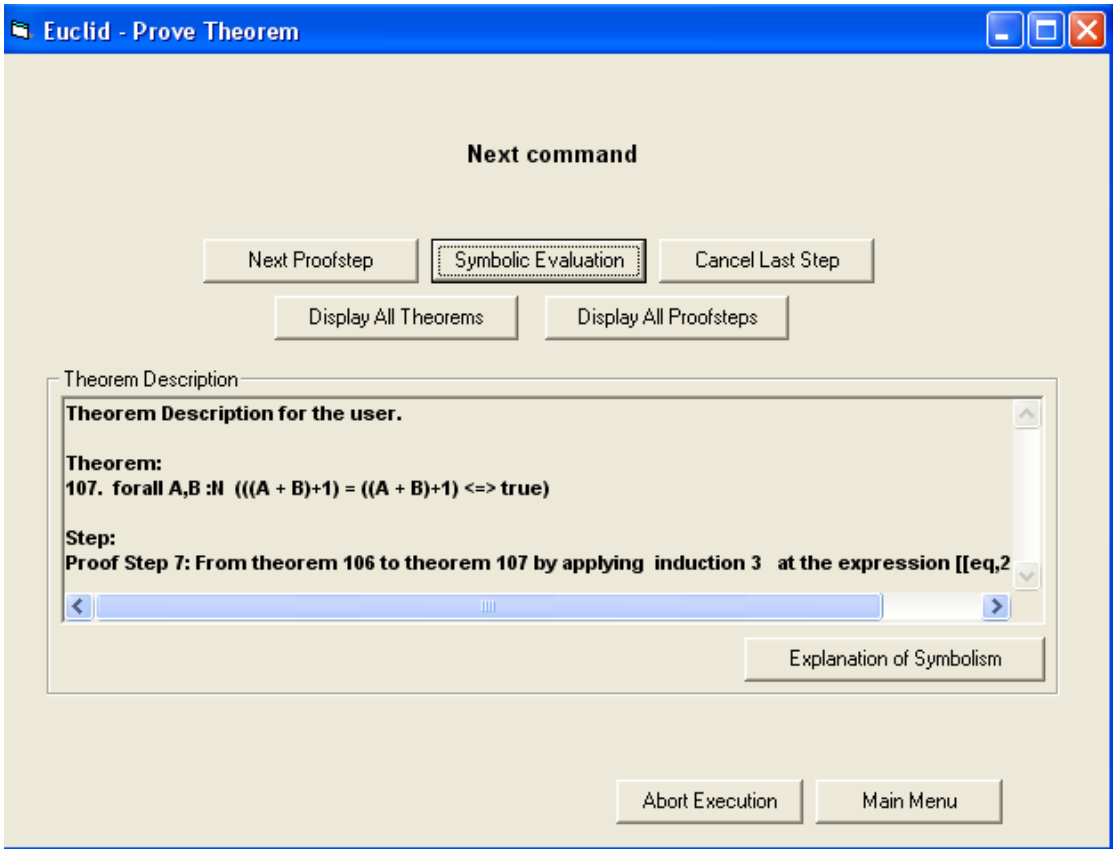

Euclid - Prove Theorem
Please Select a Theorem.
theorem_associativity
theorem_even
theorem_plus
theorem_prodplus
theorem_product2
theorem_product3
Theorem Description
Theorem Description for the user.
Theorem:
105. forall A,B :N (((A + B)+1) = (B + (A+1)) <=> true)
Step:
Proof Step 5: From theorem 104 to theorem 105 by applying axiom 2 at the expression [[eq,2,lef
Explanation of Symbolism
Please give the transformation.
Apply
Assign
Back to Menu
Abort Execution
Main Menu

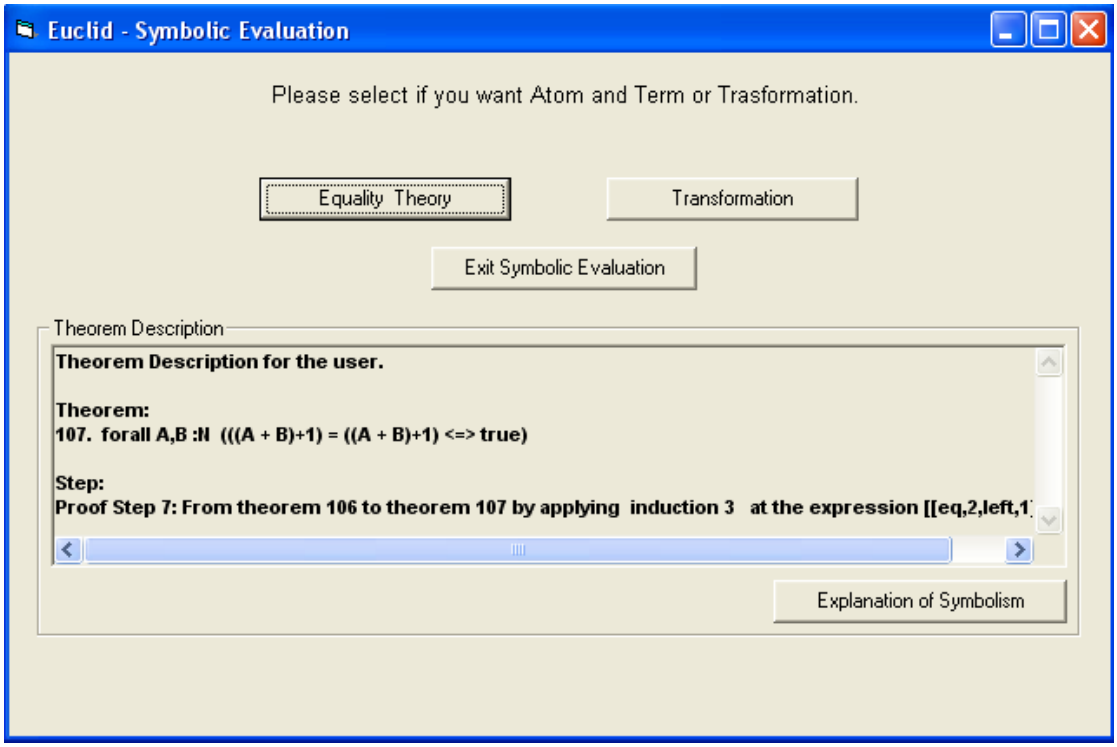

Euclid - Transormation "Apply"
1. Axiom
2. Lemma
3. Induction
4. Fol - Law
1. (0 + A) = A <=> true
2. ((A+1) + B) = ((A + B)+1) <=> true
3. (A + (B+1)) = ((A + B)+1) <=> true
1. Atom & Term
2. Literals
3. Theorem
Please give Atom and Term.
[[
eq, 2, left,1
],[
2
]]
OK
Back

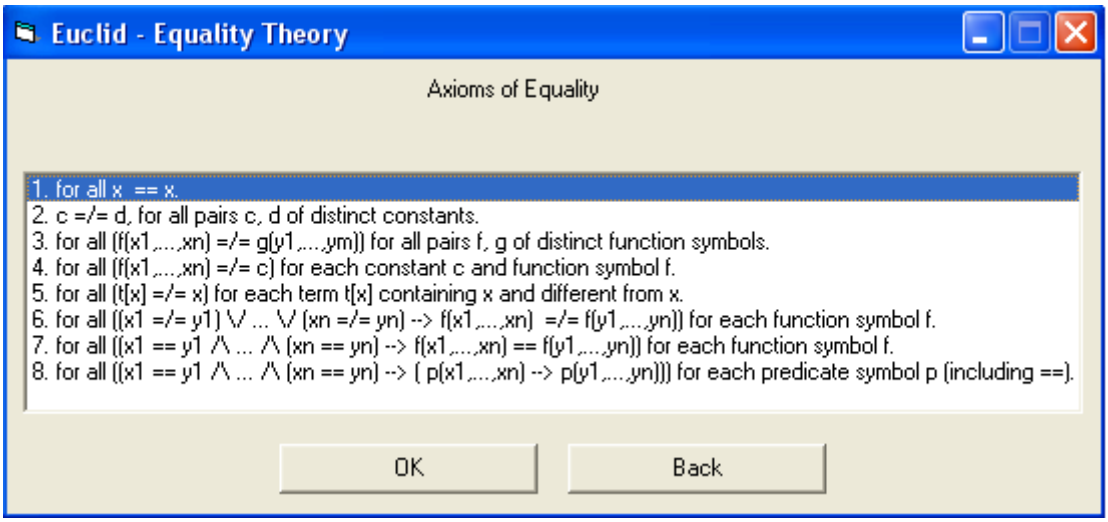

Euclid - Prove Theorem
Next command
Next Proofstep
Symbolic Evaluation
Cancel Last Step
Display All Theorems
Display All Proofsteps
Theorem Description
Theorem Description for the user.
Theorem:
106. forall A,B :N (((A + B)+1) = ((B + A)+1) <=> true)
Step:
Proof Step 6: From theorem 105 to theorem 106 by applying axiom 3 at the expression [[eq,2,lef
Explanation of Symbolism
Abort Execution
Main Menu

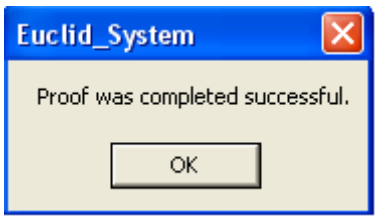

Euclid - Prove Theorem
Please Select a Theorem.
theorem_associativity
theorem_even
theorem_plus
theorem_prodplus
theorem_product2
theorem_product3
Theorem Description
Theorem Description for the user.
Theorem:
106. forall A,B :N (((A + B)+1) = ((B + A)+1) <=> true)
Step:
Proof Step 6: From theorem 105 to theorem 106 by applying axiom 3 at the expression [[eq,2,lef
Explanation of Symbolism
Please give the transformation.
Apply
Assign
Back to Menu
Abort Execution
Main Menu

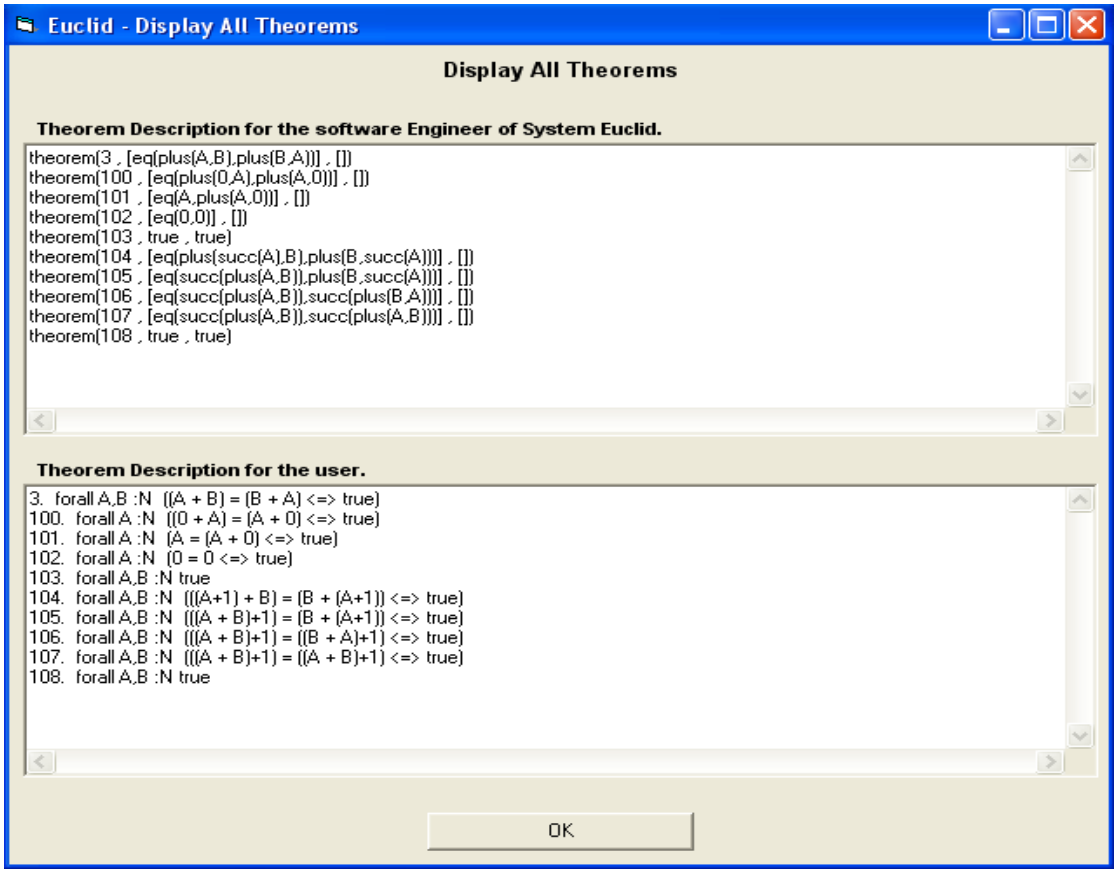

Euclid - Transormation "Apply"
1. Axiom
2. Lemma
3. Induction
4. Fol - Law
3. (A + B) <=> (B + A)
1. Atom & Term
2. Literals
3. Theorem
Please give Atom and Term.
[[
eq, 2, left,1
],[
2,1
]]
OK
Back

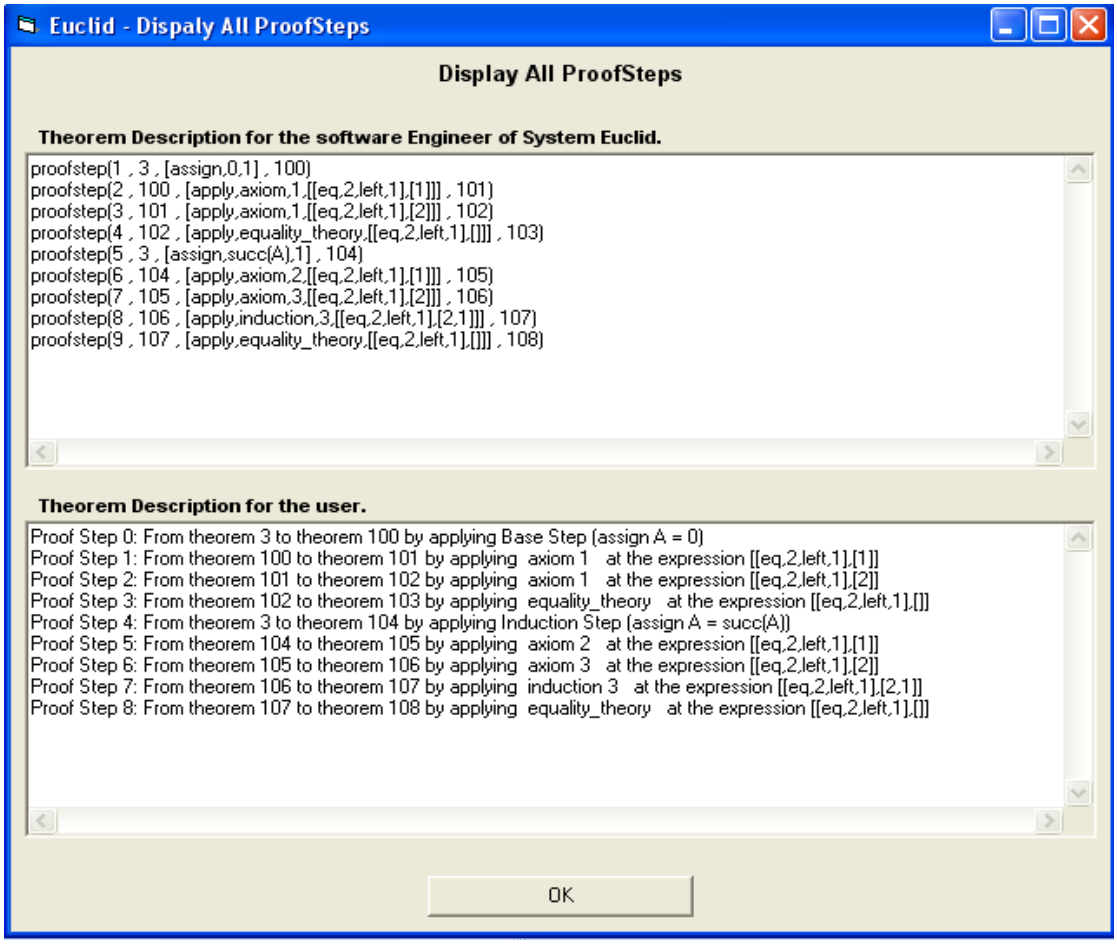

Euclid - Prove Theorem
Next command
Next Proofstep
Symbolic Evaluation
Cancel Last Step
Display All Theorems
Display All Proofsteps
Theorem Description
Theorem Description for the user.
Theorem:
107. forall A,B :N (((A + B)+1) = ((A + B)+1) <=> true)
Step:
Proof Step 7: From theorem 106 to theorem 107 by applying induction 3 at the expression [[eq,2
Explanation of Symbolism
Abort Execution
Main Menu


Euclid - Symbolic Evaluation
Please select if you want Atom and Term or Trasformation.
Equality Theory
Transformation
Exit Symbolic Evaluation
Theorem Description
Theorem Description for the user.
Theorem:
107. forall A,B :N (((A + B)+1) = ((A + B)+1) <=> true)
Step:
Proof Step 7: From theorem 106 to theorem 107 by applying induction 3 at the expression [[eq,2,left,1
Explanation of Symbolism


Euclid - Equality Theory
Axioms of Equality
1. for all x == x.
2. c =/= d, for all pairs c, d of distinct constants.
3. for all (f(x1,...,xn) =/= g(y1,...,ym)) for all pairs f, g of distinct function symbols.
4. for all (f(x1,...,xn) =/= c) for each constant c and function symbol f.
5. for all (t[x] =/= x) for each term t[x] containing x and different from x.
6. for all ((x1 =/= y1) \/ ... \/ (xn =/= yn) --> f(x1,...,xn) =/= f(y1,...,yn)) for each function symbol f.
7. for all ((x1 == y1 /\ ... /\ (xn == yn) --> f(x1,...,xn) == f(y1,...,yn)) for each function symbol f.
8. for all ((x1 == y1 /\ ... /\ (xn == yn) --> (p(x1,...,xn) --> p(y1,...,yn))) for each predicate symbol p (including ==).
OK
Back


Euclid_System
Proof was completed successful.
OK


Euclid - Display All Theorems
Display All Theorems
Theorem Description for the software Engineer of System Euclid.
theorem(3 , [eq(plus(A,B),plus(B,A))] , [])
theorem(100 , [eq(plus(0,A),plus(A,0))] , [])
theorem(101 , [eq(A,plus(A,0))] , [])
theorem(102 , [eq(0,0)] , [])
theorem(103 , true , true)
theorem(104 , [eq(plus(succ(A),B),plus(B,succ(A)))] , [])
theorem(105 , [eq(succ(plus(A,B)),plus(B,succ(A)))] , [])
theorem(106 , [eq(succ(plus(A,B)),succ(plus(B,A)))] , [])
theorem(107 , [eq(succ(plus(A,B)),succ(plus(A,B)))] , [])
theorem(108 , true , true)
Theorem Description for the user.
3. forall A,B :N ((A + B) = (B + A) <=> true)
100. forall A :N ((0 + A) = (A + 0) <=> true)
101. forall A :N (A = (A + 0) <=> true)
102. forall A :N (0 = 0 <=> true)
103. forall A,B :N true
104. forall A,B :N (((A+1) + B) = (B + (A+1)) <=> true)
105. forall A,B :N (((A + B)+1) = (B + (A+1)) <=> true)
106. forall A,B :N (((A + B)+1) = ((B + A)+1) <=> true)
107. forall A,B :N (((A + B)+1) = ((A + B)+1) <=> true)
108. forall A,B :N true
OK


Euclid - Dispaly All ProofSteps
Display All ProofSteps
Theorem Description for the software Engineer of System Euclid.
proofstep(1 , 3 , [assign,0,1] , 100)
proofstep(2 , 100 , [apply,axiom,1,[[eq,2,left,1],[1]]] , 101)
proofstep(3 , 101 , [apply,axiom,1,[[eq,2,left,1],[2]]] , 102)
proofstep(4 , 102 , [apply,equality_theory,[[eq,2,left,1],[]]] , 103)
proofstep(5 , 3 , [assign,succ(A),1] , 104)
proofstep(6 , 104 , [apply,axiom,2,[[eq,2,left,1],[1]]] , 105)
proofstep(7 , 105 , [apply,axiom,3,[[eq,2,left,1],[2]]] , 106)
proofstep(8 , 106 , [apply,induction,3,[[eq,2,left,1],[2,1]]] , 107)
proofstep(9 , 107 , [apply,equality_theory,[[eq,2,left,1],[]]] , 108)
Theorem Description for the user.
Proof Step 0: From theorem 3 to theorem 100 by applying Base Step (assign A = 0)
Proof Step 1: From theorem 100 to theorem 101 by applying axiom 1 at the expression [[eq,2,left,1],[1]]
Proof Step 2: From theorem 101 to theorem 102 by applying axiom 1 at the expression [[eq,2,left,1],[2]]
Proof Step 3: From theorem 102 to theorem 103 by applying equality_theory at the expression [[eq,2,left,1],[]]
Proof Step 4: From theorem 3 to theorem 104 by applying Induction Step (assign A = succ(A))
Proof Step 5: From theorem 104 to theorem 105 by applying axiom 2 at the expression [[eq,2,left,1],[1]]
Proof Step 6: From theorem 105 to theorem 106 by applying axiom 3 at the expression [[eq,2,left,1],[2]]
Proof Step 7: From theorem 106 to theorem 107 by applying induction 3 at the expression [[eq,2,left,1],[2,1]]
Proof Step 8: From theorem 107 to theorem 108 by applying equality_theory at the expression [[eq,2,left,1],[]]
OK

Β Επιπλέον Παράδειγμα Απόδειξης με Μαθηματική Επαγωγή

Β.1 Παράδειγμα 1: Προσεταιριστική Ιδιότητα Φυσικών Αριθμών

Αποδείξτε ότι για όλα τα x,y,z∈N ισχύει η εξής πρόταση *P*:

$$P\text{: } \mathbf{x * (y * z) = (x * y) * z}$$

Απόδειξη:
Αρχίζουμε να εφαρμόζουμε την μέθοδο της μαθηματικής επαγωγής για x =0.

I.Βήμα βάσης

$$\begin{aligned} 0 * (y * z) &= (0 * y) * z \\ &\Leftrightarrow \text{(αξίωμα πολλαπλασιασμού: } 0 * x = 0) \\ \mathbf{0} &= (0 * y) * z \\ &\Leftrightarrow \text{(αξίωμα πολλαπλασιασμού: } 0 * x = 0) \\ 0 &= \mathbf{0} \\ &\Leftrightarrow \text{true} \end{aligned}$$

δείχνουμε ότι για x = 0 η πρόταση *P* είναι αληθής.

II.Βήμα επαγωγής

Θεωρούμε x ≥0 και υποθέτουμε ότι *P(x)* είναι αληθής. Προσπαθούμε ν' αποδείξουμε ότι η πρόταση *P* είναι αληθής και για x+1.

$$\begin{aligned} (x+1) * (y * z) &= ((x+1) * y) * z \\ &\Leftrightarrow \text{(επιμεριστική ιδιότητα: } x*(a+b)=x*a+x*b) \\ \mathbf{x * (y * z) + 1 * (y * z)} &= ((x+1) * y) * z \\ &\Leftrightarrow \text{(αλγεβρικές πράξεις: } 1 * x = x) \\ \mathbf{x * (y * z) + y * z} &= ((x+1) * y) * z \\ &\Leftrightarrow \text{(επιμεριστική ιδιότητα: } x*(a+b)=x*a+x*b) \\ x * (y * z) + y * z &= \mathbf{(x * y + 1 * y)} * z \\ &\Leftrightarrow \text{(αλγεβρικές πράξεις: } 1 * x = x) \\ x * (y * z) + y * z &= \mathbf{((x * y) + y)} * z \\ &\Leftrightarrow \text{(υπόθεση επαγωγής: } x * (y * z) = (x * y) * z) \\ \mathbf{(x * y) * z} + y * z &= ((x * y) + y) * z \\ &\Leftrightarrow \text{(επιμεριστική ιδιότητα: } a*x + b*x = (a+b)*x) \\ \mathbf{((x * y) + y) * z} &= ((x * y) + y) * z \\ &\Leftrightarrow \text{ true} \end{aligned}$$

Αποδεικνύοντας την ισχύ της πρότασης *P(x+1)*, προκύπτει από την αρχή της μαθηματικής επαγωγής, ότι η πρόταση *P(x)* ισχύει για όλα τα *x*∈N.

Ευρετήριο Αγγλικής Ορολογίας

Ευρετήριο Ελληνικής Ορολογίας