AUTOMATED POSITION ASSIGNMENT BY SYMBOLIC PROGRAMMING

BY

Ko Man Kwok

A Thesis
Submitted to the Faculty of Graduate Studies
in Partial Fulfillment of the Requirements
for the Degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba

© Ko Man Kwok 1993

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-85929-6

Canada

Name _____

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

ELECTRONICS AND ELECTRICAL

SUBJECT TERM

0	5	4	4

SUBJECT CODE

UMI

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION

General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523
Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language
- General 0679
- Ancient 0289
- Linguistics 0290
- Modern 0291

Literature
- General 0401
- Classical 0294
- Comparative 0295
- Medieval 0297
- Modern 0298
- African 0316
- American 0591
- Asian 0305
- Canadian (English) 0352
- Canadian (French) 0355
- English 0593
- Germanic 0311
- Latin American 0312
- Middle Eastern 0315
- Romance 0313
- Slavic and East European 0314

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy 0422
Religion
- General 0318
- Biblical Studies 0321
- Clergy 0319
- History of 0320
- Philosophy of 0322

Theology 0469

SOCIAL SCIENCES

American Studies 0323
Anthropology
- Archaeology 0324
- Cultural 0326
- Physical 0327

Business Administration
- General 0310
- Accounting 0272
- Banking 0770
- Management 0454
- Marketing 0338

Canadian Studies 0385
Economics
- General 0501
- Agricultural 0503
- Commerce-Business 0505
- Finance 0508
- History 0509
- Labor 0510
- Theory 0511

Folklore 0358
Geography 0366
Gerontology 0351
History
- General 0578
- Ancient 0579
- Medieval 0581
- Modern 0582
- Black 0328
- African 0331
- Asia, Australia and Oceania 0332
- Canadian 0334
- European 0335
- Latin American 0336
- Middle Eastern 0333
- United States 0337

History of Science 0585
Law 0398
Political Science
- General 0615
- International Law and Relations 0616
- Public Administration 0617

Recreation 0814
Social Work 0452
Sociology
- General 0626
- Criminology and Penology ... 0627
- Demography 0938
- Ethnic and Racial Studies 0631
- Individual and Family Studies 0628
- Industrial and Labor Relations 0629
- Public and Social Welfare 0630
- Social Structure and Development 0700
- Theory and Methods 0344

Transportation 0709
Urban and Regional Planning 0999
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture
- General 0473
- Agronomy 0285
- Animal Culture and Nutrition 0475
- Animal Pathology 0476
- Food Science and Technology 0359
- Forestry and Wildlife 0478
- Plant Culture 0479
- Plant Pathology 0480
- Plant Physiology 0817
- Range Management 0777
- Wood Technology 0746

Biology
- General 0306
- Anatomy 0287
- Biostatistics 0308
- Botany 0309
- Cell 0379
- Ecology 0329
- Entomology 0353
- Genetics 0369
- Limnology 0793
- Microbiology 0410
- Molecular 0307
- Neuroscience 0317
- Oceanography 0416
- Physiology 0433
- Radiation 0821
- Veterinary Science 0778
- Zoology 0472

Biophysics
- General 0786
- Medical 0760

EARTH SCIENCES

Biogeochemistry 0425
Geochemistry 0996
Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences 0768
Health Sciences
- General 0566
- Audiology 0300
- Chemotherapy 0992
- Dentistry 0567
- Education 0350
- Hospital Management 0769
- Human Development 0758
- Immunology 0982
- Medicine and Surgery 0564
- Mental Health 0347
- Nursing 0569
- Nutrition 0570
- Obstetrics and Gynecology .. 0380
- Occupational Health and Therapy 0354
- Ophthalmology 0381
- Pathology 0571
- Pharmacology 0419
- Pharmacy 0572
- Physical Therapy 0382
- Public Health 0573
- Radiology 0574
- Recreation 0575
- Speech Pathology 0460
- Toxicology 0383

Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences

Chemistry
- General 0485
- Agricultural 0749
- Analytical 0486
- Biochemistry 0487
- Inorganic 0488
- Nuclear 0738
- Organic 0490
- Pharmaceutical 0491
- Physical 0494
- Polymer 0495
- Radiation 0754

Mathematics 0405
Physics
- General 0605
- Acoustics 0986
- Astronomy and Astrophysics 0606
- Atmospheric Science 0608
- Atomic 0748
- Electronics and Electricity 0607
- Elementary Particles and High Energy 0798
- Fluid and Plasma 0759
- Molecular 0609
- Nuclear 0610
- Optics 0752
- Radiation 0756
- Solid State 0611

Statistics 0463

Applied Sciences

Applied Mechanics 0346
Computer Science 0984
Engineering
- General 0537
- Aerospace 0538
- Agricultural 0539
- Automotive 0540
- Biomedical 0541
- Chemical 0542
- Civil 0543
- Electronics and Electrical 0544
- Heat and Thermodynamics ... 0348
- Hydraulic 0545
- Industrial 0546
- Marine 0547
- Materials Science 0794
- Mechanical 0548
- Metallurgy 0743
- Mining 0551
- Nuclear 0552
- Packaging 0549
- Petroleum 0765
- Sanitary and Municipal 0554
- System Science 0790

Geotechnology 0428
Operations Research 0796
Plastics Technology 0795
Textile Technology 0994

PSYCHOLOGY

General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

Nom _____

Dissertation Abstracts International est organisé en catégories de sujets. Veuillez s.v.p. choisir le sujet qui décrit le mieux votre thèse et inscrivez le code numérique approprié dans l'espace réservé ci-dessous.

_____ ☐☐☐ U·M·I

SUJET — CODE DE SUJET

Catégories par sujets

HUMANITÉS ET SCIENCES SOCIALES

COMMUNICATIONS ET LES ARTS
Architecture 0729
Beaux-arts 0357
Bibliothéconomie 0399
Cinéma 0900
Communication verbale 0459
Communications 0708
Danse 0378
Histoire de l'art 0377
Journalisme 0391
Musique 0413
Sciences de l'information 0723
Théâtre 0465

ÉDUCATION
Généralités 515
Administration 0514
Art 0273
Collèges communautaires 0275
Commerce 0688
Économie domestique 0278
Éducation permanente 0516
Éducation préscolaire 0518
Éducation sanitaire 0680
Enseignement agricole 0517
Enseignement bilingue et multiculturel 0282
Enseignement industriel 0521
Enseignement primaire. 0524
Enseignement professionnel 0747
Enseignement religieux 0527
Enseignement secondaire 0533
Enseignement spécial 0529
Enseignement supérieur 0745
Évaluation 0288
Finances 0277
Formation des enseignants 0530
Histoire de l'éducation 0520
Langues et littérature 0279
Lecture 0535
Mathématiques 0280
Musique 0522
Orientation et consultation 0519
Philosophie de l'éducation 0998
Physique 0523
Programmes d'études et enseignement 0727
Psychologie 0525
Sciences 0714
Sciences sociales 0534
Sociologie de l'éducation 0340
Technologie 0710

LANGUE, LITTÉRATURE ET LINGUISTIQUE
Langues
- Généralités 0679
- Anciennes 0289
- Linguistique 0290
- Modernes 0291

Littérature
- Généralités 0401
- Anciennes 0294
- Comparée 0295
- Médiévale 0297
- Moderne 0298
- Africaine 0316
- Américaine 0591
- Anglaise 0593
- Asiatique 0305
- Canadienne (Anglaise) 0352
- Canadienne (Française) 0355
- Germanique 0311
- Latino-américaine 0312
- Moyen-orientale 0315
- Romane 0313
- Slave et est-européenne 0314

PHILOSOPHIE, RELIGION ET THÉOLOGIE
Philosophie 0422
Religion
- Généralités 0318
- Clergé 0319
- Études bibliques 0321
- Histoire des religions 0320
- Philosophie de la religion 0322

Théologie 0469

SCIENCES SOCIALES
Anthropologie
- Archéologie 0324
- Culturelle 0326
- Physique 0327

Droit 0398
Économie
- Généralités 0501
- Commerce-Affaires 0505
- Économie agricole 0503
- Économie du travail 0510
- Finances 0508
- Histoire 0509
- Théorie 0511

Études américaines 0323
Études canadiennes 0385
Études féministes 0453
Folklore 0358
Géographie 0366
Gérontologie 0351
Gestion des affaires
- Généralités 0310
- Administration 0454
- Banques 0770
- Comptabilité 0272
- Marketing 0338

Histoire
- Histoire générale 0578
- Ancienne 0579
- Médiévale 0581
- Moderne 0582
- Histoire des noirs 0328
- Africaine 0331
- Canadienne 0334
- États-Unis 0337
- Européenne 0335
- Moyen-orientale 0333
- Latino-américaine 0336
- Asie, Australie et Océanie 0332

Histoire des sciences 0585
Loisirs 0814
Planification urbaine et régionale 0999
Science politique
- Généralités 0615
- Administration publique 0617
- Droit et relations internationales 0616

Sociologie
- Généralités 0626
- Aide et bien-être social 0630
- Criminologie et établissements pénitentiaires 0627
- Démographie 0938
- Études de l' individu et de la famille 0628
- Études des relations interethniques et des relations raciales 0631
- Structure et développement social 0700
- Théorie et méthodes. 0344
- Travail et relations industrielles 0629

Transports 0709
Travail social 0452

SCIENCES ET INGÉNIERIE

SCIENCES BIOLOGIQUES
Agriculture
- Généralités 0473
- Agronomie. 0285
- Alimentation et technologie alimentaire 0359
- Culture 0479
- Élevage et alimentation 0475
- Exploitation des pâturages 0777
- Pathologie animale 0476
- Pathologie végétale 0480
- Physiologie végétale 0817
- Sylviculture et faune 0478
- Technologie du bois 0746

Biologie
- Généralités 0306
- Anatomie 0287
- Biologie (Statistiques) 0308
- Biologie moléculaire 0307
- Botanique 0309
- Cellule 0379
- Écologie 0329
- Entomologie 0353
- Génétique 0369
- Limnologie 0793
- Microbiologie 0410
- Neurologie 0317
- Océanographie 0416
- Physiologie 0433
- Radiation 0821
- Science vétérinaire 0778
- Zoologie 0472

Biophysique
- Généralités 0786
- Medicale 0760

SCIENCES DE LA TERRE
Biogéochimie 0425
Géochimie 0996
Géodésie 0370
Géographie physique 0368
Géologie 0372
Géophysique 0373
Hydrologie 0388
Minéralogie 0411
Océanographie physique 0415
Paléobotanique 0345
Paléoécologie 0426
Paléontologie 0418
Paléozoologie 0985
Palynologie 0427

SCIENCES DE LA SANTÉ ET DE L'ENVIRONNEMENT
Économie domestique 0386
Sciences de l'environnement 0768
Sciences de la santé
- Généralités 0566
- Administration des hôpitaux 0769
- Alimentation et nutrition 0570
- Audiologie 0300
- Chimiothérapie 0992
- Dentisterie 0567
- Développement humain 0758
- Enseignement 0350
- Immunologie 0982
- Loisirs 0575
- Médecine du travail et thérapie 0354
- Médecine et chirurgie 0564
- Obstétrique et gynécologie 0380
- Ophtalmologie 0381
- Orthophonie 0460
- Pathologie 0571
- Pharmacie 0572
- Pharmacologie 0419
- Physiothérapie 0382
- Radiologie 0574
- Santé mentale 0347
- Santé publique 0573
- Soins infirmiers 0569
- Toxicologie 0383

SCIENCES PHYSIQUES
Sciences Pures
Chimie
- Généralités 0485
- Biochimie 487
- Chimie agricole 0749
- Chimie analytique 0486
- Chimie minérale 0488
- Chimie nucléaire 0738
- Chimie organique 0490
- Chimie pharmaceutique 0491
- Physique 0494
- PolymÈres 0495
- Radiation 0754

Mathématiques 0405
Physique
- Généralités 0605
- Acoustique 0986
- Astronomie et astrophysique 0606
- Électronique et électricité 0607
- Fluides et plasma 0759
- Météorologie 0608
- Optique 0752
- Particules (Physique nucléaire) 0798
- Physique atomique 0748
- Physique de l'état solide 0611
- Physique moléculaire 0609
- Physique nucléaire 0610
- Radiation 0756

Statistiques 0463

Sciences Appliqués Et Technologie
Informatique 0984
Ingénierie
- Généralités 0537
- Agricole 0539
- Automobile 0540
- Biomédicale 0541
- Chaleur et thermodynamique 0348
- Conditionnement (Emballage) 0549
- Génie aérospatial 0538
- Génie chimique 0542
- Génie civil 0543
- Génie électronique et électrique 0544
- Génie industriel 0546
- Génie mécanique 0548
- Génie nucléaire 0552
- Ingénierie des systèmes 0790
- Mécanique navale 0547
- Métallurgie 0743
- Science des matériaux 0794
- Technique du pétrole 0765
- Technique minière 0551
- Techniques sanitaires et municipales 0554
- Technologie hydraulique 0545

Mécanique appliquée 0346
Géotechnologie 0428
Matières plastiques (Technologie) 0795
Recherche opérationnelle 0796
Textiles et tissus (Technologie) 0794

PSYCHOLOGIE
Généralités 0621
Personnalité 0625
Psychobiologie 0349
Psychologie clinique 0622
Psychologie du comportement 0384
Psychologie du développement 0620
Psychologie expérimentale 0623
Psychologie industrielle 0624
Psychologie physiologique 0989
Psychologie sociale 0451
Psychométrie 0632

AUTOMATED POSITION ASSIGNMENT

BY SYMBOLIC PROGRAMMING

BY

KO MAN KWOK

A Thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

© 1993

Permission has been granted to the LIBRARY OF THE UNIVERSITY OF MANITOBA to lend or sell copies of this thesis, to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film, and UNIVERSITY MICROFILMS to publish an abstract of this thesis.

The author reserves other publications rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's permission.

I hereby declare that I am the sole author of this thesis.

I authorize the University of Manitoba to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Manitoba to reproduce this thesis by photo–copying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

ABSTRACT

Logic programming is widely used for decision making and can be shown effective. This thesis examines the application of logic programming in addressing a problem of position assignment occurring in the Department of Electrical and Computer Engineering. The Prolog program makes extensive use of recursion and requires the formulation of an explicit priority policy. A field test using a manual assignment was made. A program characteristic of inherent commitment to a lower–priority assignment during successive refinement is overcome by a modification that re–routes recursion.

ACKNOWLEDGMENTS

I am grateful for the support of my advisor, Professor A. Jakobschuk, for many years of excellent guidance and precious opinions. I have to thank my wife, Connie Kan, for her encouragement and support in these years. My parents–in–law—Mr. and Mrs Shek Kan, and my parents—Mr. and Mrs. Chun Chung Kwok, are financially supporting me during my study. Finally, I have to thank the Department of Electrical and Computer Engineering provides computer facilities for my work.

Table of content

Abstract iv

Acknowledgments v

Table of content vi

List of figures vii

List of tables viii

Chapter 1 Introduction 1

1.1 The Assignment Problem 1

1.2 Programming with Prolog 2

1.3 Overview 3

Chapter 2 Preference Matrix 4

2.1 Priority Considerations 4

2.2 Priority among Courses 5

2.3 Priority among Applicants 7

Chapter 3 Availability and Replaceability 9

3.1 Replaceable Applicant 9

3.2 Available Applicants 10

3.3 Recursion and Replaceability 11

Chapter 4 Prolog and Closed–World Assumption 13

4.1 CWA and Negation 13

4.2 Negative Rule Head 14

Chapter 5 Program Descriptions 16

5.1 Data Predicates 16

5.2 Top Predicates 17
5.2.1 First Matching Process 19
5.2.1.1 The Role of sub_first(PRIORITY,SUB_PRIORITY) 21
5.2.1.2 Computation Procedure of the First Matching Process 22
5.2.2 Second Matching Process 24
5.2.2.1 Secondary Refinement 26
5.2.2.2 Choosing an Uncommitted Substitution 28
5.2.2.3 Replacement for an Assigned Substitute 30

Chapter 6 Field Testing of Assignment Program 35
6.1 Comparison of Both Assignments 36
6.2 Characteristics of the Program 38
6.3 Alternative Assignment Process 39

Chapter 7 Conclusion 41

Appendix A Utility Predicates 44

Appendix B Program Listing 49

Appendix C Assumed Course Priority used in Chapter 6 60

Appendix D The Information Used in Chapter 6 63

Appendix E Sample Goal Call 68

Bibliography 70

List of figures

Figure 1. Assigned Order in the Program 6

Figure 2. Preference–Order Lattice 7

Figure 3. Priority of the Applicants Relative to a Course 8

Figure 4. The Case of Five Applicants and Five Open Positions 18

Figure 5. The Case of Three Applicants and Three Vacancies 30

List of tables

Table 1. Priority Pairs (Priority, Subpriority) of Courses in the Department 5

Table 2. Priority Matrix of the Applicants to a Course 8

Table 3. Truth Table for `assigned/1`, `not_assigned/1` and the Negations 14

Table 4. Course Suitability for Applicants 18

Table 5. Priority Levels of Courses 19

Table 6. Assignments of the Applicants to Courses after the First Matching Process 24

Table 7. Assignments of the Applicants to Courses with Added Definition of 'Available' 29

Table 8. Final Assignment of the Applicant–Course Links 34

Table 9. Computer–Generated Assignment 36

Table 10. Actual Assignment 37

Chapter 1

INTRODUCTION

1.1 THE ASSIGNMENT PROBLEM

Each year, in the Department of Electrical Engineering at the University of Manitoba, graduate students are employed as teaching assistants. More than one hundred work units are provided for the students in each term. Usually, more than fifty students apply for the posted units. An applicant may receive two to four work units in each of the Fall and Winter terms.

However, the assignments of applicants are not on a first–come–first–serve basis, but on the following terms: professor's request, student's preference, course history and experience.

The officer in charge of this program is confronted with the challenge of selecting appropriate assignments. He must match, as judiciously as he can, the teaching assistants to each particular course. His task is made harder because of several restrictions: the graduates from Computer Engineering cannot handle certain courses in Electrical Engineering. Moreover, the department does not have the detailed records of students who graduated from other institutions. If students apply for an assignment with only partial information, then the officer is hampered in making the right choice as to which courses are suitable for them.

One realizes, therefore, that the officer must grapple with many difficulties, such as deciding which courses are suitable for the student, not finding suitable applicants for some courses and having too many suitable students for other courses.

1.2 PROGRAMMING WITH PROLOG

With the advent of logic programming, machine reasoning about symbolic information by deductive strategies has become commonplace. Prolog is a declarative programming language which provides an environment for declaring symbolic information as facts or rules. Each Prolog rule of the form

$$a :- b_1, b_2, \ldots, b_n$$

consists of a rule head a and body $[b_1, b_2, \ldots, b_n]$. The body is a conjunction of atomic subgoals implying the rule head. A rule without a body is considered a fact [4]. A predicate of name p is an atomic declaration involving zero or more object parameters in a relationship [3]:

$$p(X_1, X_2, \ldots, X_n).$$

Closed World Assumption (CWA) includes the negation of a positive literal as an extension of the database [2]. However, the extension assumed by CWA may be inconsistent with the database, even if the database is consistent [1]. The extension of a set of consistent Horn clause by CWA is consistent. Prolog implementation of Horn clauses uses CWA and returns a false value. With sufficient positive information, the extension is believed to be consistent.

In addition, unification provides computational invertibility, that is, a variable can be either input or output of a procedure. If left uninstantiated in a procedure call, the variable will behave as output.

1.3 OVERVIEW

The presented work describes a problem–solving technique in demonstrator assignment. Chapter 2 details how priorities can be attributed to courses and applicants according to the difficulties in matching. Chapter 3 examines how to handle the availability of applicants. Chapter 4 discusses Closed Word Assumption(CWA) and the negation operator, `not`, in Prolog. Chapter 5 presents the data predicates and rule predicates of the program by means of priority policies and recursive subroutines. Chapter 6 discusses the result of comparing the computed assignment of the program with a manual assignment in a particular term, examines the characteristics of the program, and suggests an alternative method in implementing the suggested solutions. Chapter 7 concludes with the discussion of the reliability of the program.

Chapter 2

PREFERENCE MATRIX

Typical assignment problems are as follows:

1. Some of the courses have only a limited number of suitable applicants. If some of these applicants have already been assigned to other courses, the choices of making alternate assignments are drastically reduced.
2. The same course may be chosen by many of the applicants. When the number of applicants significantly exceeds the number of vacancies available for a course, then making the right choice becomes difficult.

2.1 PRIORITY CONSIDERATIONS

The higher the level of the course, the greater the challenge in finding a suitable candidate for the course. Therefore, the higher–level courses are assigned with higher level of priority in searching for their suitable and available applicants. On the other hand, owing to the fact that it is more difficult to find a suitable candidate for courses in Computer Engineering than for those in Electrical Engineering, the courses in the former program are given the highest subpriority in each priority level. Priority is the governing criteria when

the program is searching for a solution. This is because the program searches for the first data, which satisfies the problem goal or subgoal that appears in the database. Priority policy in the input data is required to deal with preference order. Moreover, the program will choose the applicants for the course according to the request of the course instructors and the applicants' preference and experience.

2.2 PRIORITY AMONG COURSES

In the presented program, two priority policies are implemented, which are called course priority value, and assignment–determining value. The course–priority value involves the level of the course with the higher–level courses carrying a preference label. In each level of priority, there exists several minor priorities. The Computer–Engineering courses are given the highest subpriority, next the Electrical Engineering courses, followed by common courses for both programs among the course in the same major priorities. Such two–dimensional priority order is shown in figure 1.

	Priority Level		
Program	4th–year course	3rd–year course	2nd–year course
Computer	1,1	2,1	3,1
Electrical	1,2	2,2	3,2
Both	1,3	2,3	3,3

Table 1. Priority Pairs (Priority, Subpriority) of Courses in the Department

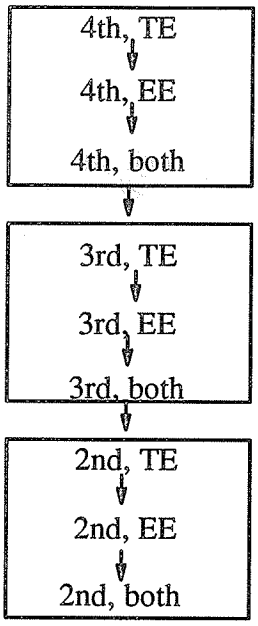

Figure 1. Assigned Order in the Program

This priority order is not rigid. The criterion determining the course priority level for the courses depends on which program has more applicants. From time to time, different situations may influence the properties of the courses. For example, the third–year Computer Engineering courses are not necessarily assigned a lower–level priority than the fourth–year Electrical Engineering courses. For instance, if there are more Computer Engineering graduate students than require for the courses in the program in a year, then the assigned priorities of the courses in this proposed method would be invalid. The preference–order lattice is shown in Figure 2; it forms a partial order.

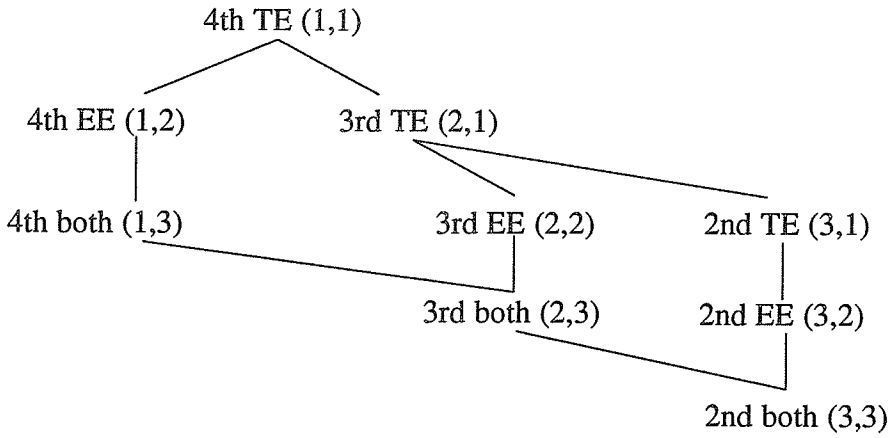

Figure 2. Preference–Order Lattice

2.3 PRIORITY AMONG APPLICANTS

For the assignment–determining value, the highest rating would be given to the applicants who are preferred by the course instructors. Then would follow the applicants with previous experience who prefer to be assigned in that course. The lowest priority refers to applicants who choose to be assigned in that course but lack experience. The program will first search for the applicants with the highest possible value. If none of the available applicants satisfies the above category, the program will skip that course and look for the other courses in the database.

When deciding on suitable applicants, assignment–determining values are used. This value gives the course instructors an opportunity to choose their favorite applicants and gives the students advantages in applying for their favorite course. The applicants may be asked to indicate their favorite courses when applying for an assignment. In addition, the

experience of their TA assignment in the previous year is taken into account. The priority policy in assigning students to a course is shown in Table 2. The order of the priority among applicants is shown in Figure 3.

	Experience	
Category	With	Without
Instructor Preference	1, 1	1, 2
Applicant Preference	2, 1	2, 2

Table 2. Priority Matrix of the Applicants to a Course

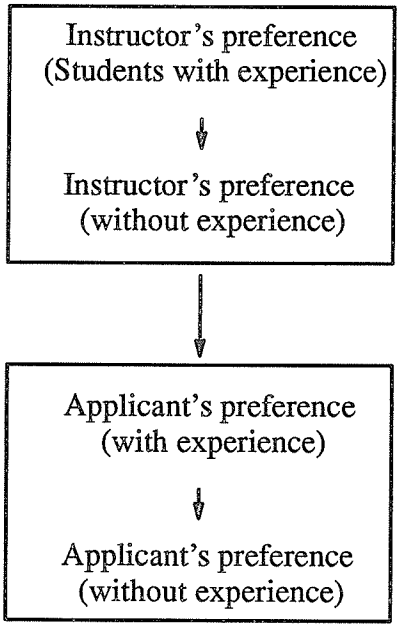

Figure 3. Priority of the Applicants Relative to a Course

Chapter 3

AVAILABILITY AND REPLACEABILITY

Definition 1: A **free applicant** is an applicant who has not yet been assigned to any course in an assignment process.

Definition 2: An **assigned applicant** is an applicant who is currently committed to a course in an ongoing assignment process.

An applicant may be assigned only one vacancy. Therefore, an assigned applicant is not eligible for another vacancy. The program searches for a free applicant when it confirms the existence of a vacancy. Nevertheless, if the program can admit only a free applicant to one of the courses, it will then have to try a number of combinations before an acceptable solution can be achieved.

3.1 REPLACEABLE APPLICANT

Definition 3.1: A **primary replaceable applicant** is an assigned applicant whose assignment can be filled by another suitable, free applicant.

By Definition 3, when the database is lacking a free applicant to fill a vacancy, there may exist some primary replaceable applicant as an alternative to choose from. The choice of applicants is thereby extended by this definition. Moreover, if an assigned applicant can

be substituted by another assigned (but replaceable) applicant, the search of an applicant for filling a vacancy may extend to all of the suitable applicants for a course as long as these are replaceable.

Definition 3.2: An ***n*–ary replaceable applicant** is an assigned applicant whose assignment can be filled by a (*n*–*1*)–ary replaceable applicant.

Definition 3.2 suggests that an assigned applicant is replaceable if the assignment of the applicant can be filled by a free applicant or another assigned applicant who is replaceable.

3.2 AVAILABLE APPLICANTS

In a dynamic assignment process, a student is considered to be available for a particular course allocation if the student applied for a vacancy. Before an applicant is assigned to a course, the program has to check if an applicant is available.

The following rules explain the availability of an applicant:

```
available(X):-applicant(X),not_assigned(X).        [4.1]
available(X):-replaceable(X).                      [4.2]
```

Rule [4.1] states that an applicant is available to be assigned to a course, unless the applicant has already been assigned to a course. Next, Rule [4.2] states that a replaceable applicant is available. An applicant is considered not available if the applicant does not satisfy these rules.

Together [4.1] and [4.2] compose a definition as follows:

Definition 4: An **available applicant** is an applicant who is free or replaceable.

3.3 RECURSION AND REPLACEABILITY

The definition of replaceable can be defined in Prolog by the following statement:

```
replaceable(X):-
assignment(X,COURSE),available(Y),suitable(Y),
X =\= Y.
```

That is, the assignment of a replaceable applicant X can be filled by another available applicant who is suitable for COURSE. On the other hand, the definition of available needs a replaceable applicant. It then forms a recursive, interactive definition among the definition of replaceable and available. Determining the availability of an free applicant can be done easily by checking the absence of an assignment about the applicant by means of the Closed–World Assumption. But, to decide the availability of an assigned applicant X, the following steps are done:

Steps

1.	Determine if X is primary replaceable. If not, go to step 2.
2.	Determine if *X* is secondary replaceable. If not, go to step 3.
⋮	⋮
n–1.	Determine if *X* is *(n–1)*–ary replaceable. If not, go to step n.
n.	Determine if *X* is *n*–ary replaceable. If not, X is assumed unavailable.

If X is determined to be *i*–ary replaceable, where $1 \leq i \leq n$, then the process stops at step *i*. The maximum value of *n* would be the number of assigned applicants in the database. The maximum value of the assigned applicants would be the total number of applicants

minus 1. This is because there exists at least one free applicant for the substitution of the primary replaceable applicant.

Chapter 4

PROLOG AND CLOSED–WORLD ASSUMPTION

In the early version of Prolog, the program returned a "?" symbol for unprovability. Because of the CWA, a Prolog program now returns a `no` if the statement is not provable. The `not` operator, which defines unprovability and CWA–negation, is not permissible in a rule head.

4.1 CWA AND NEGATION

CWA assumes all the positive information about the world has been supplied to the database. Every statement which cannot be proved true will be considered false. That is, CWA includes all the negation of a positive literal,which is not provable, as an extension of the database. The extension of CWA of a consistent database containing Horn clauses is consistent [Genesereth 1986]. For non–Horn database containing non–Horn clauses, the extension from CWA may be, but not always, inconsistent. Prolog is restricted to Horn clauses, therefore, the CWA extension of a Prolog program is consistent.

In Prolog, the negation operator is `not(P)`. However, the `not` operator in Prolog is, sometimes not adequate to present a negative meaning of a positive statement. If

`assigned(X)` represents `X` is an assigned applicant, Prolog will return a true value for the query--`not(assigned(X))`, when `X` is not an applicant. It is because `assigned(X)`, which is not possible, is not provable by the program. If `not(assigned(X))` is used to represent a free applicant as in definition 1, the program may misplace a non–applicant into a course.

4.2 NEGATIVE RULE HEAD

A statement with a form $\neg B \leftarrow A$ can be defined in Prolog with the name convention `not_B`, using an underscore and the name of the literal. This method introduces a definite meaning for a negative statement. For the status of free applicant, `not_assigned(X)` can be formulated as:

```
not_assigned(X):-applicant(X),not(assigned(X)).
```

This formula not only checks if the assignment is provable, but also checks if `X` is an applicant. Table 3 shows the difference of `not(assigned(X))` and `not_assigned(X)`. Although this two statements have the similar meaning, the truth values are different.

`applicant(X)`	`assigned(X)`	`not_assigned(X)`	`not(assigned(X))`
True	True	False	False
True	False	True	True
False	True	False	False
False	False	False	True

Table 3. Truth Table for `assigned/1`, `not_assigned/1` and the Negations

`not_assigned/1` fails when `X` is not an applicant, and may then represent the definition of a free applicant.

In mathematical logic, $\neg\neg B=B$ is valid. In Prolog, the `not(not(P))` gives the same response to `P`. But `not(not_B)` may differ from `B`. From the previous example of `assigned(X)` and `not(not_assigned(X))`, such difference occurs again when `X` is not an applicant. However, `not(not_assigned(X))` gives a true value when `X` is not an applicant and may be misunderstood as a assigned applicant. Therefore, it is recommended that the parameters of the predicate should be checked before the use of the not operator. This is called the Domain Closure Assumption (DCA). DCA suggests that axioms should only consider objects described by the world. Here, only the world of applicants is considered when checking for an assigned applicant. DCA restricts CWA–extension.

Chapter 5

PROGRAM DESCRIPTIONS

This chapter discusses the database representations and the implementation of how the program is designed in solving the problem mentioned in chapter 1.

5.1 DATA PREDICATES

```
final_event(EVENT).
```

EVENT represents the number of a current assignment being made by the program. The value of EVENT is dynamically updated in the assignment process.

```
course(COURSE,LEVEL,DEPARTMENT).
```

This predicate indicates course information. COURSE indicates the course number. LEVEL represents the level of the course and DEPARTMENT shows which department the course is in.

```
unit_left(COURSE,UNITS_LEFT).
```

This predicate reveals how many work units are left for the course. UNITS_LEFT is an integer which may decrease by the number of units which have been assigned to an applicant in the course COURSE.

`prof_assign(COURSE,STUDENTS).`

Student preference of the course instructor is given by the list of students, `STUDENTS`.

`student_info(STUDENT,PREFERENCE,EXPERIENCE,DEGREE).`

This predicate provides the information about an applicant. His preference and previous experience are given by `PREFERENCE` and `EXPERIENCE`, respectively. `DEGREE` displays the degree that the applicant is holding.

5.2 TOP PREDICATES

For any assignment problem, the program will match each course, one at a time, with the candidates. The courses are classified as to different priority levels according to which year the courses belong to. In each of the priority levels, the courses are again categorized into different subpriority levels according to the year–level and the courses program they belong to.

To determine the most suitable candidate for a course, two matching processes are required. The top predicate `first/1` finds the number of suitable and free candidates for all course positions. Another top predicate `second/1` tries to determine candidates for the remaining courses which still have positions not matched with a suitable applicant.

There are several reasons why a course may lack a suitable candidate. For one, there could be more positions in a course than suitable candidates for it. Secondly, suitable applicants could already be assigned to other courses. As an example, consider five available positions and five applicants. The course suitability and the order of applicants as appear

in the database are given in Table 4, and the priority levels of the courses are listed in Table 5.



Figure 4. The Case of Five Applicants and Five Open Positions

Applicants	Course Suitability Lists
A	1, 4
B	3, 5
C	2, 3
D	1, 2
E	1

Table 4. Course Suitability for Applicants

Courses	Course Priority Levels
1	1, 1
2	1, 3
3	2, 3
4	3, 1
5	3, 3

Table 5. Priority Levels of Courses

5.2.1 FIRST MATCHING PROCESS

The first matching process proceeds under the predicate `first(PRIORITY)`. `PRIORITY` controls the search of the courses and has a starting value 1 which represents the highest priority level. The program calls `sub_first(1,1)` with priority indices recursively incremented up to three, which is the lowest level:

```
first(PRIORITY):-
PRIORITY =< 3,
sub_first(PRIORITY,1),
PRIORITY1 is PRIORITY + 1,
first(PRIORITY1).
```

`sub_first(PRIORITY,SUB_PRIORITY)` scans all the subpriorities in the current priority level. In each of the subpriority levels, the program finds a course that matches current priority and subpriority levels, one at a time. For each course thus identified with the designated priority and subpriority levels, the program attempts to find the required number of suitable and available candidates. The process concludes when the requested

number of candidates has been satisfied or no more suitable and available candidate for that course can be found. After `sub_first(PRIORITY,1)` has established all the suitable and available applicants for the courses in a particular priority level, it increments the value of `PRIORITY` by one, becoming `PRIORITY1` and starts the recursive call `first(PRIORITY1)`. The incremented goal is called recursively for the assignment to be continued at the next lower priority level, until the value of `PRIORITY` exceeds 3.

```
first(PRIORITY):-
tell('ans1'),
write('The actions from the first trial'),
nl,
write('--------------------------------'),
nl,
nl,
listing(assign),
nl,
nl,
write('The assignment of the first trial'),
nl,
write('--------------------------------'),
nl,
solution,
told,
!.
```

Upon termination of recursion of `first/1`, the program proceeds to the second rule of `first/1` and creates a separated file, `ans1`, of the computed assignment for the convenience of the user. The existing assignment is asserted in the database and is retained for the second part of the program.

5.2.1.1 THE ROLE OF `sub_first(PRIORITY,SUB_PRIORITY)`

`sub_first(PRIORITY,SUB_PRIORITY)` is the major subroutine called by `first(PRIORITY)`. The function of `sub_first(PRIORITY,SUB_PRIORITY)` is to find a course–matching priority and subpriority levels as the pair `(PRIORITY, SUB_PRIORITY)`. For each course, the program will check for position vacancies and try to find a suitable applicant for the course. A suitable applicant is identified on the basis of applicant–priority with respect to that course. This requires the considerations of applicants requested by the instructor of the course where students with experience have higher priority than those who lack experience.

If a suitable applicant is not yet assigned, an assignment is asserted into the database, or else the next suitable applicant is pursued. The program calls the procedure `good(COURSE,STUDENT,CATEGORY,assign)` to execute the database deposition. The program continues to search until no vacancy remains, or no more suitable and available applicant can be found:

```
sub_first(PRIORITY,SUB_PRIORITY):-
course(COURSE,PRIORITY,SUB_PRIORITY),
check_unit(COURSE),
suitable_TA(COURSE,STUDENT,CATEGORY),
not_assigned(STUDENT),
good(COURSE,STUDENT,CATEGORY,assign),
final_event(EVENT),
retract(final_event(EVENT)),
EVENT1 is EVENT + 1 ,
assert(final_event(EVENT1)),
fail.
```

The `fail` operator invokes backtracking upon assignment deposition. Backtracking returns control to `choose_TA/3` in order to choose another TA when required for the

current course. Assignment completion for a course returns backtracking further to course/3, introducing the next course with vacancies in the assignment process.

When all courses with the current subpriority level have been considered in this manner, the first sub_first/2 rule fails, introducing the second rule that repeats this whole process with incremented SUB_PRIORITY.

```
sub_first(PRIORITY,SUB_PRIORITY):-
SUB_PRIORITY =< 3,
SUB_PRIORITY1 is SUB_PRIORITY + 1,
sub_first(PRIORITY,Sub_priority1).
```

Because there is no course with such subpriority level greater than three, the termination of recursion is finally set when the subpriority level exceeds three:

```
sub_first(PRIORITY,SUB_PRIORITY):-
!.
```

5.2.1.2 COMPUTATION PROCEDURE OF THE FIRST MATCHING PROCESS

By using the data order as given in Tables 4 and 5, the program examines if a course has a vacancy according to its priority standing. The program searches among the courses with lowest priority and subpriority values by activating first/1 and sub_first/2, respectively. Course 1 is attended first in course/3 and is found to have a vacancy by check_unit/1. The program then looks for a suitable applicant by calling suitable_TA/3 in the database for this course. Applicant A is first examined and is found suitable for this course. not_assigned/1 in the next step confirms that Applicant A has not been assigned to any course yet. Hence, Applicant A is assigned to course 1 by calling good/4.

After applicant A has been assigned, the program proceeds to the next subpriority level 2 and finds no course with the incremented subpriority. Then, the program goes to the subpriority level 3 and discovers course 2. Again, the program determines if any suitable and free applicant in the database is suitable for this course. `suitable_TA/3` fails with applicant A because of the ineligibility for course 2. The program then backtracks to `student/4` within `suitable_TA/3` and finds applicant B. The trial fails again because applicant B is not suitable for this course as well. `student/4` thus proceeds to applicant C. Applicant C is found to be suitable for course 2 by `suitable_TA/3`. The program then checks the availability of this applicant by means of `not_assigned/1`. Finding that the applicant is not committed to any course, the program assigned applicant C to course 2 by good/4.

After all courses in priority level 1 have been processed, the program proceeds to the next priority level. By using the same method, the program assigns applicant B to course 3, but none to the remaining courses.

A fresh assignment of applicants is given in Table 6, using the data order as appears in Table 4 that would result from `first(1)`.

Applicants	Suitability List	Courses Assignment by calling `first(1)`
A	1, 4	1
B	3, 5	3
C	2, 3	2
D	1, 2	not assigned
E	1	not assigned
none	N/A	4 (vacant)
none	N/A	5 (vacant)

Table 6. Assignments of the Applicants to Courses after the First Matching Process

The applicant–course links are retained in the database for the next matching process, and are printed on a separated file, `ans1`. In the next matching process, the program will try to furnish the unfilled courses with assigned applicants whose assignments can be replaced by available applicants.

5.2.2 SECOND MATCHING PROCESS

The first matching process is a straight–forward searching routine. Due to the large number of combinations, the assignment is unlikely to be complete. Therefore, a second matching process is necessary to refine the assignments computed by the first matching process.

The second matching proceeds under the predicate, `second(PRIORITY)`. `PRIORITY` represents the priority level of a course and starts from the value one(as it does in the procedure `first(PRIORITY)`). The routine is similar to that of `first/1`, except that `second/1` calls the subroutine `sub_second(PRIORITY,SUB_PRIORITY)`:

```
second(PRIORITY):-
PRIORITY =< 3,
sub_second(PRIORITY,1),
PRIORITY1 is PRIORITY + 1,
second(PRIORITY1),
!.
```

The purpose of `sub_second(PRIORITY,SUB_PRIORITY)` is to find the courses with vacancies in priority level `PRIORITY`. `SUB_PRIORITY` is initially instantiated to unity when passed to `sub_second/2`. When such courses are identified, the program will try a new assignment strategy. For each of the vacancies, instead of allocating a not–yet assigned applicant as done in `first(PRIORITY)`, the program reconsiders an already–assigned applicant from another course. Then, the program will try to find a replacement in the original assignment of the former applicant. The substitute must be suitable for the course as provided in the primary data. If the substitute is not committed to another course, the replacement process terminates. Otherwise, the program continues seeking a suitable applicant and tries to find another replacement.

Consider a vacancy in course 7, for example. The program looks for an applicant X suitable for course 7, even though this applicant is currently bound to course 8. Then the program looks for another applicant Y suitable for 8 who is not committed to any course. This would complete the process by assigning X to 7 and Y to 8. If Y is already bounded, the program searches for an applicant assigned to a course other than 7 or 8 and recurs on the previous process. The process continues until an uncommitted assigned applicant is accommodated or all possibilities have been tried. Such operation is performed by `sub_second(PRIORITY,1)`.

```
second(PRIORITY):-
tell('ans2'),
write('The action after the second trial:'),
nl,
write('-----------------------------------'),
nl,
nl,
listing(assign),
listing(move),
nl,
nl,
write('The assignment after the second trial'),
nl,
write('-------------------------------------'),
nl,
solution,
told,
!.
```

All the results of the computed assignments are placed into a text file. However, the new information will not overwrite the assignments in the first matching process, but enter all information into a new file, `ans2`, instead. From the two files, the user may infer how the assignment was done.

5.2.2.1 Secondary Refinement

The assignments obtained by `first/1` are refined by the subroutine `second(PRIORITY,SUB_PRIORITY)`. As done in `sub_first/2`, the program starts at the highest subpriority 1 in the current priority level, `PRIORITY`. For each course that matches the pair `(PRIORITY, SUB_PRIORITY)`, the program checks for a vacancy. With the completion of the first matching process, any applicant not assigned is unsuitable for the remaining vacancies. Therefore, the program must reexamine an assigned applicant

who is also suitable for a course vacancy. Such applicant is identified by calling `second_choice_TA(COURSE,STUDENT)` that re–assigns it to another course:

```
sub_second(PRIORITY,SUB_PRIORITY):-
course(COURSE,PRIORITY,SUB_PRIORITY),
check_unit(COURSE),
second_choice_TA(COURSE,STUDENT),
final_event(EVENT),
retract(FINAL_EVENT(EVENT)),
EVENT1 is EVENT + 1 ,
assert(FINAL_EVENT(EVENT1)),
fail.
```

Attention to all the not–yet assigned work units in the course is focused by forced backtracking initiated by the `fail` operator, thereby exhaustively attending to all vacancies. Then backtracking returns to `course/3`, examining all the remaining courses of a specified level pair `(PRIORITY,SUB_PRIORITY)`.

After all the courses in the current subpriority level have been processed, the control passes to the second rule of `sub_second/2` which starts a recursive call and repeats the matching process for courses in the next subpriority level:

```
sub_second(PRIORITY,SUB_PRIORITY):-
SUB_PRIORITY =< 3 ,
SUB_PRIORITY1 is SUB_PRIORITY + 1,
sub_second(PRIORITY,SUB_PRIORITY1).
```

Quite similarly to `sub_first/1` in `first/1`, recursion is terminated when `SUB_PRIORITY` reaches the value 4.

```
sub_second(PRIORITY,SUB_PRIORITY):-
!.
```

5.2.2.2 CHOOSING AN UNCOMMITTED SUBSTITUTION

From Table 6, not only D and E fail to receive consideration, but also courses 4 and 5 remain vacant (in that order). However, if the assignment of an applicant can be replaced by another applicant who is not assigned to any course, then there exists two alternative vacancies for the two applicants instead of one. Hence, the flexibility of the assignment increases.

Elaborating on the alternate choice for already–assigned work units, the predicate `second_choice_TA(COURSE,STUDENT)` first searches for any student who is suitable for a vacancy. Such applicant must be assigned to some other course before the program can proceed looking for a suitable and not–yet assigned applicant in the first matching process.

```
second_choice_TA(COURSE,STUDENT):-
COURSES_TRIED = [COURSE],
assigned_TA(COURSE,COURSE1,STUDENT,CATEGORY,
COURSES_TRIED,[]),
TA_TRIED = [STUDENT],
replace_assigned_TA(COURSE1,COURSES_TRIED,TA_TRIED),
good(COURSE,STUDENT,CATEGORY,move).
```

A substitution is found by calling the subroutine `replace_assigned_TA/3`. If a substitute should exist, the program assigns the applicant by instantiating `COURSE1` within the subroutine `replace_assigned_TA/3`. Subsequently, `good/4` moves the former applicant into the still–vacant course from the current assignment.

Returning to the above example in `second/1`, the program searches for available work units. When `PRIORITY` is 1, no course with a vacancy can be found. In the next priority level, course 3 has no vacancy. The following one being encountered is course 4

and an vacancy is found. Then the program tries to match the course with an already–assigned applicant with the course. Applicant A is the result from exploring the list of the already–assigned applicants. Then the program inspects if any of the free applicants suitable for course 1 may replace A. `replace_assigned_TA/3` fails when trying applicant B and C within `second_choice_TA/2`. Next, `replace_assigned_TA/3` attempts a substitution with applicant D. Applicant D is found suitable for course 1 by `suitable_TA/3` and `not_assigned/1`, then is assigned to course 1 in `replace_assigned_TA/3` by calling `good/4`. With a replacement for the assignment in course 1, applicant A is able to be transferred to course 4 by `good/4` in `second_choice_TA`. The resulting assignment is given in Table 7.

Applicants	Suitability List	Assignment by calling `first(1)`	Assignment after `first(1)` by calling `second(1)` (with replacement an assigned applicant by an free applicant)
A	1, 4	1	4
B	3, 5	3	3
C	2, 3	2	2
D	1, 2	not assigned	1
E	1	not assigned	not assigned
none	N/A	5 (vacant)	5 (vacant)

Table 7. Assignments of the Applicants to Courses with Added Definition of 'Available'

On the other hand, when the program searches for a replaceable applicant for course 5, B is the only applicant found to be suitable for this course. However, the only

uncommitted applicant, E, is not suitable for the current assignment of B in course 2. Thus, this method fails to transfer B from course 2 to course 5. None of the remaining assigned applicants is suitable for course 5. Hence, no link for course 5 with any of the applicant can be made, and applicant E remains free as well. This shows the increased flexibility when involving *n*–ary replaceability.

5.2.2.3 REPLACEMENT FOR AN ASSIGNED SUBSTITUTE

An applicant can be assigned to a course only if the applicant is available during a continuing assignment process. A currently–assigned applicant is said to be available if he is 'replaceable' according to a precise definition. As an example,

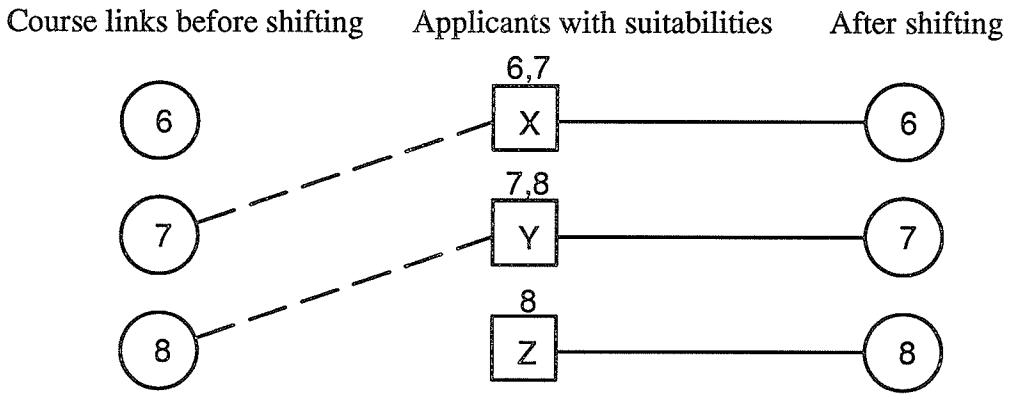

Figure 5. The Case of Three Applicants and Three Vacancies

Consider a vacancy in course 6 and X being suitable for this course, currently assigned to course 7. On the other hand, applicant Y is suitable for course 7 and currently assigned to course 8. Applicant Z is suitable for course 8 and is as yet uncommitted. (Figure 5)

The situation is thus that applicant X can be switched from course 7 to course 6 and applicant Y from course 8 to course 7. Then, the vacancy in course 6 will be transferred to course 8 where the new vacancy in course 8 can be filled by applicant Z. This transfer of assignment can be done only if a free applicant can be found for course 8.

Replaceability is defined by two Prolog rules:

```
replaceable(APPLICANT):-
assign(APPLICANT,COURSE),
suitable(APPLICANT2,COURSE),
not_assigned(APPLICANT2).

replaceable(APPLICANT1):-
assign(APPLICANT1,COURSE1),
suitable_TA(APPLICANT2,COURSE1),
replaceable(APPLICANT2).
```

In this example of three applicants, applicant 6 is said to be replaceable if there exists another applicant (here applicant 7) who is suitable for course X and fulfils either one of the following conditions:

(i) the other applicant is not assigned to any course,
(ii) the other applicant is assigned to another course and is replaceable by a new applicant who is free or replaceable.

This definition of the `replaceable/1` recursive procedure enhances the intelligence of the assignment process with rule 1 forming the boundary case. The boundary condition is satisfied by the discovery of a uncommitted applicant.

Furthermore, a restriction is needed other than the boundary condition, to prevent the infinite recursive call. The program collects courses and applicants visited so far in two separated lists to be avoided in the next recursive exploration.

The predicate `replace_assigned_TA/3` enables the determination of the existence of an uncommitted substitute in the database for course COURSE:

```
replace_assigned_TA(COURSE,COURSES_TRIED,_TA_TRIED):-
course(COURSE,PRIORITY,SUB_PRIORITY),
student_info(STUDENT1,_,_,_),
not_assigned(STUDENT1),
suitable_TA(COURSE,STUDENT1,CATEGORY),
change_unit_left(COURSE,2),
good(COURSE,STUDENT1,CATEGORY,assign),
!.
```

This first rule of `replace_assigned_TA/3`, while finding a replacement, is the same searching process as in the first matching process. If there is any free applicant suitable for COURSE, the program terminates the search. Before the termination, the applicant would be assigned into COURSE. This procedure is done by calling `change_unit_left/2` and `good/4`. However, if no such applicant is obtained, a recursive second rule of `replace_assigned_TA/4` tries searching for a replacement again from the list of the assigned applicants:

```
replace_assigned_TA(COURSE,COURSES_TRIED,TA_TRIED):-
append([COURSE],COURSES_TRIED,COURSE_TRIED1),
try_assigned_TA(COURSE,COURSE1,STUDENT,CATEGORY,
COURSES_TRIED1,TA_TRIED),
append([STUDENT],TA_TRIED,TA_TRIED1),
replace_assigned_TA(COURSE1,COURSES_TRIED1,TA_TRIED1),
change_unit_left(COURSE,2),
good(COURSE,STUDENT,CATEGORY,move).
```

This recursive process finds an assigned applicant STUDENT other than any already tried and assigned applicant in the list TA_TRIED. Such applicant must be suited to COURSE, but still committed to any course that is not listed in COURSES_TRIED. The lists are used to prevent the occurrence of infinite recursive call. The program adds STUDENT to TA_TRIED and tries finding a substitute for STUDENT in COURSE1. If a substitute exists, the program will assign the substitute into COURSE1, then assign STUDENT into COURSE by the subroutines change_unit_left/2 and good/4.

Returning to the example in Figure 4, while finding an available applicant for course 5, no uncommitted applicant is suitable for the replacement of applicant B in course 3. Therefore, it is necessary to refer back to the list of applicants who have already been assigned for substituting the current assignment of applicant B. The subroutine replace_assigned_TA/3 is responsible for matching COURSE with another available applicant, with COURSES_TRIED1 includes COURSE and TA_TRIED1 includes STUDENT to prevent the infinite recursive call.

try_assigned_TA/6 searches for anyone who has been assigned to any course other than course 3, but suited to it. The search starts with applicant A but fails because A is not suited to the course. The next applicant found is B. It fails again because B is listed in TA_TRIED. The subroutine then tries applicant C and finds that C is suited to course 3 and is assigned to course 2. Hence the subroutine starts the recursive searching for an replacement for C in course 2 by replace_assigned_TA/3, with applicant C added in the list TA_TRIED1 and course 2 added in COURSES_TRIED1.

The searching for replacement of applicant C in course 2 causes applicant D to be transferred from course 1 to course 2 and E is assigned to course 1. Hence C is moved from

course 2 to course 3. Then the replacement for applicant B in course 3 is completed and B is instead assigned to course 5.

		Assignment by calling	Assignment after `first(1)` by calling	
Applicant	Suitability	`first(1)`	`second(1)` (with replacement of an assigned applicant by a free applicant)	`second(1)` (with replacement of an assigned applicant by a replaceable applicant)
A	1, 4	1	4	4
B	3, 5	3	3	5
C	2, 3	2	2	3
D	1, 2	N/A	1	2
E	1	N/A	N/A	1

Table 8. Final Assignment of the Applicant–Course Links

At this point, the whole assignment process is completed. The applicant–course links are stored in a separated file, `ans2`, as in the first matching process, `first(1)`.

Chapter 6

FIELD TESTING OF ASSIGNMENT PROGRAM

In this chapter, an actual departmental assignment is used as the basis of testing the performance of the algorithm. The program is supplied with course and applicant data. Some applicant information regarding academic qualification was not available because of registration confidentiality and incomplete application forms. Requested applicant lists from the course instructor were not available, since with manual assignment such consideration depends on verbal understanding. The assignments generated by the program are then compared with the actual assignments for that term.

6.1 COMPARISON OF BOTH ASSIGNMENTS

The assignments obtained by calling `first(1)` and subsequently `second(1)` are tabulated below. The result is unique unless the order of base facts is changes. The underlined student numbers indicate agreement with the manual assignment.

Course Number	Applicants
210	5704146, 5618797, 5078906, 5275920, 5275581, 5507100, 5270614, 5807608
222	1234571, 5609361, 5272824, 5704218, 5836337, 5270050
347	5271788, 5271836, 5275848
348	5270619, 5702638, 5272195, 5275907, 5274549
359	5172114, 5503986
367	5174800, 5272182, 5173400, 5171211, 5825392
368	
415	5715079, 1234570, 5634131
424	5623547, 5272773, 5819534, 5835593, 5275841
426	5175868, 5613322, 5172567
428	5170745, 5173364
430	5937122, 5819524
437	5174826
440	5173513
444	5173010
450	5612845, 5172143, 5274076
483	5172898

Table 9. Computer–Generated Assignment

The course rating of an applicant listed in Chapter 2 follows the assignment policy of the assignment officer. However, the course priority may deviate from the manual assignment. The officer–assigned solution is given in Table 10.

Course Number	Applicants
210	1234568, 5272182, 5618797, 5807608, 5275907, 5819524, 5634131, 5507100, 5611388
222	5515810, 5270614, 5609361, 5272824, 5272863, 5704218,
347	5172114, 5271836, 5173400
348	5271788, 5702638, 5272195, 5274549, 5275848
359	5270619, 5171211, 5503986, 5275581
367	5174800, 5825392, 5704146, 5172567, 5173513
368	5274076, 1234574
415	5715079, 5836337, 1234573, 1234570
424	5819534, 5623547, 5272773, 5808215, 5275841
426	5835593, 5613322, 5175868
428	5078906, 5173364
430	5270050, 5937122
437	5174826
440	1234569, 5172143
444	5173010, 1234567
450	5612845, 5272773, 1234572
483	5172898, 1234571

Table 10. Actual Assignment

There were a total of 61 vacancies in the selected term used for testing the performance of the program. In the assignment generated by the program, 53 applicants were accommodated in the courses with either the request by the course instructors or themselves.

Seven of the applicants did not supply any information about their previous experience and course preference. Thus, assignments for them were left open by the program. For the remaining single applicant, 5611388, who had indicated preference for course 210, was filled by other applicants.

As to the effect of data order, consider course 210 appearing before course 222. Twenty–four of the assignments, which are underlined in Tables 9 and 10, match with the actual assignment made by the officer. On the other hand, if the order of courses 210 and 222 were reversed, twenty–six applicants in the assignment made by the program would become identical with the manual case. However, actual assignments allow many modifications. It seems that the missing information of instructor preference could account for the difference in both type of assignments. Moreover, the assignment officer may have deviated from the course priority formulated in Chapter 2. Also, course order in searching may explain part of the difference.

6.2 CHARACTERISTICS OF THE PROGRAM

In the five–applicant–and–five–vacancy example discussed in the chapter 5, there is one and only one set of applicant–course links that can match every vacancy with an applicant, and vice versa. However, the data used in for testing cannot achieve that. The matching of courses and applicants is limited by some factors; like: incomplete information about the applicants, not enough suitable applicants for a course and too many applicants apply for the same courses.

The established program operates as two distinct processes. The first process looks for a free applicant with the highest assignment–determining value in the database. The

second matching process refines the initial assignment obtained by the first matching process. The second process is aimed at a replaceable applicant suitable for a vacancy and assigns that applicant into the vacancy. By doing so, the program preserves the applicants who have the highest assignment–determining value, unless the applicants have already been assigned to a course of at least the same course priority level as the former counterpart.

An assigned applicant may be replaceable by another available applicant in the first matching process. However, after first matching process, the replaceable applicant may become not replaceable anymore. This is because the available applicant for substituting the formerly replaceable applicant may have been assigned to other courses with equal or even lower course priority level and is not replaceable. If lower, it would defy the priority policy of courses.

6.3 ALTERNATIVE ASSIGNMENT PROCESS

The presented program separates the assignment process into two Prolog procedures `first/1` and `second/1`. As an alternative, the program can be unified into single top goal. That is, whenever the database lacks a free applicant to fill a vacancy and `sub_first/2` fails, it looks directly for a replaceable applicant by calling `sub_second/2` in the `second/1` procedure. This method can preserve an applicant suitable for a course with equal or even higher course priority level. However, it may not preserve a higher assignment–determining value for the substitute.

For example, consider the case of two free applicants who are suitable for substituting a replaceable applicant with applicants having the same assignment–determining preference

value for that substitution. If they are also potential applicants for another course which is not yet considered by the program with different assignment–determining value, the modified program may assign one of them as the substitution for the replaceable applicant according to the order of the appearance of the applicants in the database. Hence, it may choose the one with a higher assignment–determining value in the other course for substitution. When the program later finds a suitable and free applicant for the such course, it has to choose the one with a lower assignment–determining value.

Chapter 7

CONCLUSION

The position assignment problem of TA's in the the Department of Electrical and Computer Engineering is posed annually and requires much attention on the part of the assignment officer. The problem involves the matching of more than fifty vacancies and applicants.

With one procedure call for each matching process in Prolog (as in Appendix E), the program searches for all the vacancies in the database by recursion and backtracking. It automatically chooses the most suitable and available applicant for each vacancy. The automation of this problem requires explicit priority policies. In particular, courses that have fewer suitable applicants must be given detailed consideration.

Availability of an applicant is solved by recursive definition rule in Prolog. Recursion handles the undoing of an assignment by successive shifts over assignment links together with the filling of the arising vacancies.

The program was tested by comparing the automated outcome with the actual manual assignment made by the assignment officer in a particular term. From a total of 62 applicants and 62 vacancies in that term, 23 applicants share the same assignment between the two methods of assignment. However, some information was not always available, such as

experience, instructor preference and applicant requests. This may account for some disagreement in the results.

APPENDIX

APPENDIX A

UTILITY PREDICATES

The suitability of an applicant for a course is determined by several factors: the request of the course instructor, and the applicant's preference and experience. Determination of the suitability is done by `suitable_ta(COURSE,STUDENT,CATEGORY)`, where `COURSE`, `STUDENT` and `Category` represent the course number, student number and the category how the applicant is chosen, respectively. The program will first choose the applicants who are requested by the course instructor from the list of students, `STUDENTS`, in the database given by `prof_request/2`. According to the preference of the applicant and his experience in that course, he would be assigned to that course with the highest ranking. They would be followed by the applicants who are preferred by the course instructor and have preference but have no experience in that course:

```
suitable_ta(COURSE,STUDENT,CATEGORY):-
prof_request(COURSE,STUDENTS),
suitable_ta1(COURSE,STUDENT,CATEGORY1),
CATEGORY is CATEGORY1 + 4,
member(STUDENT,STUDENTS).
```

Any other applicant who is listed on the course instructor's request will then be chosen:

```
suitable_ta(COURSE,STUDENT,7):-
prof_request(COURSE,STUDENTS),
member(STUDENT,STUDENTS).
```

The following categories are the applicants with preference in that course. Again, the applicants with experience will be chosen first:

```
suitable_ta(COURSE,STUDENT,CATEGORY):-
suitable_ta1(COURSE,STUDENT,CATEGORY).
```

The subroutine `suitable_ta1(COURSE,STUDENT,CATEGORY)` is able to find out the applicants with preference in COURSE, those who have experience in that course will be selected first:

```
suitable_ta1(COURSE,STUDENT,6):-
ape(COURSE,STUDENT).


suitable_ta1(COURSE,STUDENT,4):-
course(COURSE,PRIORITY,SUB_PRIORITY),
apn(COURSE,SUB_PRIORITY,STUDENT).
```

The following predicates determine the assignment determining value of the applicants.

```
ape(COURSE,STUDENT):-
student_info(STUDENT,PREFERENCE,EXPERIENCE,DEGREE),
member(COURSE,EXPERIENCE),
member(COURSE,PREFERENCE).


apn(COURSE,PRIORITY,STUDENT):-
student_info(STUDENT,PREFERENCE,EXPERIENCE,DEGREE),
member(COURSE,PREFERENCE),
not_violate(DEGREE,PRIORITY).
```

`not_violate/2` prevents a inexperienced graduate being assigned to course for the other department:

```
not_violate(DEGREE,3):-
!.


not_violate(3,DEGREE):-
!.
```

```
not_violate(DEGREE,DEGREE):-
!.
```

check_unit(Course) is true if the the not yet assigned units, given by Unit in unit_left/2, of COURSE is greater than one:

```
check_unit(COURSE):-
unit_left(COURSE,UNIT),
UNIT > 1.
```

assigned_TA(COURSE,COURSE1,STUDENT,CATEGORY,COURSE_TRIED, TA_TRIED) finds an assigned applicant for a still empty position starts with the students who are assigned to any course with a low priority level of assignment. The subroutine will find out the current assignment of the found applicant. Thus, the program knows which course needs a substitute:

```
assigned_TA(COURSE,COURSE1,STUDENT,CATEGORY,COURSE_TRIED,TA
_TRIED):-
low_priority_assign(STUDENT,COURSE1,4),
not(member(STUDENT,TA_TRIED)),
not(member(COURSE1,COURSE_TRIED)),
course(COURSE1,PRIORITY,SUB_PRIORITY),
suitable_TA(COURSE,STUDENT,CATEGORY).
```

The value of LOWEST, the category of assignment, starts with one, and it stops when an applicant is found for the course or the value exceeds ten:

```
low_priority_assign(STUDENT,COURSE,LOWEST):-
in(STUDENT,COURSE),
assign(STUDENT,COURSE,LOWEST,_).


low_priority_assign(STUDENT,COURSE,LOWEST):-
LOWEST<11,
```

```
LOWEST1 IS LOWEST + 1,
low_priority_assign(STUDENT,COURSE,LOWEST1).
```

APPENDIX B

PROGRAM LISTING

```
/* first(Priority) invokes the primary matching process
   until 'Priority'=4. With 'Priority'=<3, it keeps on
   searching the database for available TA who are good for
   the course in that 'Priority'. */

first(Priority):-
Priority =< 3,
sub_first(Priority,1),
Priority1 is Priority + 1,
first(Priority1).

first(Priority):-
tell('ans1.1'),
write('The actions from the first trial'),
nl,
write('--------------------------------'),
nl,
nl,
listing(assign),
nl,
nl,
write('The assignment of the first trial'),
nl,
write('--------------------------------'),
nl,
solution,
told,
!.

sub_first(Priority,Sub_priority):-
course(Course,Priority,Sub_priority),
check_unit(Course),
suitable_TA(Course,Student,Category),
not_assigned(Student),
good(Course,Student,Category,assign),
final_event(Event),
retract(final_event(Event)),
Event1 is Event + 1 ,
assert(final_event(Event1)),
fail.
```

```
sub_first(Priority,Sub_priority):-
Sub_priority =< 3,
Sub_priority1 is Sub_priority + 1,
sub_first(Priority,Sub_priority1).

sub_first(Priority,Sub_priority):-
!.


/* suitable_TA(Course,Student,Category) is true if there is
   an available TA for 'Course' under one of the 'Category'.
   */


suitable_TA(Course,Student,Category):-
prof_request(Course,Students),
suitable_TA1(Course,Student,Category1),
Category is Category1 + 4,
member(Student,Students).

suitable_TA(Course,Student,7):-
prof_request(Course,Students),
member(Student,Students).

suitable_TA(Course,Student,Category):-
suitable_TA1(Course,Student,Category).

suitable_TA1(Course,Student,6):-
ape(Course,Student).

suitable_TA1(Course,Student,4):-
course(Course,Priority,Sub_priority),
apn(Course,Sub_priority,Student).


/* check_unit(Course) is true if the unit_left in that
   course is greater than one. */


check_unit(Course):-
unit_left(Course,Unit),
Unit > 1.
```

```
/* ape(Course,Student) is true if the student is experienced
   and has preference in teaching that course. */

ape(Course,Student):-
student_info(Student,Preference,Experience,Degree),
member(Course,Experience),
member(Course,Preference).


/* apn(Course,Priority,Student) is true if the student
   indicates preference but has no experience. */

apn(Course,Priority,Student):-
student_info(Student,Preference,Experience,Degree),
member(Course,Preference),
not_violate(Degree,Priority).


/* not_violate(Degree,Priority) is true if the academic
   degree of the student is compatible with the course. */

not_violate(Degree,3):-
!.

not_violate(3,Degree):-
!.

not_violate(Degree,Degree):-
!.


/* not_assigned(Student) is true if the student is not
   assigned. */

not_assigned(Student):-
in(Student,Course),
!,
fail.

not_assigned(Student):-
student_info(Student,_,_,_),
!.
```

```
/* good(Course,Student,Category) is true if there are units
   left for that course. */

good(Course,Student,Category,Action):-
check_unit(Course),
retract(unit_left(Course,Unit)),
New_unit is Unit - 2,
asserta(unit_left(Course,New_unit)),
put_in_back(student_info(Student,_,_,_)),
insert_assign(Course,Student,Category,Action),
!.

put_in_back(Data):-
retract(Data),
assertz(Data).


/* insert_assign(Course,Student,Category,Event,Action)   into
   the database. */

insert_assign(Course,Student,Category,move):-
assign(Student,Any_course,Any_category,Any_event),
final_event(Event),
assertz(move(Student,Any_course,Course,Category,Event)).

insert_assign(Course,Student,Category,assign):-
not_assigned(Student),
final_event(Event),
assertz(assign(Student,Course,Category,Event)).

member(X,[]):-
!,
fail.

member(X,[X|Y]):-
!.

member(X,[H|T]):-
X \== H,
member(X,T).
```

```
append([],L,L):-
!.

append([H|T],L,[H|L1]):-
append(T,L,L1).


/* delete(E,L,L1) is true if L1 contains the same elements
   in L except E. */

delete(E,[],_):-
!,
fail.

delete(E,[E|T],T):-
!.

delete(E,[H|T],[H|T1]):-
delete(E,T,T1),
!.


/* ta_list(Course,List) is true if List is the list of the
   TAs for Course. */

ta_list(Course,List):-
not(var(List)),
ta_list(Course,List1),
compare_list(List,List1),
!.

ta_list(Course,List):-
ta_list(Course,List,[]),
!.

ta_list(Course,List,Acc):-
in(Student,Course),
not(member(Student,Acc)),
!,
ta_list(Course,List,[Student|Acc]).
```

```
ta_list(Course,List,List):-
!.


/* in(Student,Course) is true if Student is in the list of
   TAs in Course. */

in(Student,Course):-
assign(Student,Course,_,Event),
not_moved(Student,Course,Event).

in(Student,Course):-
move(Student,_,Course,_,Event),
not_moved(Student,Course,Event).


/* not_move(Student,Course,Event) is true if Student is not
   moved to any other course from Course after event Event.
   */

not_moved(Student,Course,Event):-
move(Student,Course,_,_,Event1),
Event1 > Event,
!,
fail.

not_moved(Student,Course,Event):-
!.


/* compare_list(List,List1) is true if List and List1 have
   the same elements. */

compare_list(List,List):-
!.

compare_list(List,[H|T]):-
member(H,List),
delete(H,List,List1),
compare_list(List1,T).
```

```
/* second(Priority)  invokes  a  secondary  refinement  until
   'Priority'=4. */

second(Priority):-
Priority =< 3,
sub_second(Priority,1),
Priority1 is Priority + 1,
second(Priority1),
!.

second(Priority):-
tell('ans2.1'),
write('The action after the second trial:'),
nl,
write('-----------------------------------'),
nl,
nl,
listing(assign),
listing(move),
nl,
nl,
write('The assignment after the second trial'),
nl,
write('-------------------------------------'),
nl,
solution,
told,
!.

sub_second(Priority,Sub_priority):-
course(Course,Priority,Sub_priority),
check_unit(Course),
second_choice_TA(Course,Student),
final_event(Event),
retract(final_event(Event)),
Event1 is Event + 1 ,
assert(final_event(Event1)),
fail.

sub_second(Priority,Sub_priority):-
Sub_priority =< 3 ,
```

```
Sub_priority1 is Sub_priority + 1,
sub_second(Priority,Sub_priority1).

sub_second(Priority,Sub_priority):-
!.


/* second_choice_TA(Course,Student) is true if there is
   another choice of TA for that course. */

second_choice_TA(Course,Student):-
Course_tried = [Course],
try_not_free_TA(Course,Course1,Student,Category,Course_tried
,[]),
Ta_tried = [Student],
replace_not_free_TA(Course1,Course_tried,Ta_tried),
good(Course,Student,Category,move).


/* try_not_free_TA(Course,Course1,Student,Category,Course_tr
   ied,Ta_tried) is true if there is TA in Course1, not in
   Course_tried, is also a potential candidate for Course,
   but not in Ta_tried.

try_not_free_TA(Course,Course1,Student,Category,Course_tried
,Ta_tried):-
low_priority_assign(Student,Course1,4),
not(member(Student,Ta_tried)),
not(member(Course1,Course_tried)),
course(Course1,Priority,Sub_priority),
suitable_TA(Course,Student,Category).

low_priority_assign(Student,Course,Lowest):-
Lowest<11,
in(Student,Course),
assign(Student,Course,Lowest,_).

low_priority_assign(Student,Course,Lowest):-
Lowest<11,
Lowest1 is Lowest + 1,
low_priority_assign(Student,Course,Lowest1).
```

```
/* replace_not_free_TA(Course,Course_tried,Ta_tried) is true
   if there is a free potential candidate who is not
   assigned and he can also be a TA of that course. */


replace_not_free_TA(Course,Course_tried,_TA_tried):-
course(Course,Priority,Sub_priority),
student_info(Student1,_,_,_),
not_assigned(Student1),
suitable_TA(Course,Student1,Category),
change_unit_left(Course,2),
good(Course,Student1,Category,assign),
!.

replace_not_free_TA(Course,Course_tried,Ta_tried):-
append([Course],Course_tried,Course_tried1),
try_not_free_TA(Course,Course1,Student,Category,Course_tried
1,Ta_tried),
append([Student1],Ta_tried,Ta_tried1),
replace_not_free_TA(Course1,Course_tried1,Ta_tried1),
change_unit_left(Course,2),
good(Course,Student1,Category,move).


/* change_unit_left(Course,Unit_change) is true if the unit
   left in that course is changed by Units_change. */


change_unit_left(Course,Units_change):-
retract(unit_left(Course,Unit)),
New_unit is Unit + Units_change,
asserta(unit_left(Course,New_unit)).

solution:-
in(Student,Course),
write('assign '),
write(Student),
write(' to course '),
write(Course),
nl,
fail.
```

```
solution:-
!.
```

APPENDIX C

ASSUMED COURSE PRIORITY USED IN CHAPTER 6

/* course(COURSE,PRIORITY,SUB_PRIORITY) is the information of COURSE , which is the course number of a course; PRIORITY is the priority level which the course number belongs to, and; SUB_PRIORITY is the sub–priority which the course belongs to in the corresponding priority level. */

```
course(415,1,3).
course(426,1,3).
course(420,1,2).
course(424,1,3).
course(428,1,2).
course(430,1,2).
course(437,1,2).
course(440,1,2).
course(483,1,3).
course(450,1,3).
course(348,2,2).
course(359,2,2).
course(367,2,3).
course(210,3,3).
course(222,3,3).
course(414,1,2).
course(416,1,2).
course(419,1,2).
course(422,1,3).
course(423,1,3).
course(425,1,3).
course(427,1,2).
course(429,1,2).
course(431,1,2).
course(439,1,2).
course(442,1,3).
course(458,1,2).
course(353,2,2).
course(358,2,3).
course(360,2,3).
course(361,2,3).
course(365,2,2).
course(369,2,3).
course(213,3,2).
```

```
course(216,3,3).
course(221,3,2).
course(224,3,2).
course(444,1,1).
course(445,1,1).
course(446,1,1).
course(447,1,1).
course(451,1,1).
course(366,2,1).
course(347,2,2).
course(225,1,2).
course(368,2,3).
```

APPENDIX D

THE INFORMATION USED IN CHAPTER 6

/* **unit_left(COURSE,UNITS)** represents UNITS of works is not assigned for COURSE. The sentence might been changed whenever a candidate is assigned to COURSE. */

```
unit_left(210,18).
unit_left(222,14).
unit_left(347,6).
unit_left(348,10).
unit_left(359,10).
unit_left(367,10).
unit_left(368,4).
unit_left(415,8).
unit_left(424,10).
unit_left(426,6).
unit_left(428,4).
unit_left(430,4).
unit_left(437,2).
unit_left(440,4).
unit_left(444,4).
unit_left(445,2).
unit_left(450,6).
unit_left(483,4).
```

/* **student_info(STUDENT,PREFERENCE,EXPERIENCE,DEGREE)** represents the information of **STUDENT**, **PREFERENCE** is a list of the preferred course chosen by **STUDENT**, **EXPERIENCE** is the list of courses that **STUDENT** has experience in, and **DEGREE** is the degree that **STUDENT** is holding. */

```
student_info(5270050,[430,222,431,442],[222,353],2).
student_info(5174800,[367,210,216,221],[216,367],2).
student_info(5819534,[424,222,423,361],[222,358],2).
student_info(5515810,[],[360],2).
student_info(5270614,[222,210,216,220],[222,216],2).
student_info(5270619,[359,348,213,224],[213],2).
student_info(5835593,[222,424,442,422],[],2).
student_info(1234567,[],[],1).
student_info(5170745,[428,359,224,213],[213,428],2).
student_info(5612845,[450,222],[450,423],2).
student_info(5171211,[367,210,216,101],[221,359],2).
```

```
student_info(5715079,[415,416],[415,216],2).
student_info(5503986,[],[213,359],2).
student_info(1234568,[],[],2).
student_info(5172114,[210,359,224],[224,359],2).
student_info(5172143,[450,483,360,419],[360,348],2).
student_info(5271788,[347,348,221,414],[348,221],2).
student_info(5271836,[210,347,224,427],[],2).
student_info(5825392,[210,367,353,216],[],2).
student_info(5702638,[437,348,365,442],[365],2).
student_info(5272182,[367,210,360,221],[367,360],2).
student_info(5609361,[222,361,361,423],[361,748],1).
student_info(5937122,[210,430,222,221,431,216],[],2).
student_info(5704146,[210,422],[210,422],2).
student_info(5618797,[210],[210,429],2).
student_info(5807608,[210,367,221,365],[],2).
student_info(5623547,[222,424,361,221],[424,361],3).
student_info(5613322,[],[426],2).
student_info(5272824,[222,367,216,353],[],2).
student_info(5172567,[367,426,358,365],[367,365],2).
student_info(5173010,[444,222,361,400],[444,361],2).
student_info(5078906,[210,428,216,358],[210,225],2).
student_info(5272773,[],[423,424],2).
student_info(5172898,[483,360],[360,483],2).
student_info(5272863,[210,222,361,221],[],2).
student_info(5173364,[222,428,224,221],[428,213],2).
student_info(5808215,[222,210,221,361],[],2).
student_info(5275907,[210,348,213,353],[],2).
student_info(1234569,[],[],2).
student_info(5173400,[367,221,360],[367,347],2).
student_info(5819524,[210,430,216,221],[],2).
student_info(5173513,[367,442,440],[367,442],2).
student_info(5175868,[426,444,358,361],[426,358],2).
student_info(5275920,[210,222,221,216],[210,225],2).
student_info(5174826,[437,348,365,216],[437,365],2).
student_info(5275581,[210,221],[210,213],2).
student_info(5704218,[210,222,216,221],[],2).
student_info(5275841,[424,450,361],[],2).
student_info(5836337,[210,222,353,221,216],[],2).
student_info(5275848,[347,348,221,431],[348],2).
student_info(1234570,[210,415,221,225],[],2).
```

```
student_info(5274076,[450],[424,366],1).
student_info(5634131,[210,415,429,416],[429],2).
student_info(5507100,[220,228,358,353,442,221],[210,353],2).
student_info(5611388,[210,439],[439],2).
student_info(1234571,[222,358],[222,358],2).
student_info(1234572,[],[],2).
student_info(5274549,[210,348,437,216,431],[210,365],2).
student_info(5272195,[437,348,365,442],[],2).
student_info(1234573,[],[],2).student_info(1234574,[],[],2).
```

/* **final_event(EVENT)**, **EVENT** represents the numbers of events which has been generated by the program. */

```
final_event(0).
```

/* **prof_assign(COURSE,LIST)**:– **LIST** represents the list of students who are requested by the instructor of **COURSE**. */

Course Number	Students			
437	5274549			
483	5172898	5172143		
424	5623547	5272773	5819534	5835593
	5275841			
426	5175868	5613322	5172567	
415	5715079	1234570	5634131	
440	5173513			
430	5270050	5937122		
428	5170745	5173364		
444	5173010			
445				
450	5612845	5274076		
210	5704146	5618797	5078906	5275920
	5275581	5507100	5270614	5807608
	5819524			
222	1234571	5609361	5272824	5272863
	5808215	5704218	5836337	
347	5271836	5271788		
348	5275848	5702638	5275907	5272195
	5174826			
359	5172114	5503986	5270619	
367	5174800	5272182	5173400	5171211
	5825392			
368				

APPENDIX E

SAMPLE GOAL CALL

The program, as listed in appendix B, is stored in file 'test905', information of course, as listed in appendix C, is stored in 'course.info', and general information used, as listed in appendix D, is stored in 'info.fall'. The result of the computation will be listed in 'ans1' and 'ans2'.

```
C-Prolog version 1.5+
| ?- [test905,'course.info','info.fall'].
test905 consulted 7708 bytes 0.15 sec.
course.info consulted 1840 bytes 0.0833334 sec.
info.fall consulted 6472 bytes 0.166667 sec.

yes
| ?- first(1).

yes
| ?- second(1).

yes
| ?- exit.

no.
| ?-
[ Prolog execution halted ]
```

BIBLIOGRAPHY

[1] Genersereth, M. R. and Nilsson N. J., "Logical Foundation of Artificial Intelligence", Los Altos, California: Morgan Kaufumann, 1987.

[2] Nilsson, N. J., "Principles of Artificial Intelligence", Palo Alto, California: Tioga Publishing Co., 1981.

[3] Rowe, Neil C., "Artificial Intelligence Through Prolog", Englewood Cliffs, N.J. : Prentice–Hall, 1988.

[4] Sterling, Leon and Shapiro, Ehud, "The Art of Prolog: Advanced Programming Techniques", Cambridge, Massachusetts: The MIT Press, 1986.