A KNOWLEDGE BASED APPROACH TO PLANNING IN CARD GAMES

A Thesis

Presented to

The Faculty of the Department of Computer Science

University of Houston - University Park

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Pai-Yen Lo

December, 1987

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my advisor Dr. Christoph Eick for his help, guidance and encouragement during the research for this thesis. I especially appreciate his confidence in me and critical review of this thesis.

I am also grateful to my thesis committee members, Drs. Marek Rusinkiewicz and Fajtowicz Siemion for their criticisms and suggestions.

Finally, I thank to Mrs. Regina Townsend for her help to correct the grammar errors on the documentation of my thesis.

A KNOWLEDGE BASED APPROACH TO PLANNING IN CARD GAMES

An Abstract of a Thesis

Presented to

The Faculty of the Department of Computer Science

University of Houston - University Park

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Pai-Yen Lo

December, 1987

ABSTRACT

The purpose of this research is to explore computational approaches to the planning in card games. Because of the complexity of the problem, the study was restricted to the Declarer playing in a NT-contract in Bridge. A program has been provided which simulates the planning process of a human player. It uses a knowledge based approach which encodes knowledge in form of rules and in form of plans. The program uses rules to select playing techniques for each suit held in the player's hand during static analysis and to guide a small tree search which confirms a particular technique is best. Once a technique is selected, the plans are used to construct the playing sequence of cards for this technique. The possibilities and limitations of the overall approach are discussed.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Topic of the thesis :

One of the central concerns of artificial intelligence is expressing knowledge and reasoning with it. Game playing has been the first popular domain for AI study. There are many popular games such as Chess, Bridge, Tic-tac-toe, Go, Checkers, and Backgammon. For a human player playing in one of these games, the general process is to analyze an entire sequence of steps in advance to discover where it will lead before the first step is actually taken. We refer to this process as a planning process. The planning process is very straightforward in such a game that the result of a move is predictable.

However, in card game such as Bridge, the planning process is a little bit complicated since we do not know exactly where all the cards are located or what the opponents will do on their turns. This means that it is impossible to plan an entire sequence of moves and be confident that we know what the resulting state will be. Therefore, in planning these kinds of games, what we would like to do is to investigate several plans and choose a plan which can make the best of the current situation and go on from there.

In playing Chess, a player plays alone against his opponent; but in playing Bridge, a player needs to cooperate with his partner as a team. Furthermore, he can be either a defender or an offender during the game.

There are some programs written to play Bridge card-games. Among them, only three programs were found to be developed by taking AI techniques. One is the Bridge bidding program [1]. The other one is the program which locates a missing card [2]. And another is a program named BRIPP [10] which is a Bridge-playing program. Although the details about about BRIPP [10] couldn't be found, Stanier in his paper [1] criticizes it as being incapable to play a good game. However, he doesn't mention the problems with BRIPP [10]. For a computer program to play a good game, a better approach for representing knowledge and a searching technique guides a small tree search to find the best move must be exploited. The search is small in the sense that the size of the search tree is of the same order of magnitude as a human master's search tree (ten and hundreds of nodes). The intentions of this study are therefore to present an overall approach to express knowledge and to exploit a searching technique to find the best move. As a result of this study, a Bridge-playing program was built.

1.2 Planning problem in Bridge card games :

The process of problem solving is a search through a state space in which each point corresponds to a situation that might arise. Many

problems addressed by AI techniques involve search through a large space of possible solutions. For the complicated problem domains such as game playing, it becomes important to be able to work on small pieces of a problem separately and then to combine the partial solutions at the end into a complete problem solution. Planning is the action of decomposing the original problem into appropriate subparts and recording and handling interactions among the subparts as they are detected during the problem-solving process.

Planning is familiar to all of us. We need to get something done, so we make a list of all the steps involved and check them off as we accomplish them. Say we want to vacation in Tahiti. You need to call a travel agent, budget some money, buy a new swimsuit, and so on. Each step in the plan involves a subplan. To buy a swimsuit, you need to get in your car, drive down to a department store, find the sporting goods section, pick a swimsuit, and pay for it at the cash register.

Planning is not just a matter of making a list of all the steps involved in order to get something done. A very important issue of planning is to find the sequence of steps which is the best among those possible sequences. A sequence of steps is the best if the job can be done correctly and efficiently by taking it. Correctness and efficiency are very important considerations for doing planning.

Humans make plans against their opponents in games. For example, in playing Chess, a player decides what steps need to be taken for the par-

ticular goal and decides on what to do in the event of various opponent replies. The planning here is somehow different from the one discussed in the previous paragraph.

In playing a Bridge card-game, what we would like to do is to plan the entire hand before making the first play. But it is impossible to do such planning with certainty, since the knowledge collected so far would not enable us to describe the situation absolutely. The plans generated in this particular world must more or less rely on probability. By investigating these plans and by assigning probabilities of the various outcomes, the plan which has the highest expected probability of leading to a success has to be chosen.

Playing a game, the offense wants to have a reply ready for every defensive alternative. A plan cannot therefore be a linear sequence of goals or moves, but must contain conditional branches depending on the opponent's actions. When the offense is on his move, a specific move or goal provided by the plan. When the defense is on his move, a list of alternative sub-plans for the offense may be given. For example, your partner leads a card, if your right-opponent has a card left in the suit led, then he can play either a card whose rank is higher than the card led or a card whose rank is lower than the card led. If the suit led is a void suit in your right-hand opponent's hand, then he must discard a card from one of side suits in his hand. To respond to the right-opponent's play, the plan suggests the possible offensive move for the third-hand

player. In the later chapters, we will discuss this in more detail.

The discussions in the rest of this chapter and in the following chapters will tackle with planning the play of a Bridge game under the following motivations :

a. to develop a program which is capable of planning under uncertainty.

b. to provide knowledge representation framework for planning in uncertainty that are suitable to make correct decision quickly.

1.3 The thought process of a human player in a Bridge card game :

Suppose you are the declarer in a contract of 3NT. West, your left-hand opponent, leads the queen of diamonds and your partner puts his hand down as dummy :

```
              (dummy)   Spade    7  5  2
                        Heart    8  3
                        Diamond  K  9  2
                        Club     A  K  6  4  3
(West)
  Opening lead - Diamond Q

                    (declarer)
                            Spade    A  K  4  3
                            Heart    A  Q  4
                            Diamond  A  7  3
                            Club     8  5  2
```

How do you, as declarer, go about playing the hand to make the contract? First, you must make a plan. In order to construct a plan for

this hand, you need to examine each suit to see what kind of contributions it can make for the contract.

A. The first step -- setting up your goal

At very beginning, you need to set up your goal. In other words, you need to make your guess at how many tricks you intend to make.

In this example, your goal is to make at least 9 tricks, which is calculated by adding 6 to the level of the contract.

B. The second step -- counting tricks

Next, you need to count how many sure tricks you have in your hand and your partner's hand.

The number of sure tricks on hand is :

Spades	:	2	- the Ace and the King
Hearts	:	1	- the Ace
Diamonds	:	2	- the Ace and the King
Clubs	:	2	- the Ace and the King

The total number of sure tricks is 7. Therefore, at least 2 extra tricks must be built.

C. The third step -- suggesting the possible techniques for each suit.

Spades : By playing the Ace and King and then giving the opponents a trick, the fourth Spade will become

established as a trick if the opponents' Spades are divided three and three.

Hearts : By leading from dummy and finessing the Queen you may get an extra trick by trapping your right-hand opponent's King.

Diamonds : There is no way to build any extra tricks.

Clubs : By playing 3 runs of Clubs, 2 extra tricks will be built if the outstanding Clubs are divided three and two.

In books [3] and [4], there are several techniques presented for No-Trump contract. Among them, the following four techniques are the most common techniques used to build extra tricks.

Technique 1 - By promoting card
Technique 2 - By suit length
Technique 3 - By finesse (trapping a missing high card)
Technique 4 - By throw-in (forcing opponent to play first)

In his example, Both the Spade suit and Club suit can use the Technique 2. The Heart suit can use Technique 3 or Technique 4.

D. The fourth step -- making up an executable plan

To formulate an executable plan, the guidelines described in book [3] are used. These guidelines are used very often in planning the game for No-Trump contract.

Guidelines

a. Build extra tricks you need to make contract before taking your sure tricks.

b. Find the suit which contributes more extra tricks and play it first.

c. Watch your entries to be sure that you can get to the hand from which you want to lead to the next trick.

For each suit, we have conveyed the possibility of building extra tricks and the possible techniques are suggested. Among these techniques, we then evaluate each technique. Eventually, we come up with a feasible plan.

a. Two extra tricks must be built --

Both the Heart suit and the Spade suit can only provide one extra trick. You would have to be successful in building an extra trick in both cases in order to get the two tricks needed. The Club suit seems to offer the best possibility for providing both extra tricks.

b. Considerations about entry setup -

If you win the first Diamond trick with dummy's King, after you have built the 2 extra Club tricks, you will have no way to get to dummy to lead them. Therefore, you win the first trick with your Ace. You will then be able to use dummy's King as an ENTRY to the dummy to take your two Club tricks.

In summary, the thought process of a human player in playing a bridge game is as follows : First, he counts how many tricks needed to make the contract. Second, he checks each suit in hand to find the number of sure tricks, the number of extra tricks, and the possible techniques for building these extra tricks. Then he examines these techniques to select the best one. A technique is found for each suit which is capable of building extra tricks. The tech-

nique which can build most of the extra tricks is selected to be played first. So he starts at this technique and tries to construct a plan. If it turns out to be impossible to build a plan for this technique, then he tries the next best technique if there is one available. Eventually, a plan is constructed and executed. After he has built a sufficient number of extra tricks, he then plays all the sure tricks.

Our overall approach is an approach which simulates each step described above. The detailed explanations of an overall approach is given in the next chapter.

1.4 Explanations on how to play Bridge :

The text in this section explains how to play a Bridge card-game. Some of the words or phrases are circled with a pair of double-quotes. These words (or phrases) have their own meanings in the Bridge card-game and will be mentioned somewhere in the later chapters. The explanation here is very brief. There are many books available describing the terms used by Bridge players. The following explanation is extracted from one of the books [3]. This book also includes the glossary of terms used in the Bridge card-game.

Bridge is a partnership game. The 4 players split themselves into 2 partnerships. The cards are dealt clockwise and face-down. After each

player has sorted his hand into suits, the dealer has the first chance to bid or pass and then, clockwise, each player in turn.

The objective is to win as many tricks as possible. A "trick" consists of 4 cards, one contributed by each player. The cards are played one at a time moving clockwise around the table. The play to each trick follows some rules :

* One of the players "leads" to the trick by placing any card he wishes face-up one the table.
* The other three players play a card, one at a time, in clockwise rotation.
* Players "follow" suit to the card led by playing a card in the same suit where possible.
* If a player cannot follow suit, he plays any card from a "side" suit. This is called "discarding".
* The trick is won by the highest card played in the suit that was led. The player winning the trick leads to the next trick.

A Bridge hand can be played either in "No Trump" or with a "trump" suit. In No-Trump, the highest card played in the suit led wins the trick. In a trump suit, one suit is trump. If a player can't follow suit, a trump can be played. The is called "trumping" or "ruffing" the trick.

Before the play can start, a "contract" must be decided through a process called "bidding". Bridge bidding is like an auction. The first player to open the bidding during this process is called the "opening

bidder" or "opener". A player makes a "bid" by naming a "level" and a "denomination". For example, level of the bid "one Spade" is 1 and the denomination of the bid is "Spades". The bidding usually starts at the one-level. The first 6 tricks are taken for granted. These 6 tricks are called the "book". The one-level, then, is 6+1=7 tricks. If a player bids 3 No Trump (or NT) and this is followed by Pass, Pass, Pass. The contract is 3NT, a commitment to take 6 + 3 = 9 tricks, with no suit as the trump suit. The first step in bidding is to value the hand. There are two factors which determine the trick-taking potential of a hand:

* High cards (Aces, Kings, Queens, Jacks)
* Long suits (A suit consisting of the Ace, King, Queen, seven, Six, and Three, for example, will often take 5 or 6 tricks)

Hand valuation points are given for both high cards and for long suits.

High Card Points	:	Ace - 4 points, King - 3 points, Queen - 2 points, Jack - 1 point
Length points	:	5-card suit - 1 point 6-card suit - 2 points 7-card suit - 3 points 8-card suit - 4 points

The high card points are added to the length points to determine the total value or point count of the hand. For example :

		High card point	Length points
Spade	A 7 3	4	0
Heart	K 4	3	0
Diamond	J 9 8 6 3	1	1
Club	Q 7 3	2	0
		10	1 = 11 points.

After the bidding is complete, the final contract has been determined. There will be two teams. The "offense" will be the side that make the highest bid. They will make their contract if they win at least the number of tricks contracted for. The offensive player who first mentioned the denomination of the final contract becomes the "declarer". the other member of the offensive team is the "dummy". The player on the Declarer's left makes an "opening lead", and the dummy puts his hand face up on the table. Declarer plays both hands for the offense and tries to make enough tricks to make his contract. The "defense" works together to try and take enough tricks to defeat the contract.

We have given the basic ideas about playing a Bridge card-game. From that, we know the objective during the play is to take tricks. There are two types of tricks :

* "sure tricks" which you can take without giving up the lead to your opponents.

* "building tricks" or "extra tricks" which can be developed by adopting one of the following techniques :

- prompting cards
- establishing long suits
- finessing ... trapping opponents' high cards
- throw-in ... forcing opponent play before you

The detailed description of these techniques is in the book [3].

Details of how to apply these techniques will not be discussed here. However, in later chapters it will be discussed by giving the examples.

Before we leave this section, there is one important term we need to explain. That is "entries". When you, as a declarer, have a choice of winning a trick in dummy or in your own hand, there is sometimes an advantage to win the trick in a particular hand. You may want to lead a suit starting from the dummy or you may want to have the lead in your own hand. Used in this way, sure tricks can represent "entries" from one hand to the other. The value of "entries" will be seen in a later example.

1.5 Constraints on the Problem Domain :

As we have mentioned before, playing a bridge game is a complicate domain. It may take years to develop a complete computer system to play Bridge. This kind of system can handle bidding, drawing deductions on cards played by opponents, and making plans for the defenders and offenders. In our study, we intend to build a Bridge planner which is not complete, but has all the basic components. We wish that this prototype can be a building block for making a perfect Bridge planner in the future. To reach this goal, we will limit our study to the problem of planning in

No-Trump contract. The techniques currently used for making this planner are the Finesse technique, the Promote technique, the Throw-in technique, and the Win-tricks-by-long-suit technique. There are two teams in a game, but the planner built is only used for making offensive play. For the defensive play, no planning is performed. To make our study more dedicated to the planning itself, the planner built does not have to have the ability to produce answers for the questions about cards held by opponents. For each card played by the opponents, no deduction is drawn by the planner.

1.6 Organization of Thesis :

In Chapter 2, we give an overview of our computational approach to the planning problem of playing a Bridge game. The approach divides the human planning process into three phases. One phase is the phase of doing suit analysis. The other phase is the phase of selecting the best playing technique. Another phase is the phase of constructing a card sequence for a playing technique.

In Chapter 3, we present an overall approach of knowledge representation. The schemes used for different data structures such as rules and frames are explained.

In Chapter 4, the trick counting technique used in the suit-analysis phase and the best-technique phase is introduced. The technique for evaluating threats coming from opponents on a card led and the technique

for discarding a card are also explained.

In Chapter 5, the Plan Language is defined and an example is given to show how the statements defined in Plan Language are used.

Chapter 6 and 7 have detailed explanations of the overall approach described in Chapter 2. In Chapter 6, we show how to do suit analysis and how a technique is proposed. The selecting scheme of a best technique is also presented in this chapter. In Chapter 7, we try to show how the Plan language is used in constructing plans which are used to generate a card sequence for a selected playing technique.

In Chapter 8, the control mechanisms of the Bridge planner are discussed.

In Chapter 9, we explain how a program is constructed to implement the planning.

In Chapter 10, we give an example to demonstrate how the planning works.

Finally, in Chapter 11, the conclusions are drawn. The possibilities and limitations of our overall approach are discussed.

CHAPTER 2

A COMPUTATIONAL APPROACH TO PLANNING THE PLAY IN BRIDGE

In Chapter 1, we discussed the thought process of a good player in playing a No-Trump contract. In this chapter, we are about to discuss a computational approach to simulate this thought process.

The computer program playing games is implemented by dividing the whole period into two stages :

Stage 1 - The Declarer side is in the defensive position and he and his partner play against the opponent's opening lead. This stage is ended when the Declarer side gains the lead.

Stage 2 - The Declarer side plays the game by rotating the playing side in either the offensive or defensive position until the game is over. Three situations exist during this stage :

* Declarer side is in offensive position

  -- planning is performed to decide the offensive moves.

* Declarer side is in defensive position

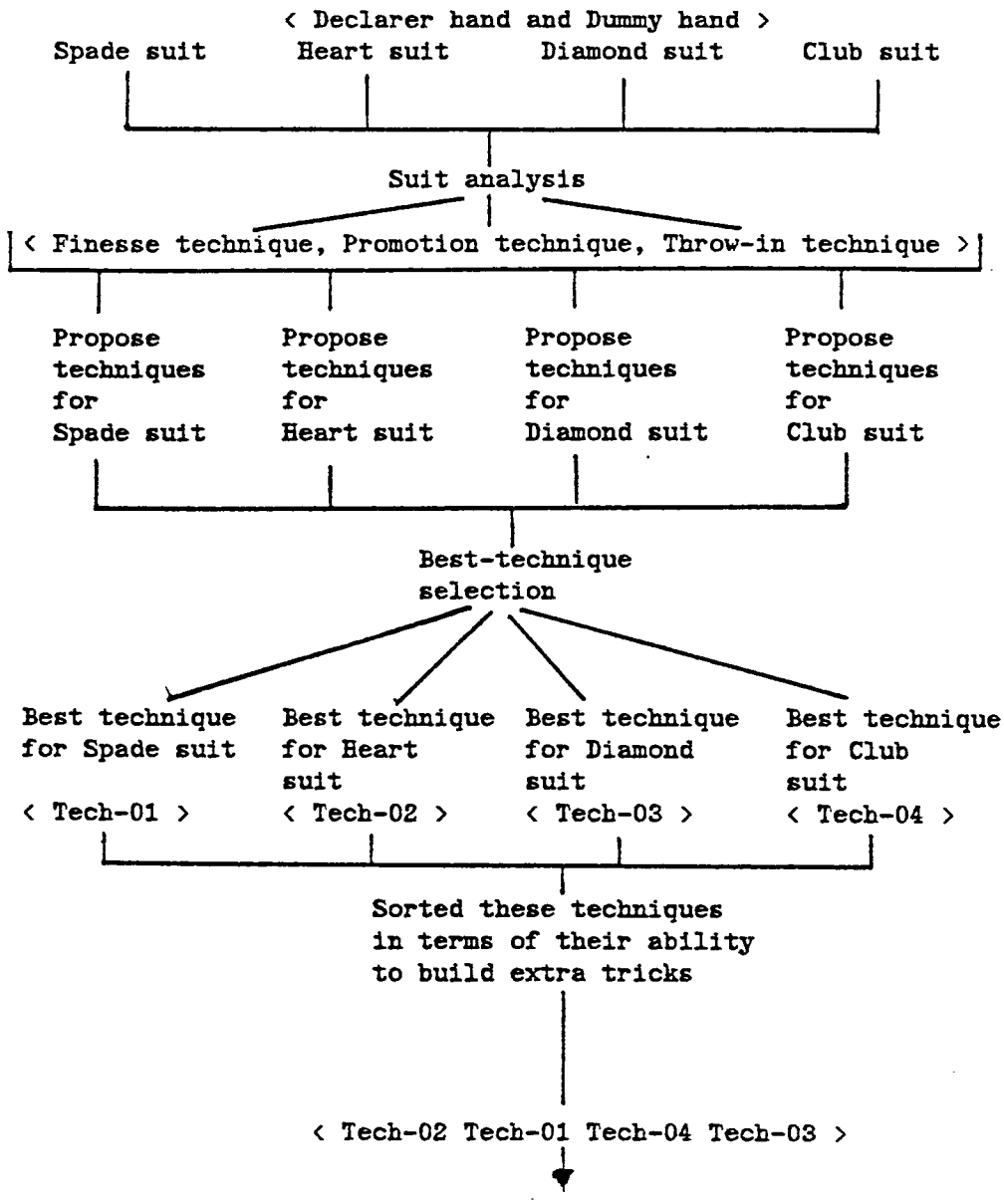
  -- a set of rules is used to decide the second hand player's moves.
  -- another set of rules is used to decide the fourth hand player's moves.

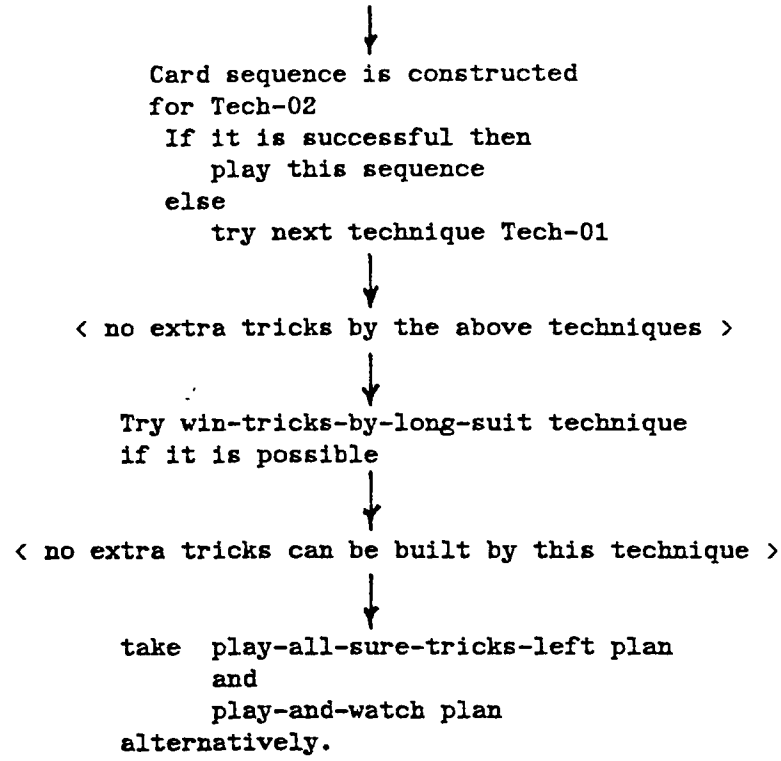
* Opponent side is in either the defensive or offensive position

  -- Every opponents' move is entered from the screen terminal.

2.1 The planning phases :

In our approach, the planning process is divided into three phases. The first phase is the suit analysis phase. The second phase is the best-technique selection phase. The last phase is the card sequence construction phase. The process can be pictured as follows :





2.1.1 The suit analysis phase :

The task of the suit analysis phase is to propose playing techniques for each suit in the Declarer-Dummy's hands. At the end of this phase, a technique-list is generated for each suit if it is possible to build extra tricks in this suit. If a suit doesn't have any extra tricks to build, then there is no technique-list found for this suit. For example, the following technique-lists are generated if extra tricks can be built in all four suits.

Technique-list-1 for Spade suit
Technique-list-2 for Heart suit
Technique-list-3 for Diamond suit
Technique-list-4 for Club suit

Each technique-list contains the general information and the the specific information. The general information includes a card pattern describing the cards in the Declarer-Dummy's hands, the expected number of tricks that a suit can make and a list of possible techniques. The specific information is the detailed information of of each technique in the list. As we have mentioned in the Chapter 1, a technique can be one of the following techniques :

* Finesse technique
* Promotion technique
* Throw-in technique

In our approach, the win-tricks-by-long-suit technique is used when the above techniques are no longer feasible and if extra tricks can be built by this technique.

2.1.2 The best-technique selection phase :

The task of this phase is to select the best technique for each suit and then sort them in terms of their ability to build extra tricks. In the previous example, a technique is selected for each suit from the technique-list generated at the first phase. every technique in the list has its own precondition, limitations such as entry limitation and the limitation of leading toward a particular opponent, and risks such as losing a trick to an unexpected opponent, etc. The examination starts at the first technique in the technique-list for a suit. It stops as long as the precondition of

a technique satisfies with the current situation. The technique found so far is considered as the best technique for the suit being examined. The last technique in the list is respected as a default technique for the suit being examined. After examining the four techniques found in the previous phase, the result is generated as follows :

In Spade suit, tech-01 is the best among those techniques in technique-list-1.

In Heart suit, tech-02 is the best among those techniques in technique-list-2.

In Diamond suit, tech-03 is the best among those techniques in technique-list-3.

In Club suit, tech-04 is the best among those techniques in technique-list-4.

If there is no techniques-list found in a suit, then no examination is performed for this suit during this phase.

2.1.3 The card sequence construction phase :

The task of this phase is to select cards and formulate them into a playing sequence for each technique in the sorted list generated by the best-technique selection phase. The construction is a process of traversing a plan tree. The offensive play made by the Declarer side represents the root of this tree, and each technique represents a subtree of this root. Each node in this tree is a plan.

In the previous example, the offensive plan consists of four subplans. Each of these subplans corresponds to each technique in the list. The first technique in the sorted list is applied first. A card sequence is built by traversing the subtree of this technique. It is possible that a card sequence is unable to be built for some reasons. If this happens, we say that this technique is unfeasible. Thus, the next technique in the list is tried.

If there is no technique left in the list, the suits are examined to see if it is possible to build extra tricks by taking the the Win-tricks-by-long-suit technique. If a suit is found to have this possibility, then a card sequence is constructed for this technique. If no extra tricks can be built, two plans are used alternatively to play the cards left in the players' hands. They are the Play-all-sure-tricks-left plan and the Play-and-watch plan. Here, Here, the term "technique" refers to a playing technique used in Bridge and the term "plan" refers to the steps of applying a technique. A technique is applied by executing a plan.

The Play-all-sure-tricks-left plan is executed first. This plan tries to play all sure tricks left in the Declarer-Dummy's hands. If there are no sure tricks left, the Play-and-watch plan is then executed. The intention of this plan is to turn the Declarer side from the offensive position into the defensive position and wish that the Declarer side can establish sure tricks by chance during the de-

fensive play.

In order to simulate a human playing in a game and demonstrate how planning is performed, the games played by computer will be proceeded by rotating between the following two modes :

Simulation mode -- In this mode, the game pauses. A card sequence is built for a technique or a plan. If the sequence can be built, then the simulation stops and the game is set to "game" mode. If it fails to build the sequence, then the next technique is tried and the simulation proceeds.

Game mode -- In this mode, the game proceeds. In each run, every player plays a card. If the Declarer is in the offensive position, the card played is retrieved from the sequence generated in the simulation mode. Otherwise, the card played is selected by scanning rules. If the opponents are in the offensive position, the card is entered from the terminal screen.

2.2 Replanning :

When a technique becomes unfeasible, the replanning process will be taken for the related suit. However, the replanning process will not be performed immediately unless there is no other technique left in the sorted techniques-list or all the techniques left in this list are also unfeasible.

When a replanning process needs to be performed, the same steps used in the planning process are taken. Actually from an implementation point

of view, there is no difference between planning and replanning.

CHAPTER 3

THE OVERALL APPROACH OF KNOWLEDGE REPRESENTATION

In order to play Bridge, we know we need the knowledge to suggest the techniques for suits, the knowledge to construct a card sequence for a technique, the knowledge for the declarer playing a defensive position, the knowledge for bidding, and the knowledge for making an open lead. We are only concerned with the first three kinds. In order to store these knowledges into a knowledge base, we need to arrange them in the way that the retrieving, adding, and updating of these knowledges is efficient.

Knowledge	Storing structure	Used by
Facts	Simple associated list	All phases of the planning process
Analysis information	Rules	Suit-analysis phase
Technique information	Complex associated list	Best-technique selection phase and The card sequence construction phase
Technique applying information	Complex associated list	The card sequence construction phase

In the later explanations, we will use the term "frame" to refer to the complex associated list. The details and implementation of frame representation can be found in the paper [7].

3.1 Representation of facts :

Hand information

The structure of storing a hand is the associated list. The header of the list is the name of a player. Under it there are attributes - Spades, Hearts, Diamonds, and Clubs. The value of each attribute is a list of cards. The following example shows the hand held by West.

```
(west  (spade    (Q 10 6 3))
       (heart    (Q 10 7))
       (diamond  (6 5 2))
       (club     (K 10 6)))
```

Bidding information

The structure of storing the bids is a simple list. Each element in the list represents a bid. The first item of an element is the name of the bidder. The second item consists of the bid-level and the bid-suit. The value of the bid-level is in the range of (0, 7). The value of bid-suit can be Spades, Hearts, Diamonds, Clubs, NT, Pass, Double, or ReDouble. The following list is an example.

```
((west (0 pass)) (north (1 diamond))
         (east (0 pass)) (south (2 nt))
 (west (0 pass)) (north (3 nt))
         (east (0 pass)) (south (0 pass))
 (west (0 pass)))
```

The declarer and the contract are not specified explicitly. Their values can be derived from the above list. After scanning the entire list, the value of a bid-suit is NT, and the bid-level is three. The declarer is SOUTH.

The lists we discussed above are external representations of facts. This representation is easy to understand, but it is not easy for implementation. For example, the Spade suit has cards (Q 10 6 3) and there is a set of rules whose premises describe the various card-patterns (shapes). If we write a statement to check whether the Queen exists or not, it is obvious that the statement must invoke a function to implement this checking. Thus, many functions are needed for the various checkings. Another problem with this representation is repeatly implementing the same checking on each scanning of rules. If one of rules checks whether the Queen and the Jack exist, and another rule checks whether the Queen and the 10 exist, then the checking on the Queen is implemented twice. Therefore, poor performance of the rule scanning process is expected by using this representation. To get better performance, we must convert this representation into a form which is easy to implement. In order to perform this conversion, two sets of statements are defined. One is for describing a hand. The other one is for describing bidding.

Group 1 statements

```
(spade is the major suit)
(heart is the major suit)
(diamond is the minor suit)
(club is the minor suit)
(?suit has no more than 4 cards)
(?suit has at least 5 cards)
(?suit has ?cards  cards)
(?suit has 1 card)
(?suit is in complete sequence)
(?suit is in partially complete sequence)
(?suit is not in sequence)
(?suit sequence is headed by card ?card)
(?suit has ?card)
(?suit has a missing ?high-card)
(?suit has no high cards)
(?suit is the longest suit)
(?suit is the strongest suit)
(?suit is the shortest suit)
(there are entries in suits other than ?suit)
(?suit has ?entries entries)
(?suit has 1 entry)
(?suit has at least ?entries entries)
(?suit does not have an entry)
(?suit1 has the same length as ?suit2)
(?suit1 is better than ?suit2)
(?suit1 has more cards than ?suit2)
(?suit1 has the same texture as ?suit2)
```

Group 2 statements

```
(the contract is notrump contract)
(the contract is trump contract)
(?suit is the trump suit)
(?player is the declarer)
(?player bid ?suit for the ?bid-time  time)
(?player has bid ?suit once)
(?player never bid ?suit)
(?player has bid ?suit at the ?bid-level level)
(?player never bid any suit at all)
(?player single raise bid ?suit suit)
(?player jump raise bid ?suit suit)
(?player bid double against ?suit)
```

In order to make a statement match as many facts as possible, the variables are used in the statement. The word prefixed with "?" is recognized as a variable. Each variable has an assigned range of valid values. When a statement is accessed, the binding process is executed to substitute the variables in this statement with the correct values. The paper [8] discusses how the binding process is executed. The following table shows the variables and their possible values.

```
?player      : west, north, east, south
?suit        : spade, heart, diamond, club (for hand)
               nt, double, redouble, pass  (only for bidding)
?suit1,
?suit2       : spade, heart, diamond, club
?card        : ace, king, queen, jack, 10, 9, 8, 7 , 6 , 5, 4,
               3, 2
?high-card   : ace, king, queen, jack
?entries     : number of entries (sure tricks)
?cards       : number of cards
?bid-level   : first, second, third, forth, game
?bid-time    : first, second, third, forth, fifth, sixth,
               seventh
?1           : the highest-ranking card in the suit
?2           : the second highest-ranking card in the suit
?3 .... ?14  : the rank of a card in the suit
```

Now, let's see an example. The Spade suit in the following hand :

```
(west  (spade    (Q 10 6 3))
       ...................)))
```

can be converted into a list of statements

```
((SPADE IS THE MAJOR SUIT) (SPADE HAS NO MORE THAN 4 CARDS)
 (SPADE HAS 4 CARDS) (SPADE IS NOT IN SEQUENCE)
 (SPADE HAS A MISSING ACE) (SPADE HAS A MISSING KING)
 (SPADE HAS QUEEN) (SPADE HAS A MISSING JACK) (SPADE HAS 10)
 (SPADE HAS 6) (SPADE HAS 3) (SPADE DOES NOT HAVE AN ENTRY)
 (SPADE IS THE STRONGEST SUIT) (SPADE IS THE LONGEST SUIT)
 (SPADE HAS MORE CARDS THAN DIAMOND)
 (SPADE HAS MORE CARDS THAN CLUB)
 ..............)
```

and the bidding

```
((WEST (0 PASS)) (NORTH (1 DIAMOND))
        (EAST (0 PASS)) (SOUTH (2 NT))
 (WEST (0 PASS)) (NORTH (3 NT))
        (EAST (0 PASS)) (SOUTH (0 PASS))
 (WEST (0 PASS)))
```

can be converted into

```
((THE CONTRACT IS NOTRUMP CONTRACT) (SOUTH IS THE DECLARER)
 (WEST BID PASS FOR THE FIRST TIME)
 (NORTH BID DIAMOND FOR THE FIRST TIME)
 (NORTH BID DIAMOND AT THE FIRST LEVEL FOR THE FIRST TIME)
 (EAST BID PASS FOR THE FIRST TIME)
 (SOUTH BID NT FOR THE FIRST TIME)
 (SOUTH BID NT AT THE SECOND LEVEL FOR THE FIRST TIME)
 (WEST NEVER BID ANY SUIT AT ALL) (NORTH NEVER BID SPADE)
 (NORTH NEVER BID HEART) (NORTH HAS BID DIAMOND ONCE)
 (NORTH HAS BID DIAMOND AT THE FIRST LEVEL)
 (NORTH NEVER BID CLUB)
 (NORTH HAS BID NT ONCE) (NORTH HAS BID NT AT THE GAME LEVEL)
 (NORTH NEVER BID DOUBLE) (EAST NEVER BID ANY SUIT AT ALL)
 (SOUTH NEVER BID SPADE) (SOUTH NEVER BID HEART)
 (SOUTH NEVER BID DIAMOND)
 (SOUTH NEVER BID CLUB) (SOUTH HAS BID NT ONCE)
 (SOUTH HAS BID NT AT THE SECOND LEVEL)
 (SOUTH NEVER BID DOUBLE))
```

3.2 Representation of knowledge :

The knowledge in Bridge can be classified into several categories :

(a) knowledge for interpreting opponents' moves
(b) knowledge for the Declarer side playing the defensive position
(c) knowledge for analyzing a hand
(d) knowledge about playing techniques
(e) knowledge about using a playing technique

The (a) category of knowledge is not included in our study, so we do not discuss it. The (b)(c) categories of knowledge are stored in rules. The other two categories of knowledge are stored in frames.

3.2.1 The scheme of expressing knowledges in rules :

The structure of an individual rule has the following form :

```
(Rulename
   (Premises   ((statement-1)
                (statement-2)
                ........
                (statement-n)))
   (Actions    ((action-1)
                (action-2)
                ........
                (action-m))))
```

Premises

The statements in the premises part of a rule are the statements defined in the previous section. The logical relation of statements is an AND relation. However, the logical operator NOT can be used in a statement. For example :

```
(premises ((?suit has ?1)
           (not (?suit has ?2))))
(actions  ((.....))))
```

The negative statement (not (?suit has ?2)) is the same as the statement (?suit has a missing ?2). (Null conditions) is a special statement which is similar to the ELSE statement used in other programming languages. It guarantees that the forward reasoning on a set of rules has a terminated point.

Actions

An action in "actions" list is a LISP macro-function. An action can be a NOP. In this case, the function DO-NOTHING is invoked.

Example

```
(Analysis-rule-001
  (premises ((?suit has  ?1)
             (not (?suit has  ?2)) ))
  (actions  ((Propose  Technique-001)) ))
```

In this example, ?suit, ?1, and ?2 are binding variables and Propose is a macro-function which proposes the technique "Technique-001".

3.2.2 The scheme of expressing Knowledges in frames :

The knowledge about playing techniques such as Finesse, Promotion and Throw-in are stored in a frame structure. The knowledge about constructing a card sequence for each of the playing techniques is also stored in a frame structure. The contents of these two frames are different.

3.2.2.1 Storing technique information in frames :

The structure of storing technique information is a complex associated list which we refer as the "frame". Each frame has seven primary attributes. They are

(a) Player-relations
(b) Max-card-index
(c) Holdings
(d) Sure-tricks
(e) Sure-tricks-list
(f) Extra-tricks-list
(g) Techniques

The attribute (g) has six attributes under it. They are

(g1) Depend-on
(g2) Objective
(g3) Avoid
(g4) Risk
(g5) Lead
(g6) Technique-name

The general format of this category of knowledge is :

```
(Tech-info-id
  (player-relation     value-1)
  (max-card-index      value-2)
  (holdings            value-3)
  (sure-tricks         value-4)
  (sure-tricks-list    value-5)
  (extra-tricks-list   value-6)
  (more-extra-tricks   value-7)
  (techniques          techniques-list) )
```

The format of a technique in techniques-list is :

```
(technique-sequence-no
   (depend-on       statement-1)
   (objective       statement-2)
   (avoid           statement-3)
   (risk            statement-4)
   (lead            statement-5)
   (technique-name  statement-6) )
```

Now, we explain the meaning of each attribute used in the following example.

```
(Tech-info-4-3-002
  (Player-relations   ((?p1 ?p2) (?right-op ?p1) (?left-op ?p1))
  (Max-card-index     3)
  (Holdings           (((?1 ?3 ?x1 ?x2) (?x3 ?x4 ?x5 ?y))))
  (Sure-tricks        1)
  (Sure-tricks-list   ((?p1 (?1))))
  (Extra-tricks-list  ((Technique-01  1) (Technique-02  1))))
  (Tricks-won-by-length  0.36)
  (Techniques
    (Technique-01
      (Depend-on      ((?right-op  has the ?suit  ?2)))
      (Objective      ((?3 wins 1 trick)))
      (Avoid          nil)
      (Risk           ((?left-op may gain the lead)))
      (Lead           ((?p2))
      (Technique-name ((Finesse  ?3  ?p1))) )
    (Technique-02
      (Depend-on      nil)
```

```
(Objective       ((?3 wins 1 trick)))
(Avoid           ((?left-op gains the lead)))
(Risk            ((?right-op may gain the lead)))
(Lead            ((?p2  ?p1))
(Technique-name ((Throw-in  ?right-op))) )))
```

Attribute (a) -- player relation

This attribute is used to build the binding information for the player relation. The example shows that the ?left-op is the left-hand opponent of ?p1. If ?p1 is West, then the value of ?left-op is North.

Attribute (b) -- Max-card-index

In order to perform pattern matching on card holding in a suit, the card held in a suit must be converted into a pattern where only the significant cards are recognized.

The value of the Max-card-index indicates the most significant card. For example: if the value is 2, only the first two highest ranking cards are significant. The pattern appeared in attribute "holdings" has the elements ?1 and ?2. These are the binding variables. The number indicates its ranking.

Attribute (c) -- Holdings

The value of this attribute is a matching card-pattern. It decides whether or not a technique-information list can be used

for a suit. If the Declarer side holds the cards in a suit whose pattern is the same as the one described in this attribute, all information stored in this list is related to the playing of this suit. The variables used here have their own meanings.

?1 ?2 ....?n : represents the rank of a card. ?1 represents the highest card currently left in the suit being examined.

?x1 ?x2... : represents any other cards which are not significant for the pattern matching. The "x" in ?x1 indicates a card in the suit being examined.

?y : represents a card in the side suit. The use of "?y" is mainly for the convenience of implementation. If the cards in a suit are divided into 3 and 2, then the pattern will be built as follows :

((?x1 ?x2 ?x3) (?x4 ?x5 ?y))

Now, two sub-lists have same number of elements.

Attribute (d) -- sure-tricks

It tells how many sure tricks are in the suit being examined.

Attribute (e) -- sure-tricks-list

It contains a list of sure tricks held by both Declarer and Dummy.

Attribute (f) -- extra-tricks-list

It contains the information about how many extra tricks can be built and what kind of technique to use.

Attribute (g) -- Tricks-won-by-length

It tells how many extra tricks can possibly be made by taking advantage of the suit length. The attribute (f) tells how many tricks can be built by a technique other than the Win-tricks-by-long-suit technique. To store the information of extra tricks exactly built by the Win-tricks-by-long-suit technique and the other techniques separately won't miscalculate the real contribution made by a technique other than the Win-tricks-by-long-suit technique. The information stored in this attribute can tell whether a suit can provide more extra tricks if there are no techniques other than the Win-tricks-by-long-suit technique found to be feasible.

Attribute (ha) -- depend-on

The content of this attribute is a list of conditional statements. If the conclusion of statements is true, then this technique is said to be applicable. The statements used always return answers with some degree of certainty. As we mentioned before, checking on the preconditions stored in this attribute

may expect an answer from the scheme which was designed to provide this answer. For example, if the declarer is missing King in Spade, he can apply the "Finesse" technique by leading a small card from his partner toward the right-hand opponent. In order to finesse the Spade Q successfully, the technique expects that the right-hand opponent must have the Spade King. When deciding if the "Finesse" technique should be applied, the declarer needs an answer for the question "which opponent is likely to have the King of the Spade?". The answer is a statement which states :

(It is likely that West has the Spade King)

The word "likely" is one of certainty phrases defined in this study. The underlined text is part of your question. From the table of certainty phrases, we can find a range of values corresponding to each phrase. In this example, the value of the phrase "likely" is between 0.6 and 0.7.

Certainty phrases table

Certainty phrases	Certainty values
Definite	0.9 - 1.0
Evident	0.8 - 0.9
Undoubtedly	0.7 - 0.8
Likely	0.6 - 0.7
Probably	0.5 - 0.6
Dubious	0.4 - 0.5
Unlikely	0.3 - 0.4
Merely	0.2 - 0.3

Improbably	0.1 - 0.2
Definitely-not	0.0 - 0.1

An article related to this area of study can be found in the premier issue of the AI EXPERT [6]. This article introduces the Inexact Reasoning model, which permits customizable inexact reasoning.

So in the example we used here, the "finesse" is unlikely to be successful because West is the opponent who is more likely to be holding the Spade King.

Attribute (hb) -- objective

It tells which card will be the vital card when the technique is used. The technique becomes inapplicable whenever these cards no longer exist.

Attribute (hc) -- avoid

It tells what must be avoided so that a technique can be applied successfully.

Attribute (hd) -- risk

It posts what kind of threat might be generated if the technique fails to apply.

Attribute (he) -- lead

It tells who is going to lead the card. Sometimes it does not matter who is going to lead the card. However, for some techniques to be applied successfully, the lead must be restricted to a particular entry.

Attribute (hf) -- technique-name

It contains the technique name and its parameters. In our study, the value of this attribute can be

* Finesse
* Promotion
* Throw-in

Example

In the following hand, South is the Declarer, and the contract is 3NT.

(NORTH - dummy)
Spade 8 5 2
Heart A 5 4
Diamond A K Q J
Club 7 5 2

openlead
Spade 3

(SOUTH - declarer)
Spade K J 7
Heart K J 6
Diamond 9 8 4
Club A Q 8 3

The analysis on this hand will find at least one technique-information list for each suit. We can retrieve these lists from the knowledge base. In the Club case, there is only one list being retrieved. It looks like :

```
(Tech-info-4-3-002
  (Player-relations    ((?p1 ?p2) (?right-op ?p1) (?left-op ?p1))
  (Max-card-index      3)
  (Holdings            (((?1 ?3 ?x1 ?x2) (?x3 ?x4 ?x5 ?y))))
  (Sure-tricks         1)
  (Sure-tricks-list    ((?p1 (?1))))
  (Extra-tricks-list   ((Technique-01  1) (Technique-02  1))))
  (Tricks-won-by-length  0.36)
  (Techniques
    (Technique-01
      (Depend-on       ((?right-op  has the ?suit  ?2)))
      (Objective       ((?3 wins 1 trick)))
      (Avoid           nil)
      (Risk            ((?left-op may gain the lead)))
      (Lead            ((?p2))
      (Technique-name ((Finesse  ?3  ?p1))) )
    (Technique-02
      (Depend-on       nil)
      (Objective       ((?3 wins 1 trick)))
      (Avoid           ((?left-op gains the lead)))
      (Risk            ((?right-op may gain the lead)))
      (Lead            ((?p2  ?p1))
      (Technique-name ((Throw-in  ?right-op))) )))
```

This list provides the following information for the Club suit.

a. The binding process generates the following list for the Club suit.

```
((?1 A) (?2 K) (?3 Q) (?x1 8) (?x2 3) (?x3 7)
 (?x4 5) (?x5 2) (?suit Club) (?p1  South) (?p2 North)
 (?left-op East) (?right-op  West))
```

As we have mentioned earlier in this chapter, the variable ?y is used for the convenience of implementation, it does not have any significant meaning for using the rest infomation in the list. Therefore, it is ignored by the binding process.

b. The Club suit has one sure trick.

c. Two techniques can be used by the Club suit. They are

Technique-02 and Technique-01

Both techniques can build one extra tricks. According to an earlier explanation, the technique-01 is the first technique in the "techniques" list. This means that it possess higher chance to win one extra trick. But whether we use technique-01 or not, we depend on the truthness of the "depend-on" attribute of this technique. Now, we examine these two techniques closely.

Technique-01
==========

It can be used when the condition in the "depend-on" is true. It posts its objective. There is no restriction for playing with this technique. But a possible threat might be generated. The threat is "the left-hand opponent might gain the lead if the technique fails to apply". The attribute "lead" tells a card must be led from player ?p2 so that the 3rd-hand player can finesse his 4th-ranking card.

Technique-02
============

It is a default technique (null condition). Its objective is the same as technique-01, but it uses the Throw-in technique. In order to make it work, the player must avoid the ?right-op to gain the lead before playing Spade suit. The threat of this play is that ?left-op may gain the lead if the Throw-in technique fails to apply. The attribute "lead" tells that it prefers ?p1 to lead the card when this technique is applied. If there is no way to build entry at ?p1, the lead from ?p2 can be considered.

3.2.2.2 Storing cards construction knowledge in frames -

In Chapter 2, we have mentioned that the construction of a card sequence is a process of traversing a plan tree. Each node in the tree represents a plan. A higher-level plan has at least one sub-plan. The sub-plans represent the steps to implement their patrent plan. The plan in a leave node of this tree is the plan which plays a card. The information of a plan is stored in a frame.

The general format of this category of knowledge is :

```
(plan-name
   (arguments         value-1)
   (plan-level        value-2)
   (player-relation   value-3)
```

```
(suits            value-4)
(cards            value-5)
(premises         statements)
(actions          actions-list-1)
(alternatives     actions-list-2) )
```

Attribute (a) -- arguments

It stores a list of variables for the instantiation of the plan.

Attribute (b) -- plan-level

There are three levels defined in this system. They are root-level, suit-level and card-level. A plan whose level is suit-level only deals with the suits. A plan whose level is card-level deals with the cards in the selected suit. Our control mechanism has two control loops. The outer loop is used for suits, the inner loop is used for cards. Therefore, the suit-level plan never runs the inner loop.

Attribute (c) -- player-relations

It stores the binding information for the variables used in the plan.

Attributes (d) -- suits

It contains the names of suits needed to be examined.

Attributes (e) -- cards

It contains the names of cards in the selected suit needed to be examined.

Attribute (f) -- premises

It contains the conditions to drive out an unwanted element in the selected suits/cards.

Attribute (g) -- actions

It contains the names of subplans, the basic move, or the predicates for checking status.

Attribute (h) -- alternatives

It contains the names of subplans, the basic move, or the predicates for checking status. They were used when the premises makes the false conclusion.

Example
==========

The example here is a plan for discarding a card.

```
(discard-a-card
 (arguments       (?suit1  ?player))
 (plan-level      suit-level)
 (player-relation nil)
 (suits           (?suit in (the-most-useless-side-suit
                             ?suit1 ?player)))
 (cards           nil)
 (premises        nil)
 (actions         ((do drop-a-card  ?suit  ?player)))
 (alternatives    nil) )
```

CHAPTER 4

THE EVALUATION SCHEME

In this chapter, we are going to survey the kind of evaluation scheme which can be used to evaluate a hand, to evaluate a threat, and to evaluate the uselessness of a suit. The evaluation of a hand simply tells how many sure tricks can be made and how many extra tricks can be made by a technique. The evaluation of the threat tells which opponent would cause more damages if he gains the lead. The evaluation of the uselessness gives each suit in hand a value to indicate its uselessness.

4.1 Counting the contributions of a suit :

In the Chapter 3, we have discussed the structure for storing technique information. Here, we are going to use the same example to do our discussion. The example is simplified so that only the values of those attributes which are related to the discussion of this topic are listed.

```
(Tech-info-4-3-002
   (Player-relations   .......)
   (Max-card-index     .......)
   (Holdings ..................)
   (Sure-tricks        1)
   (Sure-tricks-list   .......)
   (Extra-tricks-list  ((technique-01  1) (technique-02  1))))
   (Tricks-won-by-length  0.36)
   (Techniques ..... )
```

The contribution is represented by a value which is calculated by adding the number of extra tricks built by the technique and the number

of tricks gained by taking advantage of the length of the suit together.

In this example, we are interested in how the values of Sure-tricks, Extra-tricks-list, and Tricks-won-by-length are calculated.

Sure-tricks	: It stores the number of sure tricks made by the suit being examined.
Extra-tricks-list	: It stores a list of techniques and the number of tricks they can make.
Tricks-won-by-length	: It stores the number of extra tricks built by taking advantage of the length of the suit being examined.

4.1.1 Calculating the number of sure tricks :

Sure tricks are those tricks which you can take without giving up the lead to your opponents. They are ready-made tricks such as Aces and can usually be taken at any time. The example we use below shows there is only one sure trick, the Ace (the symbol ?1 in "holdings" represents the most highest card). In another example :

Dummy : K 6 3
Declarer : A Q 7

The Ace, King, and Queen are the sure tricks.

4.1.2 Calculating the number of extra tricks :

In the Notrump contract, the extra tricks can be built by

using one of the following techniques :

* Extra tricks built by taking finesse
* Extra tricks built by promotion
* Extra tricks built by taking advantage of the length of suit.
* Extra tricks built by forcing the opponent to play suit first.

4.1.2.1 Extra tricks built by promoting card :

One way to build extra tricks is through the "promotion" of cards. The basic idea is that a card is turned into a sure trick when all the higher-ranking cards in that suit have been played. Here are some examples :

a. Dummy : 4 3 2
Declarer : K Q J

--> lead the King to force the opponents to play their Ace. if Ace was played, you are able to build two extra tricks by promoting the Queen and the Jack. They are now the highest-ranking cards remaining in that suit.

b. Dummy : 6 5 4
Declarer : Q J 10

--> In this example, you are missing both the Ace and the King. You can still promote a trick with a little work. Lead your Queen to drive out the opponent's Ace or King. Next time you have an opportunity, lead the Jack to drive out the opponents' remaining high card. Now, because you have the 10, you have built a trick in the suit.

c. Dummy : 5 4 3 2
Declarer : J 10 9 8

--> This time you'll have to be very patient. The opponents have the Ace, King, and Queen. Use your Jack to drive

out one of their cards, your 10 to drive out another and your 9 to drive out the remaining high card. Eventually, you will have established your 8 as the highest card remaining in the suit.

4.1.2.2 Extra tricks built by finesse :

Another way to build tricks is by trapping the opponents' high cards. For this to work, you need the proper technique and some luck. There is only one extra trick which can be built when this technique is applied. Let's take a look at an example :

Dummy : K 5
Declarer : 4 2
(you)

--> You can lead a small card from your hand toward the King and give yourself a 50% chance of winning a trick. When you lead the 2 from your hand first, the opponent on your left must play before you have to choose a card from the dummy. If your left-hand opponent has the Ace and plays it, you can play your 5 from dummy and save the King to take a trick latter. If your left-hand opponent has the Ace but doesn't play it, you can play dummy's King on this trick to win the trick. Since it is also possible that the right-hand opponent holds the Ace, there is only a 50% chance of winning a trick with your King.

To make the play of this technique successfully, the location of a card we want to trap must be known precisely.

4.1.2.3 Extra tricks built by throw-in :

Another way to build extra tricks is to force the opponent to play a suit first. The example shown below helps to

explain why.

Dummy	7 3 2
Declarer	A Q 6

--> Suppose you are the declarer. It is un-wise for you to play this suit first because if you play this suit, you only can take 1 trick by using your Ace. But, if you let your left-hand opponent plays this suit first, then the Queen in your hand may have a chance to win one trick.

4.1.2.3 Extra tricks built by suit length :

Another way to build tricks is to use your long suits to establish extra tricks. Let's look at some examples of how this is done.

Dummy	: 6 5 4 3
Declarer (you)	: A K 7 2

--> In this example, you have eight cards in a suit which means that the opponents have only five between them. In most cases, the five outstanding cards will be divided between the opponents, three in one hand and two in the other. How does this help you ?

After you play the suit two runs, there will only be one high card left in the opponents' hands. If you lead the suit again playing little cards from both hands, you will lose the trick to the opponents. However, when next regain the lead, you will be able to take a trick with your remaining card because the opponents will have no cards left in the suit.

The trick contributions of using this technique at the beginning of game is evaluated by taking the most cases which

the adverse cards will be divided into consideration. The values in the column of "% of the time" shown on the following table which is taken from the book of Goren's Bridge Complete [4] are used as the default values. These values can be adjusted according to the bidding and the deductions drawn during the game.

Your side's hold	the adverse cards will be divided	% of the time
6 cards of a suit	4-3	62
	5-2	31
	6-1	7
	7-0	< 0.5
7 cards of a suit	4-2	48
	3-3	36
	5-1	15
	6-0	1
8 cards of a suit	3-2	68
	4-1	28
	5-0	4
9 cards of a suit	3-1	50
	2-2	40
	4-0	10
10 cards of a suit	2-1	78
	3-0	22
11 cards of a suit	1-1	52
	2-0	48

At the beginning of this section, an example of a technique is shown. The value stored in the attribute TRICKS-WON-BY-LENGTH of this list is 0.36. Now, let's see how it was calculated.

In this example, there are six adverse cards and the cards in your side are divided into four and three. We can use the following formula to calculate the value of the TRICKS-WON-BY-LENGTH.

```
number of extra tricks = [(chance #1 of adverse cards divided)
                          * (the number of cards over the cards
                             held by opponents) ] +
                         ..............................
                         [(chance #n of adverse cards divided)
                          * (the number of cards over the cards
                             held by opponents) ]
```

The following table shows how this formula is used to compute the value of 0.36 in our example. The values in the first column show all possible distributions of adverse cards. The values in the second column are the number of extra tricks made by these distributions. The values in the third column show the chances of these distributions. The values in the fourth column are the results computed by taking the formula described above. So, by adding the values appearing in the fourth column, the value of 0.36 is generated.

adverse cards divided	absolute number of extra tricks	% of time	tricks calculated
4-2	0	48	0
3-3	1	36	0.36
5-1	0	15	0
6-0	0	1	0
The value of TRICKS-WON-BY-LENGTH =			0.36

4.2 Counting the threat of playing a card :

Sometimes, you may enter into a situation when there is no sure trick you can play with. In this case, you have to play a card which is expected to lose to one of the opponents. Suppose it is your turn to lead a card, you must decide which suit is the best suit to lead. The criteria for judging which suit is the best is :

"what degree of loss can be generated by playing a card from that suit."

To lead a card toward either one of opponents, you need to decide "who is the most favor opponent". In other words, which opponent will cause less damage on your side whenever he gains the lead. A term "threat" is used to describe the threat from either one of the opponents on the card you intend to play.

To calculate the degree of the "threat", the following rules are used :

* If your opponent doesn't have any cards left in the suit you will lead a card from, then the degree of threat is 0.
* If your opponent has at least one sure trick left in the suit you will lead a card from, then the degree of threat is calculated as follows :

the degree of threat #1 = (-1.0) * (probability for an opponent holding sure trick #1)

:

:

the degree of threat #n = (-1.0) * (probability for an opponent holding sure trick #n)

The total threat = (the degree of threat #1) + ...
(the degree of threat #n)

The idea here is : if an opponent holds a sure trick in the suit you will lead a card from, then he can use it to gain the lead and play all the sure tricks in his hand. Or he can gain the better position to lead a card toward his partner to cause you more loses. The latter happens when you are holding a suit which has a missing high card and you are afraid your right-hand opponent may lead a card in that suit toward his partner.

This result may need further refinement. That means, an entry factor might be added into the calculation. The entry factor is defined as "the ability of an opponent leading a card toward his partner". Suppose, one opponent holds the six of Spades and Ace of Hearts, the other opponent holds the King of Spades and others, and you, holding Spades (A Q x x .), are the right-hand player of the opponent holding the Spade K. In a case like this if you lead a small Heart, the opponent wins it with the Ace of Hearts, then he leads the six of Spades toward his partner's King of Spades. By using a suitable technique, you probably can win two tricks in the Spade suit. But now, you can only win one trick with the Ace of Spades. The calculation of entry factor is a little bit complicated. So far we are not concerned about it. Later, we may need to use it to adjust the "threat" calculated so that it can tell us precisely which card should be played toward which opponent with the least damage.

4.3 Evaluating the uselessness of a suit :

When the game is close to its end, it often happens that you need to discard a card. You certainly don't want to discard a useful card. But among possible candidates, which card is the most useless card? To decide which card should be discarded, the suits in your hand must be evaluated, and the one which has assigned a highest value is selected as the most useless suit. After a suit was selected, you then need to decide which card should be played. If there is no special consideration, the lowest-ranking card is the one you should play. Now in this study we evaluate the suits to decide which one is the most useless suit, and we always select the lowest-ranking card to play.

To decide the uselessness of a suit, we construct the following equation.

```
degree-of-usefulness =
          0.1 * ((the face value of lowest-ranking card in suit) -
                 (the length of the suit))
          - [ 0.1 ]
```

--> the 0.1 circled with brackets is optional. If opponents have bid this suit, then the weight of this effect must be counted.

A card can be considered as "most useless" is the card which has the minimum degree-of-usefulness.

The idea here is : if the suit was bid by the opponent before, then the suit tends to be "useless". In a Notrump contract, the winning trick is determined by comparing with the face values of cards played. Therefore, a card with the lowest face value is the most useless card.

If one suit has more cards than other suits, then dropping a card from this suit will not cause more damage than the one from other suits.

Here, we use an example to explain how it works. Suppose the cards in a hand are :

Spades : (A Q)
Hearts : (10 6 5 4)
Diamonds : (A K 4 3)
Clubs : (10 8 7)

Also suppose the opponent didn't bid any suit. By using the formula outlined above, the values are generated as follows :

Spades : ((the rank of the Queen) - (the length of the suit)) * 0.1
= (12 - 2) * 0.1 = 1.0

Hearts : ((the rank of the 4) - (the length of the suit)) * 0.1
= (4 - 4) * 0.1 = 0.0

Diamonds : ((the rank of the 3) - (the length of the suit)) * 0.1
= (3 - 4) * 0.1 = -0.1

Clubs : ((the rank of the 7) - (the length of the suit)) * 0.1
= (7 - 3) * 0.1 = 0.4

Among these values, the Diamond suit seems to have the minimum value. Therefore, we can conclude that the diamond suit is the most useless suit, and the lowest ranking card in this suit is selected to be discarded.

Suppose that one of the opponents has bid a Heart suit. The value -0.1 is added to the resulting value of the Heart and new value -0.1 is generated. Now both values for the Diamond suit and the Heart suit are

the same. In this case, the following rules are used :

* if one suit has more cards than the other, then this suit is selected.

* if two suits have the same length, then selection is based on the rank of the suit. The Club suit has the lowest rank, then the Diamond suit, the Heart suit. The Spade suit has highest rank.

According to these rules, the Diamond suit is selected as the most useless suit.

CHAPTER 5

PLAN LANGUAGE

In Chapter 2, we have briefly described how to construct the playing sequence of cards for a technique. A tree is built during the construction. Each of leave nodes represents the basic move of the Bridge game. The top-level nodes represent the major steps to apply a technique. We can say that the construction process is the execution of a plan for applying a technique. The nodes in the next lower-level of a tree represent the more detailed steps of their patrent node. In this chapter, we will discuss a Plan Language used by these plans. At first, we will explain the syntax of the statement defined by this language, and then use the Throw-in technique as an example to show how the statements are used in plans.

The Plan-Language expresses plans of action for the offensive side. In general, the offense wants to have a reply ready for every defensive alternative. A plan cannot be a linear sequence of goals or moves, but must contain conditional branches depending on the opponent's reply. When the offense is on move, a specific play or move is provided by the plan. When the defense is on move, a list of alternative sub-plans for the offense may be given. The actions in plans can be divided into two categories. One category of actions make the specific play or move. Eventually, they generate a linear sequence of moves. The other category contains conditional branches depending on the opponent's move.

5.1 The syntax of the category 1 action statement :

The statements in this category basically consist of an operation qualifier and an operation and its parameters. They are used to specify a move, a subplan, or a status checking. In order to use a plan in multiple times, a nested control statement is used.

Syntax

```
action = (op-qualifier op [ parameters ] [ nest-control-stat ])

nest-control-stat = (WHILE  predicate  [parameters])

parameters  =  (token-1 token-2 .....token-m)
```

Explanations of symbols

op-qualifier : as we explained earlier in this chapter, it tells the control mechanism how to interpret the "operation".

Legal op-qualifiers are

- DO
- PERFORM
- EXAMINE

op : the operation can be either a function name, or a plan name in the next level.

If op-qualifier is

- DO ..... the "op" is the name of a subplan.
- PERFORM .. the "op" is the name of a function which makes a move.
- EXAMINE .. the "op" is the name of a predicate function.

parameters : It includes a list of parameters for invoking "op".

nest-control-stat : It is used to control the usage of a plan. It is optional. It is used only when the operation-qualifier is DO.

5.2 The syntax of the category 2 action statement :

The statements in this category are used to handle a PLAY-AND-WATCH case. The statement consists of information about the card led, the possible opponent's moves, and the response taken by the 3rd-hand player for each opponent's move.

Syntax

```
(make-lead (card suit player)
       ((when opponent-action-1)
            (3rd-hand-action-1   [ parameters ]))
       ((when opponent-action-2)
            (3rd-hand-action-2   [ parameters ]))
       ((when opponent-action-3)
            (3rd-hand-action-3   [ parameters ])) )
```

* MAKE-LEAD : It is an action taken by the card leader.

* Opponent-action : It is the name of a function which handles the second-hand player's move.

  Normally, this opponent's move can be

  a. following a small card (action-1)
  b. covering the lead with a higher-ranking card (action-2)
  c. discarding a card (action-3)

* 3rd-hand-action : It is the name of a sub-plan which is the reply of third-hand player to the opponent's action.

The first element in the statement is the pattern expressing a card led by a player (either declarer or dummy). The 2nd-hand player can only have three possible moves. Therefore, each alternative begins with a template which adequately describes defensive move. When the defense is on move, the alternative whose template matches the move just made by the defense is tried in the search.

5.3 An example :

In the next few paragraphs, we will use the Throw-in technique as an example to explain how the Plan Language is used in plans. The plan used in root level looks like this :

```
(Throw-in-plan
   (Arguments       nil)
   (Plan-level      root-level)
   (Player-relation ((?opponent   is  (opponent-to-draw-in))
                     (?entry      is  (expected-entry))))
   (Suits           nil)
   (Cards           nil)
   (Premises        nil)
   (Actions         ((DO play-all-free-sure-tricks WHILE •
                          having-a-free-sure-trick-in-hand)
                    (EXAMINE  is-correct-entry ?entry)
                    (DO draw-in-a-card ?opponent)
                    (EXAMINE  is-correct-opponent  ?opponent)))
   (Alternatives    nil) )
```

How this plan was constructed will be discussed in the Chapter 7. For the time being, we will only concentrate on attributes "Premises",

"Actions", and "Alternatives". The construction process for the Throw-in technique starts at this plan. There are four elements in "Actions". The element "EXAMINE is-correct-entry ?entry" and the element "EXAMINE is-correct-opponent ?opponent" are functions to check status. For example, the function "is-correct-entry" examines if ?entry is a correct entry. If it is false, then the next element "draw-in-a-card" which is a subplan cannot be executed. Thus, the Throw-in-technique becomes inapplicable. In In Chapter 8, we will explain how to deal with this case when it happens. The other two elements represent subplans in the next lower level. The subplans "play-all-free-sure-tricks" and "draw-in-a-card" are the names of plans in the level next to the root level.

As you can see, the statements used here all belong to the category 1 statements. To see an example of category 2 statements, let's take a look at the subplan "draw-in-a-card". The text shown below is the "Actions" part of the subplan "draw-in-a-card".

```
((MAKE-LEAD (?card ?suit ?lead1)
      ((WHEN play-low)
            (play-a-smaller-card  ?suit  ?3rd-hand))
      ((WHEN play-high)
            (play-a-higher-card ?suit ?3rd-hand))
      ((WHEN discard-card)
            (play-a-smaller-card  ?suit  ?3rd-hand)) )))
```

This statement says for the lead card, if the opponent plays a lower ranking card, then the 3rd-hand player should drop or discard a card. But if the opponent covers the lead with a higher-ranking card, the 3rd-hand player should cover his opponent's play. If the opponent discards a card

in one of the suits other than the suit led, then the 3rd-hand player plays a smaller card in the suit led if he has a card in that suit or discards a card when he doesn't have a card in the suit led.

CHAPTER 6

PROPOSING PLAYING TECHNIQUES

In Chapter 2, we mentioned that our computational approach of planning a play consists of three phases. The first phase is the suit-analysis phase. The second phase is the best-technique selection phase. The third phase is the phase is the card-sequence construction phase. In this chapter, we want to explain the first two phases in more detail. The task the suit-analysis phase is to propose playing techniques for each suit in the Declarer-Dummy's hands. The task of the best-technique selection phase is to select the best technique for each suit and then sort them in terms of their ability to build extra tricks.

6.1 The suit-analysis phase :

The suit-analysis phase is implemented in two stages. The first stage is the pre-analysis stage and the second stage is the primary-analysis stage.

6.1.1 The pre-analysis stage :

This stage of the analysis is to find the technique information which is related to the card holding of a suit in a combined hand. The knowledge used to do analysis is stored in rules. The PREMISES of an analysis rule consists of statements which describe a card pattern

(shape), and its ACTION consists of a list of names which are the keys to access a particular set of technique-information lists in the knowledge base. The rules stored in the knowledge base are organized in terms of distribution of cards in both the Declarer hand and the Dummy hand. For a particular distribution, a rule is found by forward scanning the related rules. At this stage of the analysis only certain cards are checked if they exist in the combined hand. At this moment, we are not concerned about whether a card is in the Declarer's hand or in the Dummy's hand. Let's take a look at the following example :

Example

Suppose the card holding in Club suit is

```
Dummy     : (7 5 2)
Declarer  : (A Q 8 3)
```

By scanning rules which are related to the distribution of 4-3, we find the following rule which describes the Club suit :

```
(Suit-analysis-rule-4-3-0008
  (Premises   (?combined-suit  has  ?1)
              (?combined-suit  has  ?3) ))
  (Actions    ((Propose  ((Tech-info-4-3-002)
                          (Tech-info-4-3-010))))
```

In this example, the rule tells that two technique-information lists stored in the knowledge base are related to the holding held by the Declarer and Dummy.

6.1.2 The primary-analysis stage :

This stage of the analysis is to check techniques suggested by the pre-analysis and find the one which can exactly describe the cards held in the Declarer's hand and the Dummy's hand. In the previous example, two technique-information lists are suggested. Now we need to decide which one describe the cards between the Declarer hand and the Dummy hand. Suppose the technique-information lists retrieved from the knowledge base are :

```
(Tech-Info-4-3-002
  (Player-relation  .....)
  (Max-card-index   ......)
  (Holding         ((?1 ?3 ?x1 ?x2) (?x3 ?x4 ?x5 ?y)))
  (........
          ........) )

(Tech-Info-4-3-010
  (Player-relation  .....)
  (Max-card-index   .....)
  (Holding         ((?1 ?x1 ?x2 ?x3) (?3 ?x4 ?x5 ?y)))
  (.....
             ......) )
```

By examining the values of HOLDING of these two lists, we find the first technique-information list is the one we want.

6.2 The best-technique selection phase :

At the primary-analysis stage, the technique-information list with the name "Tech-info-4-3-002" is selected for the Club suit. The complete list is shown as follows :

```
(Tech-info-4-3-002
  (Player-relations    ((?p1 ?p2) (?right-op ?p1) (?left-op ?p1))
  (Max-card-index      3)
  (Holdings            (((?1 ?3 ?x1 ?x2) (?x3 ?x4 ?x5 ?y))))
  (Sure-tricks         1)
  (Sure-tricks-list    ((?p1 (?1))))
  (Extra-tricks-list   ((Technique-01  1) (Technique-02  1))))
  (Tricks-won-by-length  0.36)
  (Techniques
    (Technique-01
      (Depend-on       ((?right-op  has the ?suit  ?2)))
      (Objective       ((?3 wins 1 trick)))
      (Avoid           nil)
      (Risk            ((?left-op may gain the lead)))
      (Lead            ((?p2))
      (Technique-name ((Finesse  ?3  ?p1))) )
    (Technique-02
      (Depend-on       nil)
      (Objective       ((?3 wins 1 trick)))
      (Avoid           ((?left-op gains the lead)))
      (Risk            ((?right-op may gain the lead)))
      (Lead            ((?p2  ?p1))
      (Technique-name ((Throw-in  ?right-op))) )))
```

There are two techniques available in this list. To select the best technique for the Club suit, we start from the first technique TECHNIQUE-01. By checking the statement embedded in attribute DEPEND-ON of this technique, we can then decide if this technique is feasible. A certainty is used in this checking. If the certainty value is greater than or equal to 0.7, the technique being examined is thought as the best technique for the suit being examined. In our case, the suit being examined is Club. The techniques contained in attribute TECHNIQUES are ordered in terms of the chance to win tricks. In our example, the technique TECHNIQUE-01 has a better chance to win tricks than the technique TECHNIQUE-02. As long as a technique is found to be feasible, the selection stops. If the technique is found to be unfeasible, then the next technique in the list is

examined. The last technique in the list is always used as the default.

CHAPTER 7

THE SCHEME OF CONSTRUCTING CARD SEQUENCE FOR A TECHNIQUE

In this chapter we will discuss how a card sequence is constructed for a technique in detail. We use the throw-in technique as an example to explain all of this. As we mentioned before, the construction for a playing sequence of cards is a process of executing a plan for applying a playing technique. For example, to apply the Throw-in technique, the construction is a process to execute the following plan. During the construction, a tree is implicitly built. The root node of this tree is the is the plan shown below.

```
(Throw-in-plan
   (Arguments        nil)
   (Plan-level       root-level)
   (Player-relation ((?opponent   is  (opponent-to-draw-in))
                     (?entry      is  (expected-entry))))
   (Suits            nil)
   (Cards            nil)
   (Premises         nil)
   (Actions          ((DO play-all-free-sure-tricks WHILE •
                          having-a-free-sure-trick-in-hand)
                     (EXAMINE  is-correct-entry ?entry)
                     (DO draw-in-a-card ?opponent)
                     (EXAMINE is-correct-opponent  ?opponent)))
    (alternatives    nil) )
```

The construction process starts at this node. It checks the statements in PREMISES. If there are statements existed in PREMISES, then each statement is checked to see if it is true. After all these statements

have been checked, a conclusion is drawn. If it is true, then all actions in ACTIONS are pushed onto stack. The first action in the list is on top after the push operation. The stack is used as a temporary storage during the construction. A stack is allocated when a technique is applied. It is deallocated when the stack becomes empty or the status checking during construction indicates that replanning is necessary. The push operation is slightly different from the traditional push operation.

Example

In the above Throw-in plan, after the actions in ACTIONS are pushed onto stack, the stack looks like :

```
Top-of-Stack  -->  (DO play-.......  WHILE .....)
                   (EXAMINE  is-correct-entry  ?entry)
                   (DO draw-in-a-card ?opponent)
                   (EXAMINE  is-correct-opponent ?opponent)
```

Now, the top item in the stack is accessed, the conditional expression of WHILE is first checked. If it posts TRUE, then the name after DO is used to retrieve a subplan which contains the detailed steps of implementing this plan. There is only one action in this subplan and it is pushed onto the stack. So, the stack becomes :

```
Top-of-Stack  --> (DO  play-free-sure-tricks-in-suit ?suit ..)
                  (DO  play-.... WHILE ..)
                  (EXAMINE  is-correct-entry  ?entry)
                  (DO draw-in-a-card  ?opponent)
                  (EXAMINE  is-correct-opponent  ?opponent)
```

The pop operation is standard. Each time it pops out the top element in the stack.

In this example, the plan doesn't have premises. Therefore, the actions in ACTIONS are added into stack. If the plan does have statements in PREMISES and the conclusion of these statements is false, then the actions in ALTERNATIVES are pushed onto the stack. The use of PREMISES, ACTIONS, and ALTERNATIVES is equivalent to the sentence "if premises then actions else alternatives".

Each time the top element in the stack is accessed. In this example, the top element is "DO play-all-free-sure-tricks WHILE having-a-free-sure-trick-in-hand". The operation qualifier DO tells the "play-all-free-sure-tricks" is a subplan. The token WHILE tells the "having-a-free-sure-trick-in-hand" is a predicate. Therefore, this predicate is examined first. If it indicates a true condition, then the subplan "play-all-free-sure-tricks" is retrieved from the knowledge base. The steps we just described will be applied on this new plan.

The new actions are added into stack. The steps are recursively applied until a node which represents a basic move in Bridge is reached. The plan for an offensive move can be one of two things : a basic move or a goal. The basic move is simply the name of a card and a player. Such a plan simply invokes the function "play-this-card" to add the card played into the sequence list. A goal is the name of a plan in the knowledge base followed by a list of parameters. The following plan is a plan which

performs a basic move.

```
(Drop-a-card
  (Arguments        (?suit  ?player))
  (Plan-level       card-level)
  (Player-relation  nil)
  (Suits            (?suit))
  (Cards            (?card  in  (cards-hold-by  ?suit ?player)))
  (Premises         ((is-the-smallest-card  ?card  ?suit  ?player)))
  (Actions          ((PERFORM play-this-card  ?card ?suit ?player)))
  (Alternatives     nil) )
```

Since it is the primitive plan, the function "play-this-card" is performed. And then this plan which is the top element in stack at this moment, is popped out from the stack. The next plan in the stack is then accessed. If the element represents a subplan, then its predicate is checked. If the predicate shows a FALSE, then the element is also removed from the stack. In this example, the node "DO play-all-free-sure-tricks" is removed from the stack as long as its predicate "having-a-free-sure-trick-in-hand" indicates a FALSE condition. If this predicate indicates that there are free sure tricks left in hand, then the plan "play-all-free-sure-tricks" is executed again, and it will be executed repeatly until there are no free sure tricks left in players' hands.

As we mentioned in the earlier chapter, the action can be a statement for performing particular checking. In this example, the statement "EXAMINE is-correct-entry" is the statement which is used to check entry and a particular opponent. If the checking shows the FALSE condition, then a close examination is taken to see if it is possible to go on. For example: if the statement "is-correct-entry" shows the current entry is

an incorrect entry after playing all the free sure tricks, then we need to check to see if the current entry is allowed to be an entry. In this example, the attribute "lead" indicates either dummy or declarer can be an entry. So the next plan "draw-in-a-card" can be performed. If the attribute "lead" shows that a player must be the entry but he is not the current entry, then the plan "setup-entry-at" is pushed onto the stack for the next execution.

The construction process proceeds until there are no elements left in the stack. At the end of construction, a card sequence is built. The construction process stops, and the stack is deallocated. The resulting card sequence consists of at least one element. The sequence has the following form :

```
((card1 suit1  player1) (card2 suit1 player2) .....(G00051))
        |                       |                     |
        V                       V                     V
    element-1               element-2             element-3
```

The element-1 represents a card led and the element-2 represents a a card played by the 3rd-hand player (partner). The element-3 is a node generated to store "actions" in the plan shown below. The token G00051 is a symbol generated by a LISP function "gensym". It does not have any significant meaning in our discussion.

```
(Play-to-finesse
   (Arguments        ......)
   (Plan-level       ......)
```

```
(Player-relation     ......)
(Suits         ........)
(Cards ..........)
(Premises  ........)
(Actions    ((make-lead (......)
                ((when play-low)
                     (play-exact-card  .....  ?3rd-hand))
                ((when play-high)
                     (play-a-higher-card ..... ?3rd-hand))
                ((when discard-card)
                     (play-without-lose  ....  ?3rd-hand)) )))
(alternatives  .....) )
```

In the next chapter, we will discuss how to control the play when this kind of node is encountered.

In the following example, suppose we want to apply the THROW-IN technique to the CLUB suit.

(Dummy - North)

Spade	8	5	2	
Heart	A	5	4	
Diamond	A	K	Q	J
Club	7	5	2	

(Declarer - South)

Spade	K	J	7	
Heart	K	J	6	
Diamond	9	8	4	
Club	A	Q	8	3

If we start with this initial hand, then a playing sequence of cards generated by the construction process is as follows :

```
( ((South  Diamond  4)  (North  Diamond  A))
  ((North  Diamond  K)  (South  Diamond  8))
  ((North  Diamond  Q)  (South  Diamond  9))
  ((North  Diamond  J)  (South  Club     2))
  (G00045) )
```

The node (G00045) has the following value :

```
((Make-lead  (2  Spade  North)
   ((WHEN play-low)     (play-a-smaller-card  Spade  South))
   ((WHEN play-high)    (play-a-higher-card   Spade  South))
   ((WHEN discard-card) (play-a-smaller-card  Spade  South))))
```

We use the stack to implement the construction process. The advantage of this scheme is to make the control easier. Since we put the checking on each stage of planning, so if there is anything wrong, we can stop immediately and backtrack to the previous stage.

CHAPTER 8

THE CONTROL MECHANISMS OF THE BRIDGE PLANNER

8.1 Top-level control mechanism :

It switches the play of the declarer's side between the offensive position and the defensive position. The algorithm of top-level control mechanism is as follows :

```
loop
(set game in game-mode)
(declarer side plays in defensive position until
          he gains the lead or game-is-over)
(if the game is over, then stop game, counting score)
(declarer side plays in the offensive position
          until an opponent gains the lead
          or game-is-over)
(if the game is over, then stop game, counting score)
(otherwise, go loop)
```

The game is played in either the game mode or the simulation mode. When the game is in its game mode, the game proceeds. If the game is in its simulation mode, the construction process is implemented, and the game is paused until the construction process is done and a card sequence is generated. When the game starts, the game is set into game mode because the declarer's side is in the defensive position to play a card against the opponent's open lead. The game proceeds until the declarer's side gains the lead. Whenever the declarer's side gains the lead, the

game is set into the simulation mode and the construction process starts. We have already explained how this process works. So it is unnecessary to repeat it again.

When the construction process is done, the game is set to the game mode and re-start again. This time, the cards played by both the declarer and dummy are retrieved from the sequence list. The game proceeds until there are no cards left in the sequence list. At this point, three situations exist :

a. The game is over
or b. The opponents gain the lead
or c. The declarer's side still gains the lead, but need to perform another run of planning for the next available technique.

In case a, the game is over, and the scores are calculated. In case b, it is the turn for the declarer's side to play the defensive position. In case c, if there is a technique left in the list, then a construction process is performed to form a card sequence. If there are no techniques left, the game enters into the end stage.

8.2 The control mechanism for defensive play

In our study, we do not try to do planning for the opponent's side to play in both the defensive and offensive positions. But we do have two sets of rules for the declarer's side to play in the defensive position. the control mechanism is very simple. If the declarer's side is in the

defensive position, then the rules are scanned to determine what card should be played. These two sets of rules are listed in Appendix A.

```
loop
(human player leads a card)
(the second-hand player plays a card)
(human player plays a card)
(the fourth-hand player plays a card)
(if the game is over, stop game, counting scores)
(determine who is the winner to lead for next run)
(if either the declarer is a winner,
    or the dummy is a winner, then exit)
(go loop)
```

One important issue of playing the defensive position by the declarer is: techniques selected may become inapplicable because the vital cards used by the techniques might be played during the defense. Therefore, during the defensive play, for each card played by the opponents, the techniques in the list must be examined to see if the requirements of the technique are changed. If they were changed, then the technique will be withdrawn from the list. Later, the replanning on the related suit is necessary to be performed.

Another case may be encountered during the defensive play. That is : the card led by one of the opponents is the one expected by the technique applied on the suit led. Therefore when it happens, the number of extra tricks to be built by that technique is re-calculated. If after this update, the number of extra tricks becomes zero, then the technique is removed from the list. If there is at least one extra trick left to be

built, then the replanning will be taken for this suit because the state of suit has changed since the technique is decided.

8.3 The control mechanism for offensive play :

In the previous chapter, we mentioned how to construct a card playing sequence. The task of this mechanism is to implement the construction process. It controls the expansion and shrinkage of the stack. It also performs a special task when exceptional cases are encountered. For example : if the current lead is not the one expected by the action taken, then a close examination is performed.

The control is divided into two levels, the level 1 control is used to control the game to play in either game mode or simulation mode and to perform as an interface to the top-level control mechanism. The level 2 control is used to retrieve plans and interpret these plans and eventually generate a card sequence.

The algorithm of Level 1 control mechanism

```
(making up plans)
loop
(get a plan)
(starts at a suit)
loop1
(apply its technique)
(check status)
(if something went wrong,  if there is a plan left
                               go loop
                           otherwise, perform replanning)
(if game proceeds, play all cards in sequence list)
```

```
(if opponents gain the lead, plays in defensive position
 otherwise,  try next technique in plan
             if it is still applicable
             otherwise, perform replanning)
(go loop1)
```

The algorithm of level 2 control mechanism

```
(get suits list)
loop
(if there is a suit in the suits list
    check each suit in the suits list
    (if plan level of the suit is suit-level
     then go loop1
     else
       get cards for the selected suit)
loop1
(examine premises)
(if the conclusions of the premises is TRUE
 then   push subplans in "actions" onto stack
 else
        push subplans in "alternatives" onto stack)
(if plan level is suit level then  go loop
 else  go loop1)
```

8.4 The control mechanism for defensive play against opponent's move :

This level of control mechanism is used to control the play which may be intervened by the 2nd-hand opponent. For the Throw-in technique, the construction process generates a card sequence which looks like :

```
((card suit player) (....)  (....)   (G0035))
```

The element (G0035) in this list is a node which stores the lead and the opponents' possible actions.

The node of (G0035) has 4 attributes :

* lead : It stores the card to be led
* opponent-play-low : It stores the action should be taken by third-hand player when the second hand player(opponent) drops a lower ranking card.
* opponent-play-high : It stores the action should be taken by third-hand player when the second hand player(opponent) cover lead with a higher ranking card
* opponent-discard : It stores the action should be taken by third-hand player when the second hand player discards a card in a side suit.

During the offensive play, the card is retrieved from the card playing sequence generated by the construction process. If the element in the list is an atom (we use list to store a card), then it is recognized as a node like (G0035). The card stored in LEAD of this node is led. After the 2nd-hand opponent plays a card, we can then check which of the above templates is matched. For example: if the 2nd-hand opponent plays a card whose ranking is lower than the card led, then the template "opponent-play-low" is matched, and the 3rd-hand's action stored for this template is taken.

8.5 The control mechanism used at the end stage of the game :

Two plans are used in the end-game play. They are

* Play-sure-tricks-left plan
* Play-and-watch plan

First, the sure tricks left in hand will be played. Then a lower-

ranking card is selected to lead. In the earlier chapters, we explained how to evaluate the threat of an opponent on the card you lead. The same technique is applied here to decide which card should be played without risks.

When there are no cards left in anyone's hand, the game is over. The status "end-of-game" is returned to the top-level control mechanism.

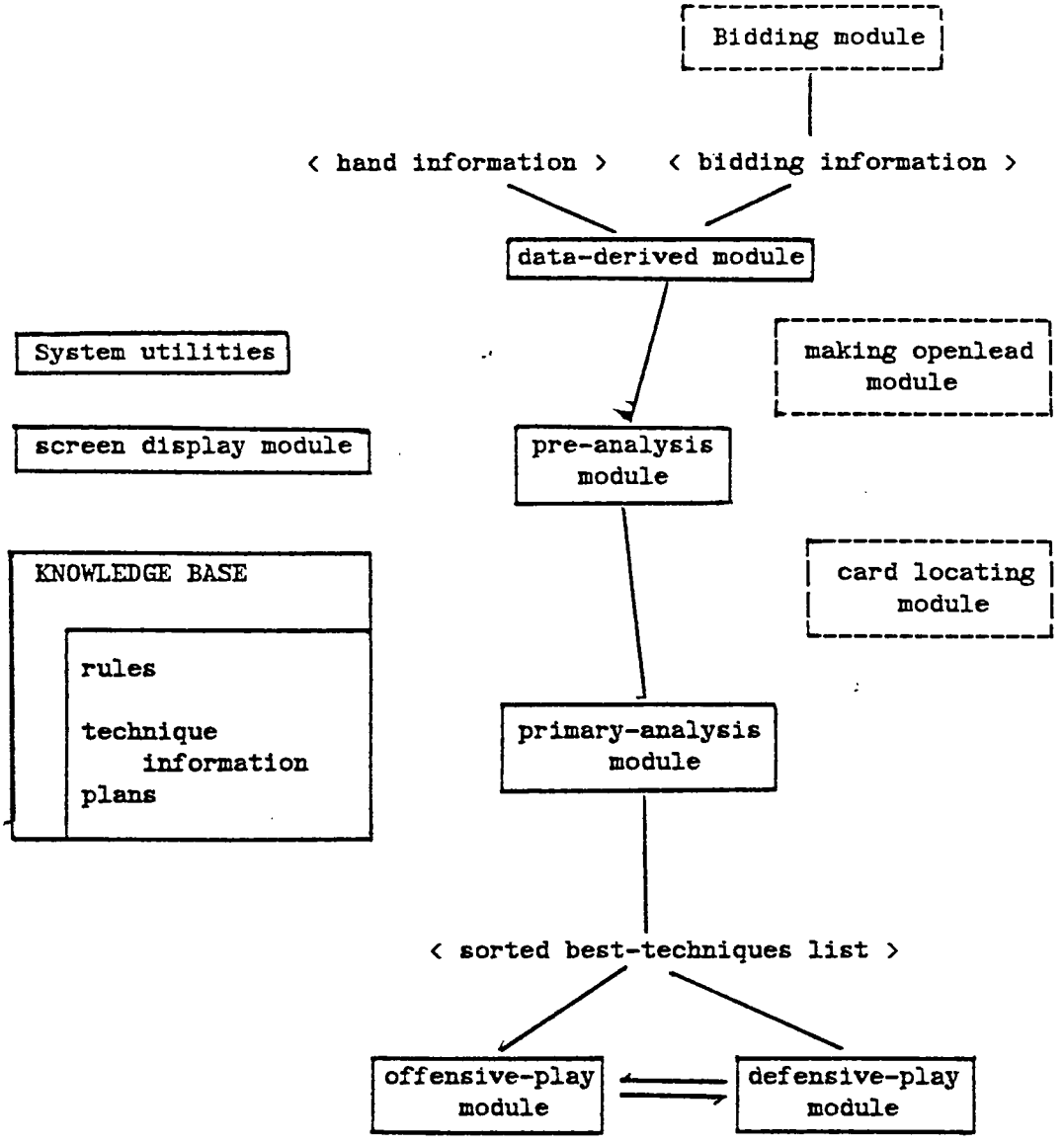
The algorithm of end-game control mechanism

```
loop
(if game-is-over, return to top-level control)
(if there are sure tricks in hand
 then plays all sure tricks)
loop1
(if game-is-over, return to top-level control)
 otherwise
     (the declarer plays a lower-ranking card)
     (if opponent gains the lead
      then the declarer side plays as defenders)
     (if delcarer side gains the lead again
      then go  loop
      otherwise   go  loop1)
```

CHAPTER 9

PROGRAM IMPLEMENTATION

In earlier chapters, we discussed how the evaluation technique is used in the suit-analysis phase to find the contribution of each suit and techniques the suit can use. We have also discussed how the best technique is selected for a particular suit. We discussed how to use these techniques found to build an offensive plan and thereafter construct the card sequence for each of the techniques in this plan. The program developed for implementing this planning process was written in IQLISP [9]. and the testing was performed on a IBM PC AT microcomputer. The program we developed is part of a Bridge-playing program which consists of several modules outlined below. For those modules which were developed in our study, we use the solid-border boxes to circle them. The exceptions are : our offensive-play module only performs planning for the offensive play made by the declarer's side. No planning is performed for the offensive play made by the opponents. The boxes used in the following chart have either the solid border or the dashed border. The use of dashed-border boxes indicates that no programs are available at this moment. The rules, the technique information and the plans constitute our knowledge base.



Now, we discuss the functions performed by the modules which are circled with the solid-border boxes.

A. The system utilities module :

The module contains the general purpose utilities, the infer-

ence engine, and the program for performing the variable bindings. The control mechanism used in our inference engine is a forward chainer.

B. The screen display module :

The module is used to display the current state of the game on the screen.

C. The data-derived module :

The module is used to derive the hand information and the bidding information into the system-defined statements described in Chapter 3.

D. The pre-analysis module :

The module is used to perform the pre-analysis on the suits held by the declarer's side.

E. The primary-analysis module :

The module is used to perform the detailed analysis on the result generated by the pre-analysis module.

F. The offensive-play module :

This is the main module of the bridge planner we built. It

controls the process of the game. When the game is in the game mode, it plays as the declarer's side. When the game is in the simulation mode, it performs the card-sequence construction for a technique being applied. It switches the declarer's play in between the offensive position and the defensive position.

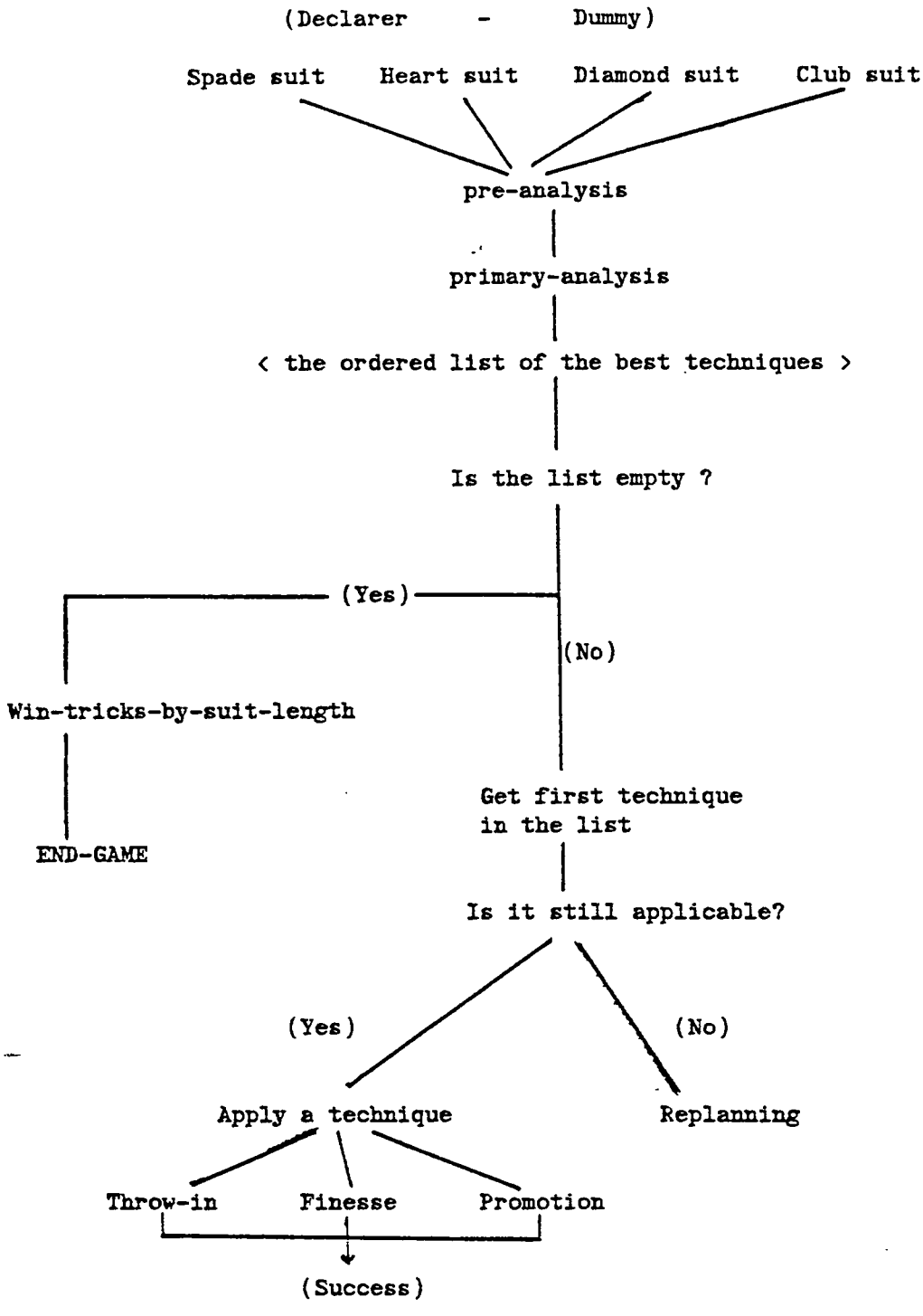
G. The defensive-play module :

This is the module to perform the defensive play for both sides.

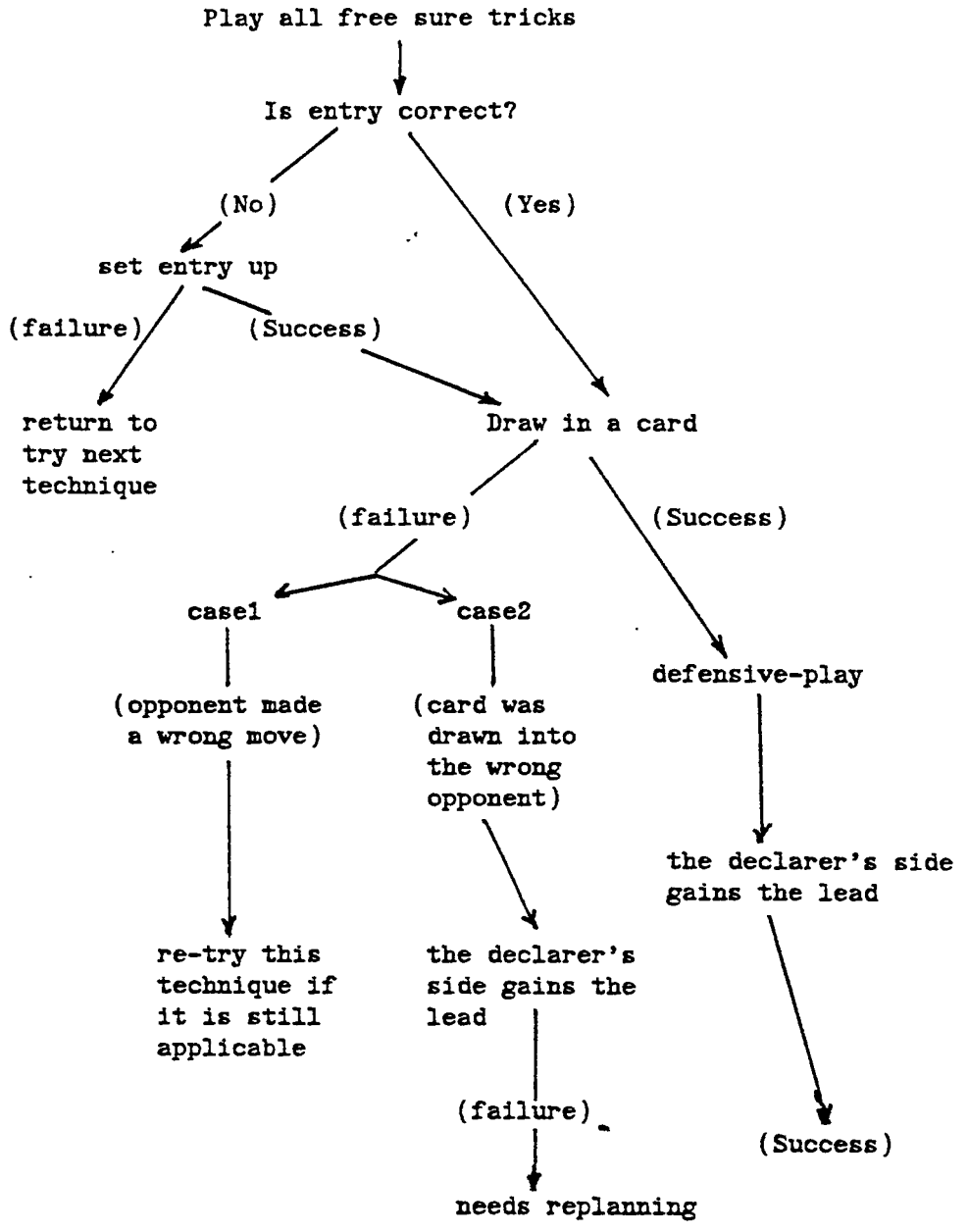
Right now, our Bridge planner can handle the following techniques : the Finesse technique, the Throw-in technique, the Promotion technique, and the Win-tricks-by-suit-length technique. In addition to these four techniques, both Play-all-sure-tricks-left plan and Play-and-watch plan are the plans used at the end-stage of a game.

Our Bridge planner allows the expansion in the future. The new techniques can be added at any time by following our technique-setup procedure. Before we discuss this setup procedure and the testing steps after adding the new technique, we would like to describe the control flow of our bridge planner. It gives you more clear idea about the system implementation and helps you to install the new techniques into the system.

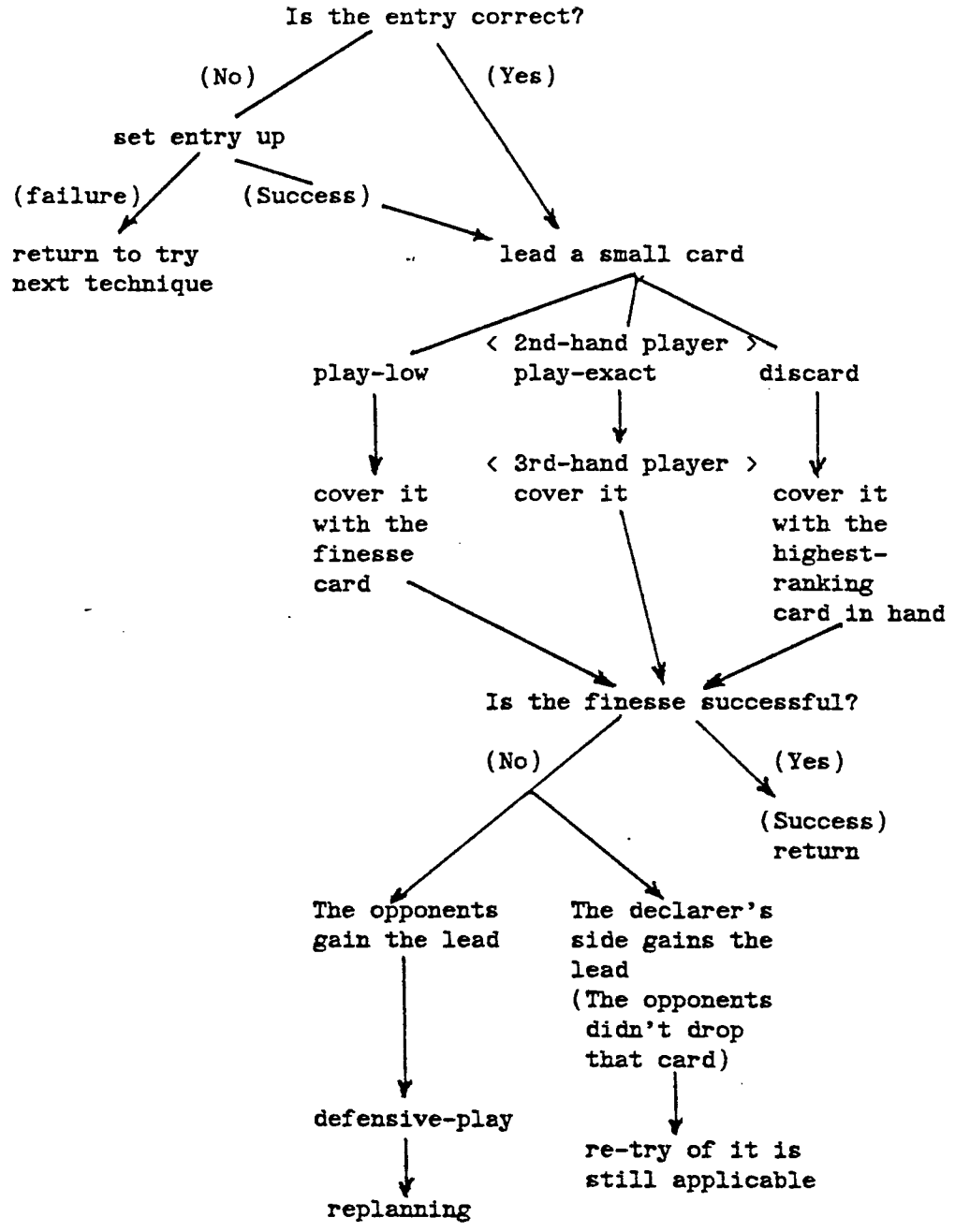
BRIDGE-PLAYING CONTROL FLOW



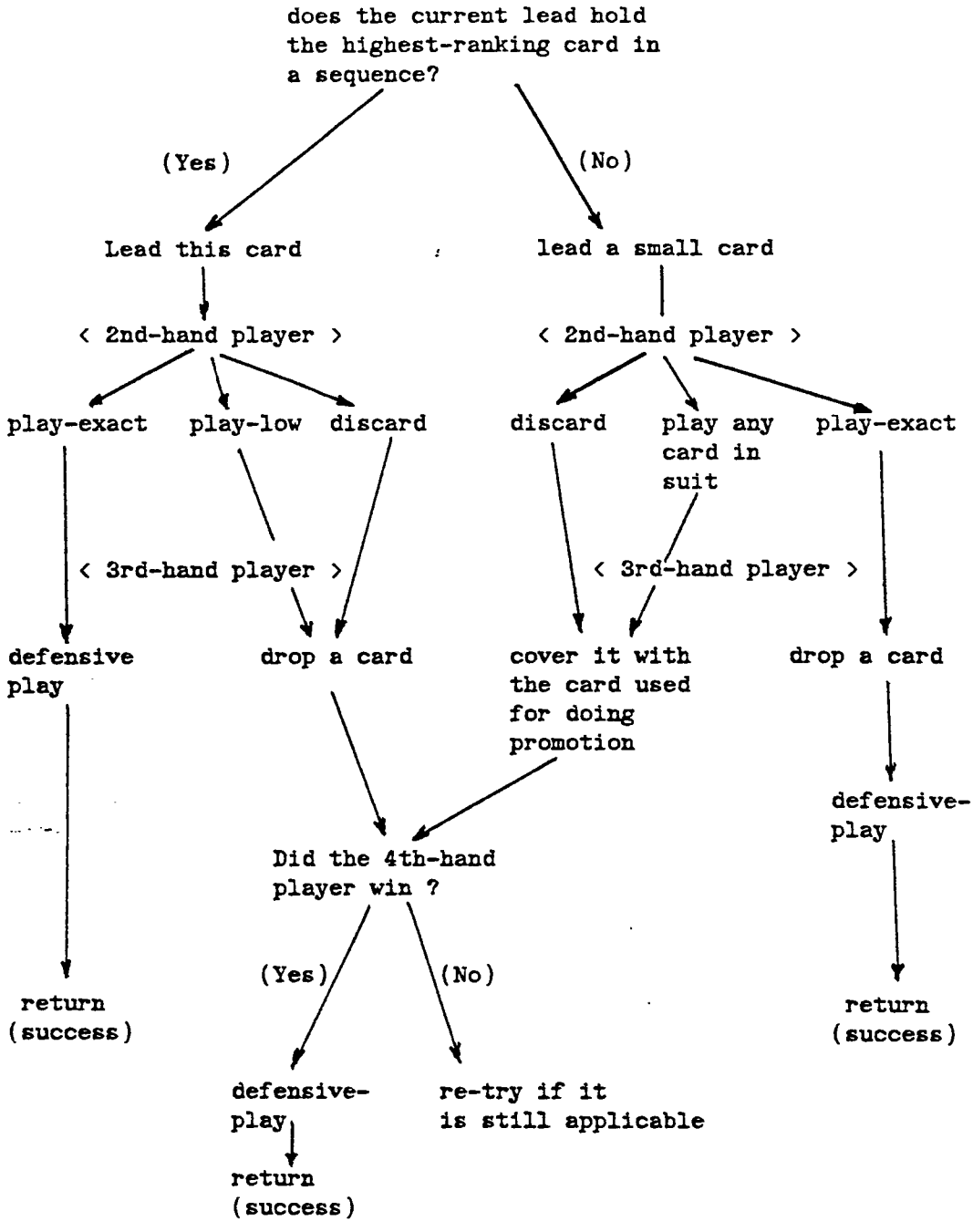
CONTROL FLOW OF THE THROW-IN TECHNIQUE



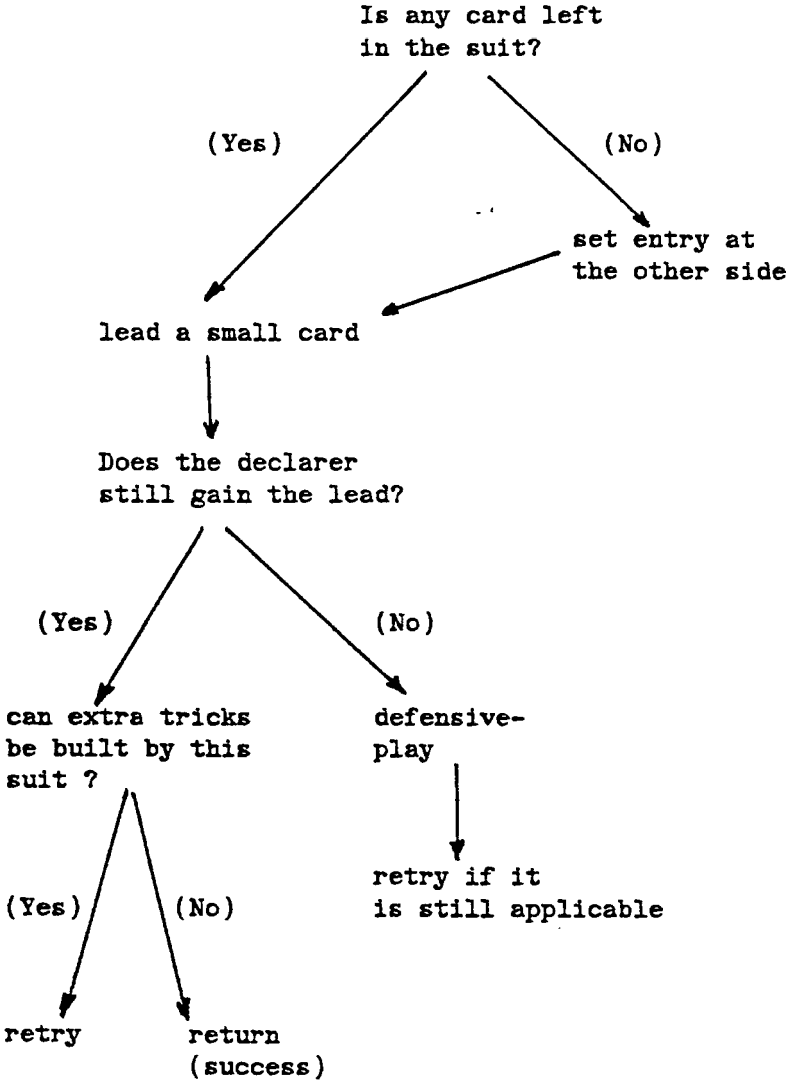
CONTROL FLOW OF THE FINESSE TECHNIQUE



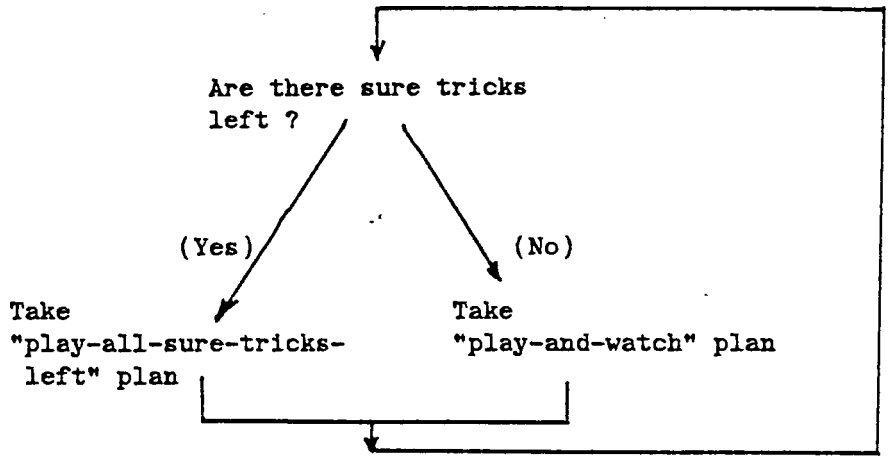
CONTROL FLOW OF THE PROMOTION TECHNIQUE



CONTROL FLOW OF
THE "WIN-TRICKS-BY-SUIT-LENGTH" TECHNIQUE



CONTROL FLOW OF THE END-GAME



The Play-all-sure-tricks-left plan is used to play all the sure tricks left in the declarer's side. And the Play-and-watch plan is used to switch the declarer's play into the defensive position. As the defenders, the declarer's side can watch the plays made by the opponents to see if there is still a chance to establish sure tricks.

The following text describes the setup procedure and the testing steps for a new playing technique :

Step 1 - Assigns a name to this new technique. The name is an id. of this new technique.

Step 2 - Constructing the root-level plan for this new technique. Starting at this plan, you then build the lower-level plans. You can use those existing plans stored in the knowledge base.

Since different technique needs different control, the function used for status checking may be different. The top-level control mechanism of the planner must be modified so that it can handle the exception properly.

Step 3 - Testing new technique

a. Prepare an example which needs to use this technique. Such an example must include the information of four hands and the bidding information.

b. Run data-derived module to derive the information of four hands and the bidding information into a form described in Chapter 5. The result is stored in a temporary file.

c. Run pre-analysis module. The input to this module is the data generated by the data-derived module. To make the pre-analysis work, for each suit in the declarer's side, there must be a suit-analysis rule whose premises describe the card-pattern held in that suit. The names stored in action part of this rule are checked to see if they exist. If a name doesn't exist, you need to build a technique source for this name. The result generated by this module is stored a temporary file.

d. Run offensive-play module. This module first invokes the primary-analysis module, and then invoke the defensive-play module. The input to this module is the data generated by the pre-analysis module. The result is a list of best techniques and their relevant information. During the execution of the primary-analysis module, few questions are needed to give the answers. During the defensive play, you need to watch each play played by either the declarer's side or the opponents to see if the play is as you expect. If it is not, the rules used for the defensive play are examined and corrected.

e. You can trace the card-sequence construction process by turning the trace on. When trace is on, certain information are displayed on the screen. By viewing these information, you can see if the planning works as you expect. If it is not, then you need to check plans you build or check to see if anything is missing in the control mechanism.

> For the attributes "suits", "cards" and "player-relation" in a plan you build, you must make sure that the values provided by the function used are correct. For the attributes "actions" and "alternatives", each element appearing in the list must be checked to see if it exists or if it performs the correct checking.
>
> The last thing you need to check is : each variable used in a plan must be defined in the binding program.

The current version of the Bridge planner was developed on an IBM PC AT microcomputer by using IQLISP interpreter. The program for generating the derived predicates to describe facts takes about 730 lines. The program for performing suit-analysis takes about 1175 lines. The program for selecting a best technique and constructing a card sequence takes about 2055 lines. And the knowledge base contains 177 analysis rules, 8 rules for the second-hand player playing in the defensive position and 6 rules for the fourth-hand player playing in defensive position, and about 20 lists storing technique-information, and 30 plans for constructing a card sequence. The number of analysis rules will be increased if all the possible distributions of cards are taken into consideration. The number of plans and the number of technique-information lists will be increased if a new playing technique is added. The program size will be increased tremendously if the planning is also applied to the defensive play.

CHAPTER 10

AN EXAMPLE

Suppose the declarer is in the contract of 1NT. West, the left-hand opponent of the declarer, leads with the 7 of the Spade. Before the analysis step takes place, a conversion is performed on the following hand using the data-derived module. This conversion transfers the initial information of a hand and the bidding into the form described in Chapter 3.

```
(North  (Spade      (6 5 2))
        (Heart      (A 3 2))
        (Diamond    (9 7 4))
        (Club       (K 6 4 3)))
```

West openleads
the Spade 7

```
(South  (Spade      (A K 4))
        (Heart      (8 6 4))
        (Diamond    (A Q 3))
        (Club       (A 8 5 2)))
```

The bidding is recorded as follows :

```
((west  (0  pass)) (north (0  pass))
 (east  (0  pass)) (south (1  nt))
 (west  (0  pass)) (north (0  pass))
 (east  (0 pass)))
```

After the conversion is done, the analysis process starts. It first finds the names of knowledges about techniques for each suit by searching through the analysis rules. In this example, the analysis rules found

will be

```
(suit-analysis-3-3-002
     (premises ((?combined-suit has ?1)
                (?combined-suit has ?2)))
     (actions  ((plans-to-be-examined  ((plan-3-3-002)
                                        (plan-3-3-003))) )))


(suit-analysis-3-3-004
     (premises ((?combined-suit has ?1)))
     (actions  ((plans-to-be-examined  ((plan-3-3-005))) )))


(suit-analysis-3-3-003
     (premises ((?combined-suit has ?1)
                (?combined-suit has ?3)))
     (actions  ((plans-to-be-examined  ((plan-3-3-004))) )))


(suit-analysis-4-4-005
     (premises ((?combined-suit has ?1)
                (?combined-suit has ?2)))
     (actions  ((plans-to-be-examined  ((plan-4-4-001))) )))
```

The state of the Spade suit satisfies the descriptions of premises of the rule "suit-analysis-3-3-002". This rule suggests two plans stored in the knowledge base are related to the card-holding in the Spade suit. Let's look at these two plans.

```
(Tech-info-3-3-002
  (Player-relations    ((?p1 ?p2) (?left-op  ?p2) (?right-op ?p2)))
  (Max-card-index      2)
  (Holdings            (((?1 ?x1 ?x2) (?2 ?x3 ?x4))))
  (Sure-tricks         2)
  (Sure-tricks-list    ((?p1 (?1)) (?p2 (?2))))
  (Extra-tricks-list   nil)
  (Tricks-won-by-length  0)
  (Techniques          nil) )
```

```
(Tech-info-3-3-003
  (Player-relations    ((?p1 ?p2) (?left-op  ?p2) (?right-op ?p2)))
  (Max-card-index      2)
  (Holdings            (((?1 ?2 ?x1) (?x2 ?x3 ?x4))))
  (Sure-tricks         2)
  (Sure-tricks-list    ((?p1 (?1 ?2))))
  (Extra-tricks-list   nil)
  (Tricks-won-by-length  0)
  (Techniques          nil) )
```

By examining the "holdings" in each plan, the card holdings in the Spade suit match the holdings described in Tech-Info-3-3-003 list. Therefore, the Tech-Info-3-3-002 list is discarded. At this point, four technique-information lists are found to relate to four suits in hand.

```
The Spade    suit  :   Tech-Info-3-3-003
The Heart    suit  :   Tech-Info-3-3-005
The Diamond  suit  :   Tech-Info-3-3-004
The Club     suit  :   Tech-Info-4-4-004
```

The last job to be done by the suit analysis module is storing of the information listed above into a temporary file. The content of this file in this example looks like :

```
(Spade suit)
-------------------------------------------------------------------------
Bindings : ((?1 A) (?2 K) (?X1 4) (?X2 6) (?X3 5) (?X4 2)
            (?SUIT SPADE) (?P1 SOUTH)
            (?P2 NORTH) (?LEFT-OP EAST) (?RIGHT-OP WEST))

Number of sure tricks          :  2

Sure tricks hold               :  ((?P1 (?1 ?2)))

Techniques for
building extra tricks          :  NIL
Extra tricks built by length   :  0

Techniques list                :  (NIL)
```

```
(Heart suit)
-------------------------------------------------------------------------
Bindings : ((?1 A) (?X1 3) (?X2 2) (?X3 8) (?X4 6) (?X5 4)
            (?SUIT HEART) (?P1 NORTH) (?P2 SOUTH) (?LEFT-OP WEST)
            (?RIGHT-OP EAST))

Number of sure tricks         :  1

Sure tricks hold              :  ((?P1 (?1)))

Techniques for
building extra tricks         :  NIL

Extra tricks built by length  :  0

Techniques list               :  (NIL)


(Diamond suit)
-------------------------------------------------------------------------
Bindings : ((?1 A) (?2 K) (?3 Q) (?X1 3) (?X2 9) (?X3 7) (?X4 4)
            (?SUIT DIAMOND) (?P1 SOUTH) (?P2 NORTH) (?LEFT-OP WEST)
            (?RIGHT-OP EAST))

Number of sure tricks         :  1

Sure tricks hold              :  ((?P1 (?1)))

Techniques for
building extra tricks         :  ((TECHNIQUE-01 1) (TECHNIQUE-02 1))

Extra tricks built by length  :  0

Techniques list               :

      ((TECHNIQUE-01 (DEPEND-ON ((?RIGHT-OP HAS THE ?SUIT ?2)))
          (OBJECTIVE (?3 WINS TRICK))
          (AVOID NIL)
          (RISK ((?LEFT-OP GAINS THE LEAD)))
          (LEAD (?P1))
          (TECHNIQUE-NAME ((FINESSE ?3 ?P1))))
       (TECHNIQUE-02 (DEPEND-ON NIL)
          (OBJECTIVE (?3 WINS TRICK))
          (AVOID ((?RIGHT-OP GAINS THE LEAD)))
          (RISK ((?LEFT-OP GAINS THE LEAD)))
          (LEAD (?P2 ?P1))
          (TECHNIQUE-NAME ((THROW-IN ?LEFT-OP)))))
```

```
(Club suit)
-------------------------------------------------------------------
Bindings: ((?1 A) (?2 K) (?X1 8) (?X2 5) (?X3 2) (?X4 6) (?X5 4)
           (?X6 3) (?SUIT CLUB) (?P1 SOUTH) (?P2 NORTH)
           (?LEFT-OP EAST) (?RIGHT-OP WEST))

Number of sure tricks          : 2

Sure tricks hold               : ((?P1 (?1)) (?P2 (?2)))

Technique for
building extra tricks          : NIL

Extra tricks built by length   : 0.68

Techniques list                : (NIL)
-------------------------------------------------------------------
```

The "bindings" is a list of variables cross-reference. For example, for the Club suit, the variable ?p1 appeared in "sure-tricks-hold" can find the corresponding value SOUTH in "bindings".

From the information listed above, we know only the Diamond suit can build one extra trick by applying either the Finesse technique or the Throw-in technique. It is possible to build 0.68 tricks in the Club suit. In the Chapter 3, we have given the reason why we separate the extra tricks built by length from tricks built by other techniques.

There are no extra tricks that can be built in both the Spade suit and Heart suit. For the Diamond suit, we need to decide which technique is the most suitable technique for the current situation. The first technique in the techniques list is checked. After the binding, the statement in attribute "depend-on" becomes

((EAST HAS THE DIAMOND K)))

You can see that the variable ?right-op is replaced with EAST; the variable ?suit is replaced with DIAMOND; and the variable ?2 is replaced with K. The best technique selection module then generates a question according to this statement. The question is :

"What do you believe that EAST is likely to have the Diamond King?"

The module expects the card-locating scheme to return an answer for this question. In this case, the expected answer is a certainty value between 0.0 and 1.0. If the answer is less than 0.5, then the Finesse technique is not suitable to play with. Since the Throw-in technique is the default technique, it will be selected as the best technique for playing Diamonds.

Now suppose the answer indicates the Finesse technique has a greater chance to win 1 extra trick. Thus, the Finesse technique is selected as the best. Since Diamond is the only suit to build the extra trick(as long as the best technique is selected for the Diamond suit, the plan is constructed. In this example, the plan only has one technique needed to be applied.

After the best technique is selected, the declarer needs to play in the defensive position against the open lead. If the opponent makes a very stupid move during this play (for example, playing the King of the Diamond, there is no need to use the finesse technique. Since the Finesse

technique is the only one technique in plan, thus the game is brought into the end-game stage.

To apply the Finesse technique, the root-level plan for the finesse technique is retrieved from the knowledge base. Using this plan as the root, a game tree is eventually built. This root-level plan may come up by either one of the subplans listed below.

```
Plan 1  --  (North is the current lead)
            North leads a small card in Diamond suit
            if East plays the King of the Diamond then
               South covers it with the Ace of the Diamond
            else
            if East plays one of the other Diamonds then
               South covers it with the Queen if the Diamond
            else (* EAST has void Diamond suit *)
               South covers it with the Ace of the Diamond

Plan 2  -- (South is the current lead)
           Set up an entry at North
           if entry is setup successfully then
              perform plan 1.
```

If it is plan 1, the construction process will generate a card sequence as follows :

```
((G00045))
```

In Chapter 8, we explained this kind of representation. This is the node generated by using a LISP function "gensym". The number shown here does not show any importance to our job. It only expresses that it is not the regular card presentation. The node stores the value of the ACTIONS

in the following plan :

```
(Play-to-finesse
  (Arguments         (?card-led  ?suit  ?lead1  ?card))
  (Plan-level        card-level)
  (Player-relation   ((?3rd-hand  is  (third-hand-player   ?lead1))))
  (Suits             (?suit))
  (Cards             (?card-led))
  (Premises          nil)
  (Actions           ((make-lead (?card-led ?suit ?lead1)
                      ((when play-low)
                          (play-exact-card  ?card ?suit  ?3rd-hand))
                      ((when play-high)
                          (play-a-higher-card ?suit ?3rd-hand))
                      ((when discard-card)
                          (play-without-lose  ?suit  ?3rd-hand)) )))
  (Alternatives       nil) )
```

The value stored in "lead" of this node is retrieved and led. Depending on what card is played by the right-hand opponent, the value stored in the corresponding attribute of the node is retrieved. The value retrieved represents the action taken by the 3rd-hand player.

If plan 2 is taken, the card sequence generated looks like:

```
((card1) (card2) (G00045))
```

The card1 is a lower-ranking card in one of the suits other than the Diamond played by the South. The card2 is a sure trick in the suit led played by the North. The card1 and card2 are generated by the plan "setup-entry-at" if the plan is executed successfully.

After the sequence is played, if the King of the Diamond is trapped,

since there is no technique left in the plan, the suits are examined to see if the Win-tricks-by-long-suit technique can be applied. In our example, the Club suit can built extra tricks by using this technique. So, the Win-tricks-by-long-suit technique is applied. The technique is used repeatly until the expected number of extra tricks is won, then the game is brought into the end-game stage. In this stage, the Play-all-sure-tricks-left plan is executed. The card sequence generated by this plan is as follows :

((card1) (card2) (card3) (card4).....(card n))

These cards are played. If the game is not over yet, the Play-and-watch plan is executed and again the card sequence generated only has one element.

((G00067))

The Play-and-watch plan invokes the subplan "lead-an-useless-card" shown below.

```
(Lead-an-useless-card
    (Arguments        nil)
    (Plan-level       card-level)
    (Player-relation ((?curr-lead  is  (current-lead))
                      (?3rd-hand   is
                         (third-hand-player  ?curr-lead))))
    (Suits           (?suit in
                       (a-suit-played-without-big-lose ?curr-lead)))
    (Cards           (?card in
                       (the-smallest-card-in-suit ?suit ?curr-lead)))
    (Premises         nil)
    (Actions         ((make-lead (?card ?suit ?curr-lead)
                      ((when play-low)
                          (play-a-smaller-card  ?suit  ?3rd-hand))
                      ((when play-high)
```

```
                    (play-a-higher-card ?suit ?3rd-hand))
                ((when discard-card)
                    (play-a-smaller-card ?suit ?3rd-hand)))))
    (Alternatives    nil) )
```

For this lead, if the opponents are wise, one of them shall win this trick. In this case, the declarer's side needs to play in the defensive position. The defensive module therefore is invoked.

The case we just described is when everything is under control. Suppose the Finesse technique is not successful. If the Finesse technique fails to be executed, there are two possible situations to consider.

Situation 1 - the king is not trapped but the declarer still gains the lead.

Situation 2 - the King is not trapped. Instead, It was played by the West to cover the Queen played by the South.

In situation 1, the Finesse technique is only one technique used, so the replanning process is performed.

In situation 2, since the opponents gain the lead, the declarer must play as defender. When the declarer re-gains the lead, the replanning process is performed.

CHAPTER 11

CONCLUSIONS

Our study on planning in card games has made several contributions. First, we provide a knowledge presentation framework for plans and rules. Second, we devise a scheme to find an applicable playing technique and a scheme to determine the best applicable playing technique. Third, we devise a stack scheme to implement the construction of a card sequence for a selected playing technique and the algorithm to control the hierarchical planning. The approach has been validated by testing several examples.

In the analysis stage, our approach uses the number of extra tricks as the decision factor for selecting a technique to apply. An evaluation technique is provided in order to calculate the number of extra tricks which can be built by a playing technique. The use of this quantity as a decision factor has two advantages. One of the advantages is to simplify the selection of a technique. The other advantage is that this approach approximates the reasoning process of a Bridge expert.

Another feature of our approach is the use of a certainty factor. By using this factor in the selection of a technique for a suit, we can simulate a human player guessing on the location of a missing high card, thus choosing a technique which can lead to a success. By combining this

factor and the number of extra tricks, we can resolve conflict between two techniques. For example, the Spade suit can make one extra trick by using the finesse technique and the Heart suit can make one extra trick by using the promotion technique. To decide which suit should be played first, we must ask "Which technique provides a higher chance to win one extra trick?". In this case, we would like to apply the promotion first because this technique is usually safer than the finesse technique.

In the card-playing sequence construction stage, our approach uses the plans to construct a card-playing sequence for a selected technique. In order to build these plans, we defined a Plan Language. The plans are grouped in terms of techniques. The control mechanism was defined independently from the plans it uses. This feature makes the future expansion to be possible and easier.

There are several problems with our approach. The first problem is the use of the derived hand information and bidding information. In order to make the scanning of the rules efficient, we need to derive the hand information and the bidding information into the form that the rules can use. Currently our program can perform this job. But it is very inefficient because the way we use is very straightforward. To resolve this problem, we need to define several primitive statements. Based on these statements, more complex statements are built. For example, we can define three statements to describe the Spades (A J 10). They are (The Spade has the card Ace), (The Spade has the card Jack) and (The Spade has the card

10). By using these statements, we can build the following statement :

(The Spade has cards Ace Jack 10)

The second problem is the way we propose techniques for a suit. There are many combination of the thirteen cards. In our approach, for each combination, there must be a technique found in the knowledge base. You can imagine that the size of the knowledge base is very huge if we prepare techniques for all the combinations. When the size of the knowledge base grows, the execution of the system will get slow unless we have a good knowledge base management system. The resolution is : instead of building a knowledge base in terms of card combination, we need to organize the knowledges in terms of techniques. For each technique there is a proposer which suggests an appropriate way to apply the technique if it finds that the technique can be applied.

The third problem is the lack of card locating scheme and the bidding module. The card locating scheme draws deductions on each card played by the opponents and the bids made by the opponents. It also provides the answer of a card location to our program. Without this scheme we can not really test our approach.

The fourth problem with our approach is the hierarchical planning. The hierarchical planning only allows that the switching from one technique to another is performed at root-level nodes of a plan tree. For the complicate hand, the planning will become inefficient by using this approach.

The fifth problem with our approach is the lack of ability to make decision of whether or not to stop current technique and try another.

Example
=======

Suppose the Diamond suit can build several tricks by using Promotion technique and the Club suit can build less tricks by using the Win-tricks-by-long-suit technique. If you first play the KING of the Diamond to drive out the opponent's Ace and unfortunately the Ace wasn't played. What are you going to do next ?

Keep trying the Promotion technique?

or

Switching to the Club suit?

Currently, our bridge planner only performs planning on the declarer's side playing in the offensive position. Future works may include the planning on the defensive side. Also by making some changes, our approach can be used for a trump contract.

In planning a No trump contract, it is advisable to first count your winners. The same is true in a suit contract, but now it is equally important to count losers too. Let's look at the following example :

Spade	K	9	7	
Heart	K	Q	7	2
Diamond	8	5	3	2
Club	A	4		

West led
Diamond Queen

Spade	Q	J	10	8	6
Heart	A	4			
Diamond	A	7	4		
Club	10	8	6		

South is the declarer at a contract of 4S, and West leads with the Queen of Diamonds. The declarer observes that he has nine winners in the shape of four trump tricks, three Hearts, and two Aces. A club can be ruffed in dummy to round out ten tricks. A Club must be conceded before one can be ruffed, and this must be done before trumps are played. Otherwise, the defenders may draw dummy's trumps and leave South a trick short.

Thus, the declarer's count of winners is eminently satisfying, but before proceeding he should also count his losers. If he wins with the Diamond opening and plays the Ace and another Club, the defenders will cash two Diamond tricks and he will wind up with four losers. To prevent this, the declarer's side first move is to cash the A K Q of Hearts, divesting himself of a losing Diamond.

We can build a planner for a trump contract based on our planner built for a No trump contract. This time the evaluation technique needs to put losers into the consideration. The analysis steps and the steps of constructing a card-playing sequence remain the same. There are many techniques which can be used by a trump contract. The knowledge of these playing techniques needs to be added into the existing knowledge base. The plans for constructing a card-playing sequence for each of these techniques must be added into the knowledge base.

The control mechanism used here basically can be used by a planner dealing with a trump contract. As we mentioned before, our control mecha-

nism is designed independently for the plans it uses. It is possible to build a common control mechanism for all techniques. The program using this control mechanism will be the final version of the bridge-playing program. For the first version of the program, we still need to modify it to accommodate the requirements of the new techniques. Different techniques have different controls. These controls are based on the result returned from the function which performs the status checking. Therefore, for the new technique we need to add the new status flags into the control mechanism so that the flow of the control can be performed correctly.

In Chapter 9, we explained how to add a new technique into the system and how to test it. These steps can be used when a planner for a trump contract is built.

We still have a long way to go to build a complete bridge-playing program. As we mentioned in Chapter 1, the intention of our work is to build a prototype of the bridge planner. Although there are several problems with our approach, we have suggested the possible solutions for these problems. The program given here is well organized whereas the replacement of a particular module is possible as long as the new module doesn't violate our input/output design. We hope this work can be a good start for those who are interested in this field of study.

APPENDIX A

RULES FOR DECLARER SIDE PLAYING IN DEFENSIVE POSITION

A.1 Rules for second-hand player :

```
(second-hand-rule-001
  (premises ((no-card-to-cover  second-hand)))
  (actions  ((defensive-play second-hand  low))))

(second-hand-rule-002
  (premises ((plan-taken-is  throw-in)
             (leader-is-the-target-for   throw-in)))
  (actions  ((defensive-play second-hand  low))))

(second-hand-rule-003
  (premises ((plan-taken-is   drive-out)
             (has-card-for   driving-out)))
  (actions  ((defensive-play second-hand  card-for-drive-out))))

(second-hand-rule-004
  (premises ((plan-taken-is  finesse)
             (has-card-for   finessing)
             (leader-is-not-the-target-for  finesse)))
  (actions  ((defensive-play second-hand  card-for-doing-finesse))))

(second-hand-rule-005
  (premises ((has-a-sure-trick   fourth-hand)
             (has-a-sure-trick   second-hand)
             (expected-entry-is  second-hand)))
  (actions  ((defensive-play second-hand  a-sure-trick))))

(second-hand-rule-006
  (premises ((has-a-sure-trick   fourth-hand)
             (has-a-sure-trick   second-hand)))
  (actions  ((defensive-play second-hand  low))))

(second-hand-rule-007
  (premises ((has-a-sure-trick  second-hand)))
  (actions  ((defensive-play second-hand  a-sure-trick))))
```

```
(second-hand-rule-008
  (premises ((nothing)))
  (actions  ((defensive-play second-hand  low))))
```

A.2 Rules for fourth-hand player :

```
(fourth-hand-rule-001
  (premises ((no-card-to-cover  fourth-hand)))
  (actions  ((defensive-play fourth-hand  low))))

(fourth-hand-rule-002
  (premises ((current-winner-is  second-hand)
             (expected-entry-is  fourth-hand)))
  (actions  ((defensive-play fourth-hand  card-to-win))))

(fourth-hand-rule-003
  (premises ((current-winner-is  second-hand)))
  (actions  ((defensive-play fourth-hand  low))))

(fourth-hand-rule-004
  (premises ((has-at-least-2-sure-tricks  fourth-hand)))
  (actions  ((defensive-play fourth-hand  a-sure-trick))))

(fourth-hand-rule-005
  (premises ((has-card-to-win  fourth-hand)))
  (actions  ((defensive-play fourth-hand  card-to-win))))

(fourth-hand-rule-006
  (premises ((nothing)))
  (actions  ((defensive-play fourth-hand  low))))
```

REFERENCES

1. Stanier, A. BRIBIP: A Bridge bidding program. In Proc. 4th IJCAI, Tbillsi, USSR, 1975.

2. J. R. Quinlan, "A knowledge-based system for locating missing high cards in bridge". In Proc. 6th IJCAI, Tokyo, Japan, 1979.

3. Audrey Grant and Eric Rodwell, "The Joy Of Bridge" Arco Publishing, Inc. New York

4. Charles H. Goren, "Goren's Bridge Complete" The Goren Editorial Board

5. Lawrence, M. How to read your opponent's cards:The Bridge Experts' Way to Locate Missing High Cards. Prentice-Hall, Inc., N. J, 1973

6. Koenraad Lecot and D. Stoott Parker, "Control over Inexact Reasoning" P 32-43, Premier Issue Of AI EXPERT, 1986

7. Tim Finin, "Understanding Frame Language, Part II" p 51-57, AI EXPERT, Dec 1986

8. Raul E. Valdes-Perez, "Inside an Expert System Shell" p 30-45, AI EXPERT, Oct 1986

9. IQLISP Reference Manual, version 1.7, Integral Quality, 1986

10. Stanier. BRIPP - A Bridge-Playing Program Essex University, 1974.