CHAPTER 1
INTRODUCTION

Research in the fields of problem solving, expert systems, and learning has been converging on the issue of problem representation. A system's ability to solve problems, answer questions, or acquire knowledge has always been bounded by the problem representation initially given to the system. These systems can only perform efficiently to the extent that their problem representations are relevant to the problem at hand. One solution to this dilemma is to develop systems which have the ability to alter their problem representation automatically. As a problem solving system changes and improves (via learning perhaps), its problem representations should also be altered to fit this new situation.

1.1. The Problem

The research described here deals with automatically shifting from one problem representation to another representation which is more efficient, with respect to a given problem solving method and a given problem class. The study of shifts of representation can be decomposed into two main aspects: (1) exploring various representations and the shifts between them, and (2) the search through this space of representations to discover which ones are better for certain problem classes and why they are better. This research deals mainly with the former aspect leaving most control issues for future research.

Korf has suggested that "...changes of representation are not isolated 'eureka' phenomena but rather can be decomposed into sequences of relatively minor representation shifts." [Korf 80] *This research has discovered general purpose primitive representation shifts and techniques for automating them.*

The main thrust of this research is to examine the following problem:

- To explore whether a system can be developed that will automatically change its problem representation into a better representation for a particular problem solving system over a specific problem class.

The two specializations of the above problem which this research has concentrated on are as follows:

- To explore whether a system can be developed that will derive problem reduction schemas to decompose a given problem into subproblems which are easier to solve. And whether these problem reduction schemas can then be used in solving other problems in the given problem class.

- To explore whether a system can be developed that will decide intelligently when to create macro-operators and thereby create macro-operators which are general over the given problem class (i.e., can be used for many problems in the problem class) and satisficing over the extent of the search (i.e., has fewer decision nodes[1] than a state space search without macro-operators).

1.2. The Solution

To illustrate that a system can automatically change its representation, this research has defined and automated the primitive representation shifts discussed by Amarel, see [Amarel 68], in the Missionaries & Cannibals (M&C) problem. The M&C problem is stated as

> "Three missionaries and three cannibals seek to cross a river (say from the left bank to the right bank). A boat is available which will hold two people, and which can be navigated by any combination of missionaries and cannibals involving one or two people. If the missionaries on either bank of the river, or 'en route' in the river, are outnumbered at any time by cannibals, the cannibals will indulge in their anthropophagic tendencies and do away with the missionaries. Find the simplest schedule of crossings that will permit all the missionaries and cannibals to cross the river safely." [Amarel 68]

The basic model of this system is to derive shifts of representation from an analysis of the state space representation for a single example problem from a given problem class. These new

[1]A decision node is defined as a node where the present goal condition must be tested for or where a decision must be made as to which operator to apply or how to bind the operator's arguments.

representations are then used to solve subsequent problems in the same problem class.

The specific research which has been done to address the problems mentioned in the previous section is as follows:

- Defined and automated some of the primitive representation shifts explored by Amarel in the Missionaries & Cannibals (M&C) problem in [Amarel 68]. The techniques for shifting representations which this research has defined are: compiling constraints, removing irrelevant information, removing redundant information, deriving problem reduction schemas, deriving macro-operators, and deriving macro-objects. A handtrace of the M&C example has been given showing which portions of the shifts of representation discussed by Amarel, see [Amarel 68], are covered by these reformulations.

- Designed and implemented a method for problem reduction which can derive problem reduction schemas to decompose a problem into subproblems which are easier to solve. This problem reduction schema can then be used in solving other problems in the problem class. Necessary and sufficient conditions have been proven for when a problem can be reduced by this method (i.e., the *applicability conditions*). Sufficient conditions have been proven for when the problem reduction schemas derived from one problem can be used to solve all problems in the problem set (i.e., the *guaranteed conditions*). Conditions for when the problem reduction schemas derived from one problem can be used to solve all problems in the problem class have been explored (i.e., the *class-guaranteed conditions*).

- Identified when it is useful to create a macro-operator by using problem reduction schemas to highlight "homogeneities" in a solution trace. A technique for creating macro-operators which are general, iterative, and satisficing has been designed and implemented. The domain of applicability for this technique has been explored.

The system implemented for deriving and applying problem reduction schemas and iterative macro-operators is called **Sojourner.**

1.3. Background

In this section the major research on the general topic of reformulation is briefly reviewed. Specifically the research by Amarel, Korf, Lowry, and Subramanian is covered. This research will now be reviewed in chronological order.

Amarel first did research in the area of reformulation in the 1960s. His two papers on which my research focussed are "On Representation of Problems and Reasoning about Actions" [Amarel 68] and "Expert Behavior and Problem Representations" [Amarel 82]. The first of these papers deals with a series of reformulations which plausibly reformulates one representation for the Missionaries and Cannibals problem domain into another equivalent, but hopefully more efficient, representation. He proposes these reformulations and demonstrates, by hand, how the problem domain is reformulated. The second of these papers deals with the reformulation of problem/problem-solver pairs. He reformulates from a state space search paradigm to a problem reduction paradigm in the Towers of Hanoi problem domain. A series of plausible reformulations for problem/problem-solver pairs are proposed and the reformulations are demonstrated by hand.

My research is closely related to Amarel's. The reformulations he proposed in these two papers [Amarel 68, Amarel 82] are used as a basis for defining my primitive reformulations. These handtraces also provided a goal of coverage for my work (i.e., to be able to automatically derive the final representations shown in these papers when being given only the initial representations). My research chose to use the same problem domains given in these papers (i.e., Missionaries and Cannibals and Towers of Hanoi) in order to find one set of primitive reformulations which does achieve the necessary reformulations in both domains.

Korf first did research in the area of reformulation in the late 1970s, see his paper "Toward a Model of Representation Changes" [Korf 80]. He proposed a model for reformulation whose basis was that large "eureka" type reformulations can be decomposed into sequences of primitive representation shifts.[2] Korf also claimed that it was advantageous to view reformulations along two dimensions: information quantity and information structure; and further characterized

[2]Shifts of representation and reformulations are used interchangeable throughout this thesis.

reformulations as either isomorphisms or homomorphisms. He defined a set of primitive representation shifts and showed that they are composable and invertible, thereby producing a large number of representational changes from a small number of primitive representation shifts. Korf did research concerning the derivation of macro-operators in the 1980s, see his paper "Macro-Operators: A Weak Method for Learning" [Korf 85]. This research will be discussed later on in the background section of the problem reduction chapter, see Section 5.4, and in the background section of the macro-operator chapter, see Section 6.4.

As stated earlier, my research subscribes to the Korfian view that large "eureka" type shifts of representation can be automated using a small set of composable primitive reformulations. My research is a case study of this reformulation model. My research also concurs that reformulations are usefully viewed along the information quantity and information structure dimensions and that isomorphisms and homomorphisms are a natural categorization. For each of the reformulations explored in this thesis, its appropriate category placement is discussed.

Lowry, [lowry 87a, lowry 87b, lowry 88a, lowry 86, lowry 88b, lowry 88c, lowry 88d], is presently finishing his Phd. thesis in the area of reformulation. His work concerns creating an overall framework in which specific reformulations can then be automated. His framework is based on the notion of abstracting a problem domain to the most abstract representation which still characterizes the problems' behavior. This abstract representation is then used to design an abstract algorithm. Finally the abstract algorithm is implemented into a concrete algorithm which is appropriate for the present task. The abstraction method is based on finding behavioral equalities and using them as a means for creating abstract classes. The design method is based on instantiating parameterized theories. The implementation method is based on stepwise refinement. His framework is implemented in a theorem prover architecture.

My research defines specific methods for transforming one problem representation into another problem representation. These methods could probably be implemented within Lowry's framework as parameterized theories. In fact for his framework to be successful it requires incorporating the specific methods defined in other research such as mine, in order to derive a sufficient set of parameterized theories. The transformations which Lowry has used to demonstrate his framework do not cover the same types of reformulations which my research has developed.

Subramanian, [subramanian 87, subramanian 86], is also presently finishing her Phd. thesis in the area of reformulation. Her work concerns using a logic of irrelevance to reformulate a problem domain into a better representation in terms of the task to be solved. Most of this research was concerned with the derivation of new inferences of irrelevance from a given set of basic facts of irrelevance. She has also done some preliminary work on how basic facts of irrelevance can be derived. Her work is implemented in a theorem prover architecture.

My research in reformulation only touched upon the area of removing irrelevant information. My research in this area was strictly concerned with deriving the basic facts of irrelevance themselves.

Further research in reformulation is just beginning at Rutgers. In the AI/Design project, a system (KBSDE[3]) is being developed, under the supervision of Christopher Tong, to optimize a generate and test problem solver by automatically incorporating the testers. Wes Braudaway, [braudaway 88], is researching the incorporating local constraints by using reformulation techniques when there is a language mismatch between the generators and the testers. Kerstin Voigt, [voigt 88], is researching the partial incorporation of global constraints by using a least commitment approach and exploiting useful properties of the representation. Sunil Mohan, [mohan 88], is researching the derivation and use of abstraction spaces for problem solving when a tester cannot completely be moved into the generator.

1.4. Contributions

These are the main contributions which my research has made to the field of reformulation of problem representations.

1. A case study of M&C has been given to support Korf's model of representing large "eureka" shifts of representation as a sequence of general primitive shifts.
2. General, primitive techniques for shifting problem representation's have been defined. These techniques consist of compiling constraints, removing irrelevant information, removing redundant information, deriving problem reduction schemas, deriving

[3]Knowledge-Based Systems Development Environment

macro-operators, and deriving macro-objects. The two techniques explored to the point of implementation are: deriving problem reduction schemas and deriving macro-operators.

- My technique for deriving Problem Reduction Schemas have the following properties:
  a. They can be used to derive problem reduction schemas from a single example problem.

  b. A necessary and sufficient applicability condition for the technique has been defined and proven. Serializability is this condition.
     - The conditions under which a problem can have serializable subgoals have been explored.

     - What it means to be a "good" subgoal has been explored.

  c. The derived (i.e., learned) problem reduction schemas are useful over a problem graph. Specifically, a sufficient (but not necessary) description of the set of problem graphs that a problem reduction schema is useful over has been defined and proved.

  d. The conditions under which the derived problem reduction schemas are useful over a problem class has been explored.

- My technique for deriving Macro-operators have the following properties:
  a. They can be used to derive macro-operators from a single example problem.

  b. A heuristic technique for deciding when it is good to create a macro-operator has been defined and implemented.

  c. The derived (i.e., learned) macro-operators are general, iterative, and satisficing and useful over a problem class.

  d. When these types of macro-operators can be derived has been explored.

- How these two techniques could be extended to use more then one solution trace (i.e., training example) has been explored.

3. A back-propagation technique over proof schemas has been defined and implemented. This technique was used in both the inter-reduction generalization, see Section 3.3.2.5, and the intra-reduction generalization, see Section 3.3.2.6, and has the following properties:
   - Back-propagation is done over a theory of the problem solver as well as a theory of the problem domain.

   - A bias for the back-propagation is pre-defined.

   - Annotations on the proof schema severely limits the problem of irrelevant or redundant information appearing in the back-propagated result.

4. A technique for back-propagating over many-to-one functional operators without using inverse[4] operators has been defined and implemented. This technique has the following properties:
   - Back-propagation can be done over any forward operator which can be described in a predicate calculus notation with linear inequalities over finite

[4]These operators are not strictly inverses in the mathematical sense, because the forward operators are not restricted to being one-to-one.

domains.

- The complexity of this procedure is also given.

5. A problem solver which integrates searching through problem reduction spaces and state spaces has been created. Their integration benefits from the lower search costs of the problem reduction space, the lower search costs of searching subspaces of the state space, and the completeness of the state space. Furthermore, the efficiency of this problem solver supports the hypothesis that the use of multiple representation spaces can lower search costs.
6. Examination of a M&C problem graph shows that there are problem graphs where using problem reduction schemas and macro-operators decreases the number of decision nodes searched on the average, even when the techniques are not guaranteed by the previous mentioned proofs to give optimum results.

1.5. Guide to the Thesis

This thesis consists of the following eleven chapters. Chapter 1 is an introduction to the research. An overview of the reformulations is given in Chapter 2. Chapter 3 provides an overview of the derivation and use of the problem reduction schemas and the macro-operators. Chapter 4 contains the definitions and examples which give a background for the later chapters. Chapter 5 is on deriving problem reduction schemas. Chapter 6 covers deriving macro-operators. Chapter 7 is on using applicability conditions of problem solvers to create proof schemas for EBG. Chapter 8 covers a general technique for back-propagation over many-to-one functional operators without using inverse operators. Chapter 9 is on the use of the derived problem reduction schemas and macro-operators. Chapter 10 contains a discussion of the future experimental analysis of the problem reduction schemas and macro-operators which would empirically support the claims of satisficiability over the extent of the search (i.e., has fewer decision nodes). Chapter 11 contains a summary of this work, its major conclusions, and possible directions for future research. Appendix A is on derivation and use of the other reformulations which have been defined besides problem reduction schemas and macro-operators. Appendix B is a handtrace of the Missionaries and Cannibals example which shows the extent of completeness of these reformulations over the reformulations discussed by Amarel in [Amarel 68]. Appendices C & D show the program traces from my system, **Sojourner**, for the Missionaries and Cannibals and Towers of Hanoi problems respectively.

CHAPTER 2

AN OVERVIEW OF THE REFORMULATIONS

The six techniques of shifting representations this research defined are: (i) compiling constraints; (ii) removing irrelevant information; (iii) removing redundant information; (iv) deriving problem reduction schemas; (v) deriving macro-operators; and (vi) deriving macro-objects. The techniques for deriving problem reduction schemas and the derivation of macro-operators, which are the main focus of my research, are described in the next chapter. In this chapter a brief description of the other four techniques is given. The techniques for deriving problem reduction schemas and deriving macro-operators have been explored to the point of implementation. The techniques previewed in this chapter have been explored in less depth. A more detailed description of these four reformulations, including methods for their automation, can be found in Appendix A.

The description of the M&C problem class used in this chapter can be seen in Figure 2-1. There are **n** missionaries and **n** cannibals and a boat of capacity **k** with any set of valid initial states, any set of valid terminal states, and a solution path constraint of 'transportation'. 'Transportation' is defined as moving from an initial state to a terminal state by the shortest path (i.e., if an initial state is a terminal state then the problem is already solved). Specifically, the 3-2 M&C problem (i.e., where the number of missionaries and cannibals, **n**, is 3 and the capacity of the boat, **k**, is 2) will be examined in this chapter. As can be seen a state vector representation is used. A complete definition for all the problem components is given in Chapter 4.

2.1. Compiling Constraints

The technique of "compiling constraints" (i.e., otherwise known as moving the tester into the generator) has been used by many fields in Computer Science. This research uses it to compile the consistency constraints of a problem into the preconditions of that problem's operators. For instance, in M&C this is the difference between saying 'traverse the river and see if you get eaten'

STATE DESCRIPTION LANGUAGE

Functions

ML	Missionaries at Left	**CL**	Cannibals at Left
MR	Missionaries at Right	**CR**	Cannibals at Right
B	side the boat is on		
n	number of cannibals or missionaries		
k	capacity of boat		

Primitive Object Sets

Z_0, non-negative integers,
(for functions ML, MR, CL, CR, n, k)
{L, R} (for function B)

State Vectors

s_i state vector at time i

Operator Arguments

MT missionaries transported
CT cannibals transported

OPERATOR DESCRIPTIONS

Action	Preconditions[5]	Change[6]
Traverse River from Left to Right		
TRLR(MT,CT)	$ML(s_x) \geq MT \wedge$	$ML(s_y)=ML(s_x)-MT \wedge$
	$CL(s_x) \geq CT \wedge$	$CL(s_y)=CL(s_x)-CT \wedge$
	$B(s_x)='L' \wedge$	$B(s_y)='R'$
	$0<MT(s_x)+CT(s_x) \leq k$	$CR(s_y)=CR(s_x)+CT$
		$MR(s_y)=MR(s_x)+MT$
Traverse River from Left to Right		
TRRL(MT,CT)	$MR(s_x) \geq MT \wedge$	$ML(s_y)=ML(s_x)+MT \wedge$
	$CR(s_x) \geq CT \wedge$	$CL(s_y)=CL(s_x)+CT \wedge$
	$B(s_x)='R' \wedge$	$B(s_y)='L'$
	$0<MT(s_x)+CT(s_x) \leq k$	$CR(s_y)=CR(s_x)-CT$
		$MR(s_y)=MR(s_x)-MT$

Figure 2-1: Description of Missionaries and Cannibals Problem Class

[5]There has been some question as to why the consistency constraints and the operator preconditions are divided up in this way. This research divided up these constraints into what seemed to be the normal division for the problem. All consistency constraints can be made into preconditions on operators and vice versa.

[6]A change list is used instead of add and delete lists for simplicity.

CONSISTENCY CONSTRAINTS[7]

Law of Survival
$(\neg(ML(s)=0) \Rightarrow \neg(ML(s)<CL(s))) \wedge$
$(\neg(MR(s)=0) \Rightarrow \neg(MR(s)<CR(s)))$

Law of Conservation
$n = ML(s)+MR(s)$(total # of Missionaries)
$= CL(s)+CR(s)$(total # of Cannibals)

SOLUTION PATH CONSTRAINT[8]

Transportation (See text for definition.)

PROBLEM PARAMETERS

n	number of missionaries and cannibals
k	capacity of the boat

Figure 2-1 Cont: Description of M&C Problem Class

or 'only traverse the river if you are sure that you will not get eaten'. This prevents any states, which violate the law of survival, from being generated in the state space graph. For instance,

[7]There has been some question as to why the consistency constraints and the operator preconditions are divided up in this way. This research divided up these constraints into what seemed to be the normal division for the problem. All consistency constraints can be made into preconditions on operators and vice versa.

[8]The consistency constraints and the solution path constraint can both be seen as constraints on the solution path. The consistency constraint must be true of ever state on the solution path. But they remain separated because their structure is inherently different. A consistency constraint can be tested at a state whereas a solution path constraint cannot. So for instance, a consistency constraint can be used to define the space of all legal states. This property is important in Chapter 7.

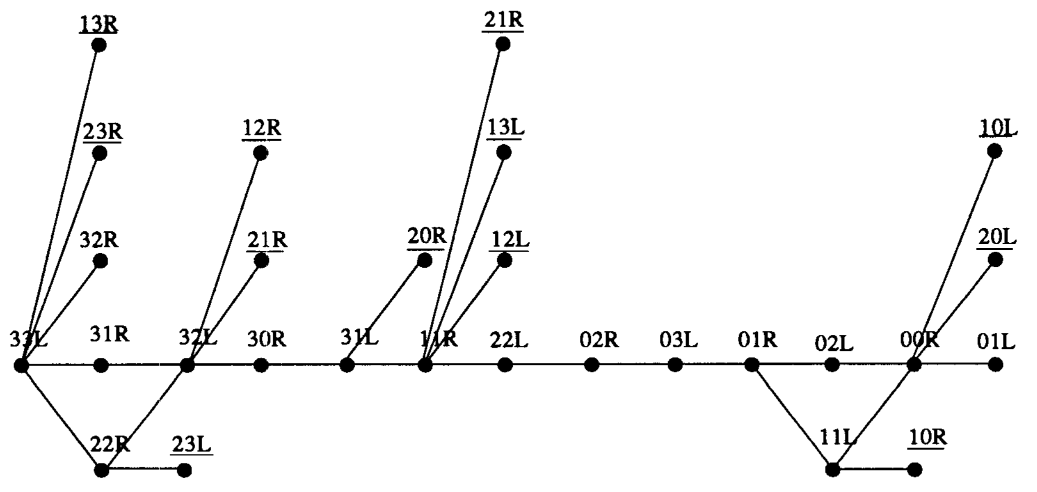
Figure 2-2 represents the elementary M&C solution graph without compiled constraints and Figure 2-3 represents the elementary M&C solution graph with compiled constraints. The method used for automating the technique of "compiling constraints" consists of back-propagating the consistency constraints and deriving the weakest preconditions for each operator. An example of the alterations made to the problem class representation from "compiling constraints" is shown in Figure 2-4. A consistency constraint is back-propagated through the operator and the resultant constraints are added to the operator as preconditions. After the compilation is performed for each of the operators, that consistency constraint is redundant information and can be removed. Research in this area has begun recently at Rutgers by Wes Braudaway, Kerstin Voigt, and Sunil Mohan. Their research is reviewed in Appendix A.

2.2. Removing Irrelevant Information

The technique of "removing irrelevant information" concerns the removal of additional information in the problem statement which is never used in solving problems in this problem class. For instance, the M&C problem stated as follows contains irrelevant information: "Three missionaries; who are named Sue, Peter, and Mark; and three cannibals; who are named Mary, Paul, and Cathy; seek to cross a river...." The names of the missionaries and cannibals are never used in solving the problem and are therefore irrelevant information. Ideally the system would derive that the names of the cannibals and missionaries are not necessary for any reasoning for solving the problem. The method used for automating the technique of "removing irrelevant information" is a much stronger constraint which consists of examining the operator descriptions, consistency constraints, and solution path constraints for this problem class. If the information in question is never referenced in any of these three areas[12], then it is irrelevant information for this problem class. This method for automation is presently very syntactic and more research will need to be done to find a better method of automation. For instance the state vector representation, **<Sue,Pete,Mark,Mary, Paul,Kathy,Boat>**, whose values are **<1,0,1,0,1,1,L>**[13], are transformed

[12]This includes any further definitions accessed by these three areas. Also it was suggested by Bernard Silver that heuristics would also need to be checked. This research has only dealt with breadth first search, but if a heuristic search is used the heuristics would also have to be checked.

[13]Where 1 represents that a missionary or cannibal is on the left bank and a 0 represents that a missionary or cannibal is on the right bank and the boat is on the left bank.



Where XYZ : X = # of missionaires

Y = # of cannibals

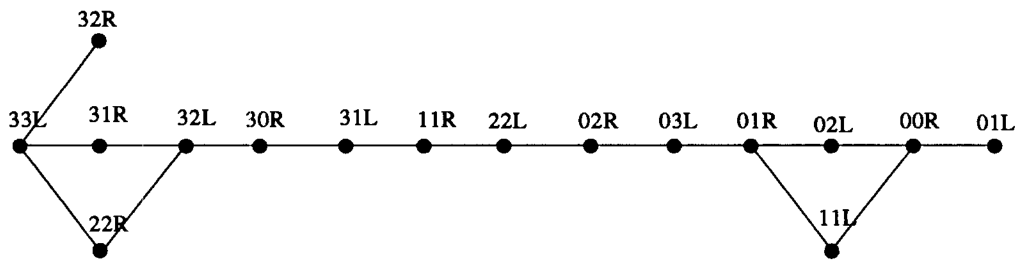
Z ∈ {L,R} the bank of the river which the boat is on

Parameters : n = 3
k = 2

Figure 2-2: Missionaries and Cannibals Solution Graph without Compiled Constraints[9]

into the state vector representation, **<M,M,M,C,C,C,Boat>**, whose values remain the same. Devika Subramanian has just completed a PhD in this area. This research will be reviewed in Appendix A.

[9]The first number represents the number of missionaries on the left side (ML). The second number represents the number of cannibals on the left side (CL). The last character represents which side of the river the boat is on (B). It contains L for left or R for right. So in the state **<1,3,R>** there is 1 missionary on the left bank, 3 cannibals on the left bank, and the boat is on the right bank (R). So by the Law of Conservation, there are 2 missionaries and 0 cannibals on the right bank. The underlined nodes represent illegal states.



Where XYZ : X = # of missionaires
Y = # of cannibals
Z ∈ {L,R} the bank of the river which the boat is on

Parameters : n = 3
k = 2

Figure 2-3: Missionaries and Cannibals Solution Graph with Compiled Constraints

2.3. Removing Redundant Information

The technique of "removing redundant information" concerns removing information which is expressed in more than one way. This must be performed such that no information is lost (i.e., it is derivable if not explicitly expressed). For instance, the M&C problem representation shown in Figure 2-5 contains redundant information (e.g., the missionaries on the right bank and the cannibals on the right bank). This information can be condensed into a more compact form as shown in Figure 2-6. Notice that not only do some of the consistency constraints change, but there is no longer a need for the representation of the number of missionaries or cannibals on the right bank.

There are several methods by which redundant information can be removed. One method is to 'notice' the redundancy, for example **ML(s)=n** ∨ **ML(s)=0** ∨ **ML(s)=CL(s)** is always true whenever **(¬(ML(s)=0)⇒¬(ML(s)<CL(s)))** ∧ **(¬(MR(s)=0)⇒¬(MR(s)<CR(s)))** is true. This example states that the Law of Survival will be satisfied for both banks whenever all the missionaries are on the left bank, none of the missionaries are on the left bank, or an equal number of missionaries and cannibals are on the left bank. A more promising method concerns using reformulation techniques to derive this compact form. For instance, the more compact

Before Compilation

Valid Operator Action	Precondition	Change
TRLR[10](MT,CT)	$B(x)='L' \wedge$	$MR(y)=MR(x)+MT \wedge$
	$1 \leq MT+CT \leq k \wedge$	$ML(y)=ML(x)-MT \wedge$
	$ML(x) \geq MT \geq 0 \wedge$	$CR(y)=CR(x)+CT \wedge$
	$CL(x) \geq CT \geq 0$	$CL(y)=CL(x)-CT$

Consistency Constraints

$$(\neg(ML(s)=0) \Rightarrow \neg(ML(s)<CL(s))) \wedge$$
$$(\neg(MR(s)=0) \Rightarrow \neg(MR(s)<CR(s)))$$

After Compilation

Valid Operator Action	Precondition	Change
TRLR(MT,CT)	$B(x)='L' \wedge$	$MR(y)=MR(x)+MT \wedge$
	$1 \leq MT+CT \leq k \wedge$	$ML(y)=ML(x)-MT \wedge$
	$ML(x) \geq MT \geq 0 \wedge$	$CR(y)=CR(x)+CT \wedge$
	$CL(x) \geq CT \geq 0 \wedge$	$CL(y)=CL(x)-CT$
	$((ML(x)-MT>0)$[11] $\Rightarrow \neg(ML(x)-MT<CL(x)-CT)) \wedge$	
	$((MR(x)+MT>0)$[11] $\Rightarrow \neg(MR(x)+MT<CR(x)+CT))$	

Figure 2-4: Before and After "Compiling Constraints" by Weakest Precondition

representation in Figure 2-6 is derived from that in Figure 2-5 simply by re-expressing the formulae in disjunctive normal form and simplifying. The derivation of this reformulation is shown in Appendix A. Notice that the Law of Conservation is no longer needed in this new formulation.

[10]TRLR means traverse river from left to right. MT is the number of missionaries to traverse the river. CT is the number of cannibals to traverse the river. All the other variables are as previously defined in Figure 2-1

[11]Some elementary simplifications have been performed on these formulae.

Law of Survival

$(\neg(\mathbf{ML(s)=0}) \Rightarrow \neg(\mathbf{ML(s)<CL(s)})) \wedge$
$(\neg(\mathbf{MR(s)=0}) \Rightarrow \neg(\mathbf{MR(s)<CR(s)}))$

Law of Conservation

ML(s)+MR(s) = CL(s)+CR(s) = n

State Description Language

ML(s) CL(s) B(s) MR(s) CR(s)

Figure 2-5: Missionaries and Cannibals Representation with Redundant Information

Law of Survival

ML(s)=n $\vee$ **ML(s)=0** $\vee$ **ML(s)=CL(s)**

State Description Language

ML(s) CL(s) B(s)

Figure 2-6: Missionaries and Cannibals Representation without Redundant Information

2.4. Derivation of a Macro-Object

After the removal of irrelevant information, mentioned earlier, the state description language contains three indistinguishable missionaries and three indistinguishable cannibals and the location of the boat. The state vector representation at this point is **<M,M,M,C,C,C,Boat>**, whose values are **<1,0,1,0,1,1,Left>**[14]. Whenever there are indistinguishable objects in a domain, they can be reformulated into a bag of such objects without loss of generality. The state vector representation now becomes **<ML-Bag,CL-Bag,Boat>**, whose values are **<{M,M},{C,C},Left>**[15]. **ML-bag**

[14]Where 1 represents that a missionary or cannibal is on the left bank and a 0 represents that a missionary or cannibal is on the right bank and the boat is on the left bank.

[15]There are two missionaries on the left bank and two cannibals on the left bank and the boat is on the left bank.

represents the bag of missionaries on the left bank and **CL-bag** represents the bag of cannibals on the left bank. There is a second transformation which can also be performed which states that if the operators can be transformed to mathematical statements, then a bag of objects can be transformed into a cardinality object, which specifies how many objects are in a specific bag. An example illustrating this last transformation can be seen in Appendix A. The state vector representation following these transformations is **<M,C,Boat>**, whose values are **<2,2,Left>**[16].

[16]There are two missionaries on the left bank and two cannibals on the left bank and the boat is on the left bank.

CHAPTER 3

AN OVERVIEW OF THE DERIVATION AND USE OF PROBLEM REDUCTION SCHEMAS AND MACRO-OPERATORS

Reformulation (i.e., shifting representations) can be broken into two parts: deriving the reformulations and applying the reformulations. The reformulations which will be discussed from these two points of view are deriving problem reduction schemas and deriving macro-operators. Section 3.1 demonstrates how the derived problem reduction schemas and macro-operators are used by the problem solver to solve problems without requiring much search. Section 3.2 gives a brief overview of how the derivation of problem reduction schemas and the derivation of macro-operators are integrated in **Sojourner**. Section 3.3 demonstrates how the problem reduction schemas and macro-operators are automatically derived by the system. For a description of the derivation and use of the other reformulations see Appendix A. For a more detailed description of the derivation of problem reduction schemas, see Chapter 5. For a more detailed description of the derivation of macro-operators, see Chapter 6. For a more detailed description of the use of both the problem reduction schemas and the macro-operators, see Chapter 9.

3.1. How Problem Reduction Schemas and Macro-operators Lessen Search

How can problem reduction schemas and macro-operators lessen the amount of search? To illustrate their power to lessen the amount of search, a problem reduction schema and some macro-operators derived by my system for the example domain of M&C will now be examined. Also the problem solver process itself will be described.

The description of the M&C problem class used in this chapter can be seen in Figure 3-1. It differs from the description in Chapter 2 because the previously discussed reformulations are assumed to have already been applied. There are **n** missionaries and **n** cannibals and a boat of capacity **k** with any set of valid initial states, any set of valid terminal states, and a solution path constraint of 'transportation'. 'Transportation' is defined as moving from an initial state to a terminal state by the shortest path (i.e., if an initial state is a terminal state then the problem is already solved). Specifically, the 8-4 M&C problem (i.e., where the number of missionaries, **n**, is 8 and the number of cannibals, **n**, is 8 and the capacity of the boat, **k**, is 4) and the problem reduction schemas and macro-operators which are derived from the training instance given in Figure 3-2 will be examined.

A flow diagram of the problem solver process can be seen in Figure 3-3 and its corresponding ALGOL description is given in Figure 3-4. The problem solver's input (i.e., the new problem to be solved) is a problem description. The output of the system is a solution path for this new problem. The major subprocesses of this problem solver are as follows:

1. Macro-operator Problem Solver
2. Problem Reduction Schema Problem Solver
3. State Space Search Problem Solver

The problem description[19], consisting of a state description language, a set of operators, consistency constraints, solution path constraints, problem parameters, and a state-pair[20], is input to the system. The problem solver also has access to the database of derived problem reduction schemas and macro-operators. The Macro-operators Problem Solver process (MPS) attempts to apply a macro-operator which solves the problem. If there is an applicable macro-operator then it is applied and the system returns a solution path and terminates. If there is no applicable macro-operator, then the problem description is sent to the Problem Reduction Schemas Problem Solver process (PRSPS). If there is an applicable problem reduction schema, then it is applied and the problem is reduced into subproblems. These subproblems must now be solved. The entire problem solver recurses upon each of the new problems (i.e., the subproblems). If this recursion succeeds, then a solution path is returned and the system terminates. If the recursion fails or there are no

[19]A precise definition of problem description is given in Chapter 4

[20]A state-pair consists of the initial state and terminal state.

STATE DESCRIPTION LANGUAGE

Functions

ML	Missionaries at Left
CL	Cannibals at Left
B	side the boat is on
n	number of cannibals or missionaries
k	capacity of boat

Primitive Object Sets
Z_0, non-negative integers,
(for functions ML, CL, n, k)
{L, R} (for function B)

State Vectors
s_i state vector at time **i**

Operator Arguments
MT missionaries transported
CT cannibals transported

OPERATOR DESCRIPTIONS

Action	Preconditions[17]	Change
Traverse River from Left to Right **TRLR(MT,CT)**	$ML(s_x) \geq MT \wedge$ $CL(s_x) \geq CT \wedge$ $B(s_x)='L' \wedge$ $0<MT(s_x)+CT(s_x) \leq k$	$ML(s_y)=ML(s_x)-MT \wedge$ $CL(s_y)=CL(s_x)-CT \wedge$ $B(s_y)='R'$
Traverse River from Left to Right **TRRL(MT,CT)**	$n-ML(s_x) \geq MT \wedge$ $n-CL(s_x) \geq CT \wedge$ $B(s_x)='R' \wedge$ $0<MT(s_x)+CT(s_x) \leq k$	$ML(s_y)=ML(s_x)+MT \wedge$ $CL(s_y)=CL(s_x)+CT \wedge$ $B(s_y)='L'$

Figure 3-1: Description of Missionaries and Cannibals Problem Class

[17]There has been some question as to why the consistency constraints and the operator preconditions are divided up in this way. This research divided up these constraints into what seemed to be the normal division for the problem. All consistency constraints can be made into preconditions on operators and vice versa.

CONSISTENCY CONSTRAINTS[18]

Law of Survival
$ML(s)=n \vee ML(s)=0 \vee ML(s)=CL(s)$

SOLUTION PATH CONSTRAINT

Transportation (See text for definition.)

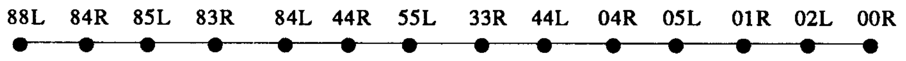
PROBLEM PARAMETERS

n	number of missionaries and cannibals
k	capacity of the boat

Figure 3-1 Cont: Description of M&C Problem Class

applicable problem reduction schemas, then some form of search must be used to solve the problem. The MPS process, the PRSPS process and the macro-operators and problem reduction schemas they apply will now be examined in more detail.

[18]There has been some question as to why the consistency constraints and the operator preconditions are divided up in this way. This research divided up these constraints into what seemed to be the normal division for the problem. All consistency constraints can be made into preconditions on operators and vice versa.



Where XYZ : X = # of missionaires
Y = # of cannibals
Z ∈ {L,R} the bank of the river which the boat is on

Parameters : n = 8
k = 4

Figure 3-2: 8-4 Missionaries and Cannibals Problem Solution Path

3.1.1. The Structure and Use of Problem Reduction Schemas

A problem reduction schema which is derived by **Sojourner**[21] for the 8-4 M&C problem with the training instance shown in Figure 3-2 can be seen in Figure 3-5. The problem is broken into three subproblems (i.e., preparation, critical, and remainder) which are connected sequentially. Intuitively these three subproblems correspond to the following. The critical subproblem is the most "important" subproblem. The preparation subproblem is the subproblem of reaching a state where the "important" subproblem can be solved from the initial state. The remainder subproblem is the subproblem of solving the rest of the problem from a state where this "important" subproblem has been solved. The three subproblems are *connected sequentially* if the following two conditions hold. Every terminal state of the preparation subproblem can reach some initial state of the critical subproblem by the application of one single state space operator (i.e., the first transition operator). Every terminal state of the critical subproblem can reach some initial state of the remainder subproblem by the application of one single state space operator (i.e., the second transition operator). These three subproblems correspond to the "**Z**" diagram for M&C described by Amarel [Amarel 68], see Figure 3-6.

Intuitively the problem reduction schema in Figure 3-5 specifies that a solution to a new problem whose initial state occurs during the first subproblem (i.e., the preparation subproblem) and whose

[21]The implementation of the system described in this thesis is called **Sojourner**.

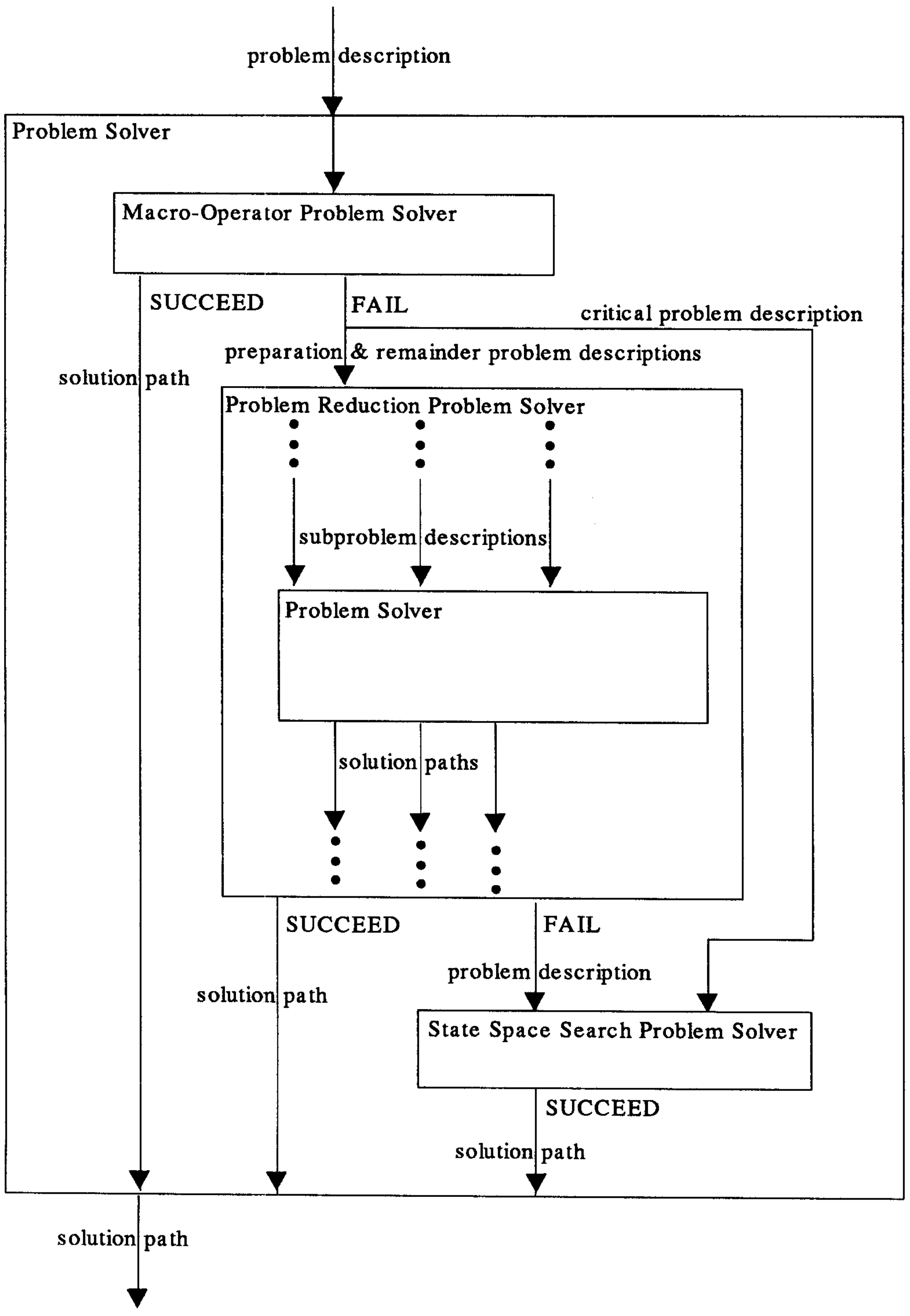


Figure 3-3: Flow Diagram of Problem Solver Algorithm

```
Problem-Solver(PROBLEM,TYPE,SOLUTION)
begin
 Macro-Operator-Problem-Solver(PROBLEM,TERMINATE-1,SOLUTION-1),
 If TERMINATE-1=true
   then SOLUTION=SOLUTION-1
   else if TYPE=CRITICAL
        then State-Space-Search-Problem-Solver(PROBLEM,SOLUTION)
        else do
            Problem-Reduction-Problem-Solver(PROBLEM,TERMINATE-2,SOLUTION-2)
            if TERMINATE-2=true
              then SOLUTION=SOLUTION-2
              else State-Space-Search-Problem-Solver(PROBLEM,SOLUTION)
           end
end

Problem-Reduction-Problem-Solver(PROBLEM,TERMINATE,SOLUTION)
begin
   .
 parse into three subproblems
   .
 Problem-Solver(SUBPROBLEM-1,preparation,SOLUTION-1)
 Problem-Solver(SUBPROBLEM-2,critical,SOLUTION-2)
 Problem-Solver(SUBPROBLEM-3,remainder,SOLUTION-3)
   .
 compose solutions to subproblems
   .
end
```

Figure 3-4: Algorithm for the Problem Solver

	preparation	**critical**	**remainder**
initial state description:	ML(s)=n	ML(s)≠n,ML(s)≠0, ML(s)=CL(s)	ML(s)=0
terminal state description:	*or*([ML(s)=n, CL(s)≥*max*(1,n-k), CL(s)≤*min*(n,n+k-2,n+(k/2)-1), B(s)="L", n≥2,k≥1], [ML(s)=n, CL(s)=n, B(s)="L", n≥2,k≥2])	*or*([ML(s)≥1, ML(s)≤*min*(n-1,k), CL(s)=ML(s), B(s)="L"], [ML(s)≥1,ML(s)≤*min*(n-1,k/2), CL(s)=ML(s), B(s)="L"])	terminal state

Figure 3-5: Problem Reduction Schema for the Missionaries and Cannibals Problem Class

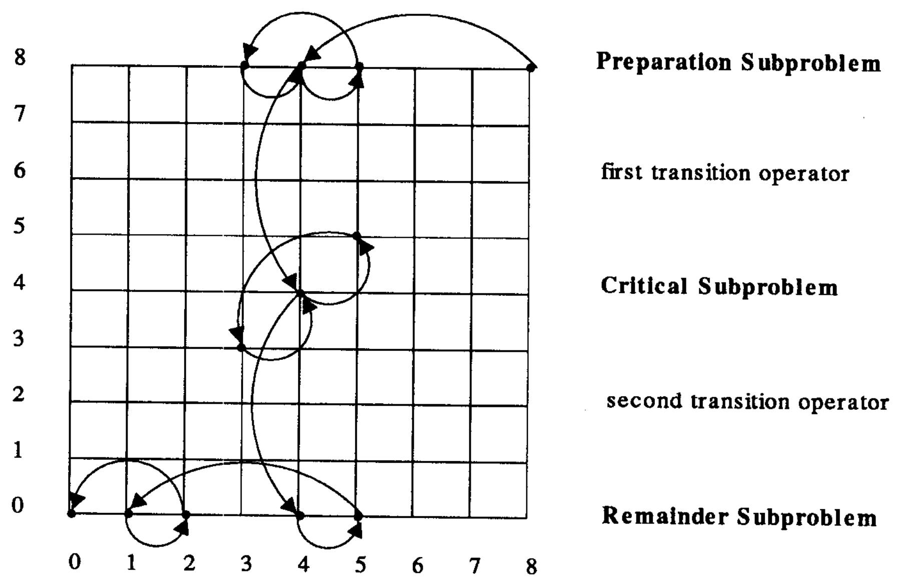


Figure 3-6: Z diagram for the Elementary 8-4 Missionaries and Cannibals Problem

terminal state occurs during the last subproblem (i.e., the remainder subproblem) can be obtained by sequentially solving the following three subproblems and finding the transition operators between them:[22]

1. From the initial state, while keeping all the missionaries on the left bank, transport some number of cannibals to the right bank such that you can transport the same number of missionaries to the right bank in a single trip across the river.

   • Transport to the right bank the number of missionaries necessary to make the number of missionaries and cannibals equal on the left bank (i.e., the first **transition** operator).

2. While keeping the number of cannibals and missionaries equal on the left bank, transport missionaries and cannibals to the right bank until you can transport the rest of the missionaries to the right bank in a single trip across the river.

   • Transport all the missionaries remaining on the left bank to the right bank (i.e., the second **transition** operator).

3. While keeping all the missionaries on the right bank transport the rest of the cannibals to their final position as defined by the terminal state.

The PRSPS process is responsible for applying a stored problem reduction schema in order to solve the new problem. The process takes as input the problem description of the new problem to be solved. It decides if there is a problem reduction schema which will be useful in solving this problem. If there is, it applies the problem reduction schema and the entire problem solver recurses on the preparation and remainder subproblems until the entire problem is solved. The problem solver does not recurse on the critical subproblem because it is, by definition, an "homogeneous" subproblem[23]. The critical subproblem is solved by either applying a macro-operator or by some form of state space search. The PRSPS process will now be examined step by step.

[22]Bear in mind that this is just an illustration of one possible use of the problem reduction schema. Other possibilities are where the initial and terminal states fall respectively in the preparation and critical, the critical and remainder, the preparation and preparation, etc. The only restriction is that the subproblem in which the terminal state occurs does not precede the subproblem in which the initial state occurs. This is why they are referred to as problem reduction schemas instead of problem reduction operators (i.e., they can be applied in many ways).

[23]An homogeneous subproblem is only concerned with solving a single subgoal. The subgoal can be expressed as a conjunction of subgoals, but it is solved as a single entity.

First a problem reduction schema is retrieved[24]. If the initial and terminal states of the new problem occur in subproblems of the problem reduction schema, and the subproblem in which the terminal state occurs does not precede[25] the subproblem in which the initial state occurs, then the algorithm sets up the new subproblems. Otherwise another problem reduction schema is retrieved and the process is repeated. A *state occurs in a subproblem* if it implies that subproblem's invariant[26]. The *invariant* of a subproblem specifies a constraint which must be satisfied by all states in any solution path for the subproblem. It therefore defines an equivalence class of states. The invariant can be used as a partial description of the set of initial states for a subproblem. The initial state descriptions shown in Figure 3-5 are actually the invariants of their respective subproblems.). The subproblem which contains the initial state, the subproblem which contains the terminal state, and any subproblems in between these two subproblems are set up. Subproblems are set up by having their initial and terminal states partially defined so they can be recursed on to be solved.

For instance if the new initial state occurs in the preparation subproblem and the new terminal state occurs in the remainder subproblem, then three subproblems are created (i.e., the preparation, the critical, and the remainder). The preparation subproblem is defined by the new problem's initial state and the preparation subproblem's goal. The goal of a subproblem specifies the constraints which are satisfied by all states which can, by the application of the transition operator with some valid set of operator arguments, reach the following subproblem (i.e., the transition operator is specified but its arguments are variablized). This goal can therefore be used as a partial description of the set of terminal states for a subproblem. The terminal state descriptions shown in Figure 3-5

[24]Presently the problem reduction schemas are retrieved sequentially from the list of stored problem reduction schemas.

[25]The initial state must not appear in a later subproblem than the terminal state because all of the problem reduction schemas and macro-operators have an inherent direction. This problem can be over come by using the inverse of each training example as a further training example. Then the problem reduction schemas and macro-operators for the opposite direction will also be derived.

[26]For instance the state vector <3,3,L> satisfies the invariant **ML=n** where **n=3**. In **Sojourner** this is actually implemented by proving that the state implies the invariant. For instance the state vector <3,3,L> represents the state components **ML=3** $\wedge$ **CL=3** $\wedge$ **B=L** which must imply **ML=n** where **n=3**. "Imply" is used instead of "satisfy" throughout this thesis because a state can be a partial description in many cases. In these cases the term "satisfy" cannot be used. Therefore to avoid confusion by using both terms, the weaker term "implies" is used throughout.

are actually the goals of their respective subproblems.). The critical subproblem is defined by the critical subproblem's invariant (i.e., the partial description of the set of initial states for the critical subproblem) and the critical subproblem's goal. The remainder subproblem is defined by the remainder subproblem's invariant (i.e., the partial description of the set of initial states for the remainder subproblem) and the new problem's terminal state.

The next step in the algorithm sequentially solves the three subproblems by recursing on the entire problem solver. For instance back in our example, the preparation subproblem is solved by recursion on the entire problem solver (i.e., either by applying a macro-operator, by applying a problem reduction schema, or by executing a state space search). This produces a totally defined terminal state for the preparation subproblem. Then the single state space operator between the preparation and critical subproblems is chosen and bound. This requires finding an applicable operator and binding its arguments. Because this operator must connect a state which is totally defined with a state which is partially defined, its state space is very limited. Then the critical subproblem, which now has a totally defined initial state, is solved by recursion on the MPS process or by executing a state space search[27]. This produces a totally defined terminal state for the critical subproblem. Then the single state space operator between the critical and remainder subproblems is chosen and bound as above. The remainder subproblem is then solved by recursion on the entire problem solver (i.e., either by applying a problem reduction schema, by applying a macro-operator, or by executing a state space search). This procedure produces a totally defined terminal state for the remainder problems, which is also the terminal state for the original problem.

3.1.2. The Structure and Use of Macro-Operators

The macro-operators which are derived by **Sojourner** for the 8-4 M&C problem with the training instance shown in Figure 3-2 can be seen in Figure 3-7. The basic premise behind the macro-operators created by my system is that the body of the macro-operator will be applied iteratively until the goal of the macro-operator has been achieved.

[27]No problem reduction schema can be applicable to a critical subproblem, because it is by definition a "homogeneous" subproblem.

Preparation macro-operator

```
invariant(ML(s)=n),
precondition([B(s)='L',   ML(s)=n   1≤CL(s)≤n,   k≥1]),
goal(or([B(s)='L',   ML(s)=n,   n≥2,   max(1, n-k)≤CL(s)≤min(n, n+k-2, n+(k/2)-1)],
        [B(s)='L',   ML(s)=n,   n≥2,   CL(s)=n,   k≥2])),
operators([trlr,trrl]),
arguments([[0,x1],   [0,1]]),
start(1),
definitions([0 < x1 ≤ min(k, CL(s))])
```

Critical macro-operator

```
invariant([ML(s)≠n,   ML(s)≠0,   ML(s) = CL(s)]),
precondition([B(s) = 'R',   CL(s) = ML(s),   1≤ML(s)≤n-1.5,   k≥1]),
goal(or([B(s) = 'L',   1≤ML(s)≤ min(n-1, k),     CL(s) = ML(s)],
        [B(s) = 'L',   1≤ML(s)≤ min(n-1, k/2),   CL(s) = ML(s)])),
operators([trrl,trlr]),
arguments([[2,2],   [1,1]]),
start(2),
definitions([])
```

Remainder macro-operator

```
invariant([ML(s) = 0]),
precondition([B(s) = 'R',   0≤CL(s)≤n-1,   ML(s) = 0,   k≥1]),
goal(terminal state),
operators([trrl, trlr]),
arguments([[0,1],   [0,x2]]),
start(1),
definitions([0 < x2≤min(k, Cl(s))])
```

Figure 3-7: Macro-operators for the Missionaries and Cannibals Problem Class

An intuitive look at the three macro-operators created by **Sojourner** will now be presented. There are four basic parts to the macro-operators: the precondition, the goal, the loop body, and the invariant. The loop body is specified in Figure 3-7 by the operator list, *operators*, the argument list, *arguments*, the placement in the operator list of the initial operator, *start*, and the list of definitions, *definitions*. The invariant is used by my system to specify constraints which must be implied by all the states within a macro-operator. The basic premise is that if the preconditions are satisfied then the loop body will be executed until the goal is achieved. Also all states expanded during the execution of the macro-operator must imply the invariant.

The preparation macro-operator's preconditions are as follows: the boat is on the left bank, all the missionaries are on the left bank, there are some cannibals on the left bank, and the boat has a capacity of at least one. The preparation macro-operator's loop body consists of: transporting as many cannibals as will fit in the boat or are on the left bank from the left bank to the right bank followed by transporting one cannibal back to the right bank in order to row the boat. The preparation macro-operator's goal is as follows: the boat is on the left bank, there are at least two of each missionaries and cannibals, all the missionaries are still on the left bank, and either the number of cannibals is equal to the number of missionaries and the capacity of the boat is at least two or the number of cannibals on the left bank are such that after transporting one more boat load of missionaries to the right bank, the number of cannibals and missionaries on the left bank will be equal[28]. The preparation macro-operator's invariant is that all the missionaries are on the left bank.

The critical macro-operator's preconditions are as follows: the boat is on the right bank, the number of cannibals on the left bank is equal to the number of missionaries on the left bank, at least two, but not all, of the missionaries are on the right bank, and the capacity of the boat is at least one[29]. The critical macro-operator's loop body consists of: transporting one cannibal and one missionary from the right bank to the left bank possibly followed by some number of executions of

[28]It should be noted that the system only verifies when it is possible to reach a state in the next subproblem using a single operator, not when it is possible to achieve that next subproblem's goal from that state.

[29]It should be noted that the system allows the preconditions for a macro-operator to be over-generalized because the operator arguments are treated as variables in deriving the preconditions whether or not they are actually variablized in the macro-operator itself.

the loop of transporting two cannibals and two missionaries from the left bank to the right bank and transporting one cannibal and one missionary from the right bank to the left bank[30]. The critical macro-operator's goal is as follows: the boat is on the left bank, there is an equal number of missionaries and cannibals on the left bank, there are some missionaries on the right bank, and either all the missionaries on the left bank can fit in half the boat whose capacity is greater than two or all the missionaries can fit in the boat. The critical macro-operator's invariant is that there are missionaries on both banks and the same number of missionaries as cannibals are on the left bank.

The remainder macro-operator's preconditions are as follows: the boat is on the right bank, all the missionaries are on the right bank, some, but not all, of the cannibals are on the left bank, and the capacity of the boat is at least one. The remainder macro-operator's loop body consists of: transporting one cannibal from the right bank to the left bank in order to row the boat followed by transporting as many cannibals as will fit in the boat or are on the left bank from the left bank to the right bank. The remainder macro-operator's goal is the terminal state of the new problem. The remainder subproblem's invariant is that all the missionaries are on the right bank.

First the structure of a macro-operator will be examined, then how they are actually applied by the system will be illustrated. The macro-operator is composed of seven components: *invariant*, *precondition*, *goal*, *operators*, *arguments*, *start*, and *definitions*. The *invariant* component specifies some condition which must be true of every state which occurs within this macro-operator's domain (i.e., the subproblem to which the macro-operator corresponds). The *precondition* component is the set of conditions which must be satisfied for this macro-operator to be applicable. The *goal* component is the set of conditions which this macro-operator attempts to achieve. The *operators* component is the list of operators which comprises the body of the loop. The *arguments* component is the list of arguments for each of the operators in the operators component. The *start* component specifies the placement in the operator list of the initial operator. The *definition* component is the set of conditions which must be satisfied by any variablized arguments in the arguments component. The invariant and goal components of the macro-operator are also the set of initial and terminal

[30]Notice that the technique for deriving macro-operators does allow for under-generalization. If an argument to the instances of an operator in the training example matches exactly, then no generalization is done.

states of some subproblem in a problem reduction schema. Therefore a macro-operator is said to be associated with a subproblem and vice versa.

The MPS process is responsible for applying the stored macro-operators in order to solve the new problem. It takes as input the problem description of the new problem. From this it determines if there is a stored macro-operator which will be useful in solving this problem. If there is such a macro-operator it applies the macro-operator and solves the new problem. The MPS process will now be described step by step.

First a macro-operator is retrieved[31]. If the initial and terminal states of the new problem occur in the subproblem which is the domain[32] of this macro-operator (i.e., they satisfy the invariant), then this macro-operator should be applied. Otherwise another macro-operator is retrieved. When a macro-operator should be applied, its preconditions are checked to see if the macro-operator is applicable.

If the initial and terminal state occur in a subproblem which is associated with a macro-operator (i.e., the initial and terminal state both imply the invariant), then the macro-operator *should* be applied although the macro-operator *may not actually be applicable* in this initial state (i.e., the initial state does not imply the preconditions). If the present state does not imply the preconditions, then the system will expand the next state in the breadth-first search sequence. Since the search space is restricted within this subproblem, any state expanded must satisfy the invariant. This new state is tested to see if it implies the goal of the macro-operator. If it does imply the goal, then the process returns and the macro-operator need not be applied. If it does not imply the goal, then the state is tested to see if it implies the preconditions of the macro-operator. If it does imply the preconditions, then the process returns and the macro-operator can now be applied. If it does not imply the preconditions, then the breadth-first state space search will continue, expanding one new state at a time, until the goal is achieved, a state is found where the macro-operator is applicable, or this subproblem has been exhaustively explored by the breadth-first search.

---

[31]Presently the macro-operators are retrieved sequentially from the list of stored macro-operators.

[32]A macro-operator's domain is the subproblem in which the macro-operator should be applied.

When a macro-operator is applicable, the system must next determine whether the terminal state of the new problem implies the goal of this macro-operator. This must be determined, in order to decide when to test for the arrival at the terminal state. If the terminal state of the new problem does not imply the goal of the macro-operator, then the achievement of the terminal state must be checked for between each operator in the loop body. If the terminal state of the new problem implies the goal of the macro-operator, then the loop body of the macro-operator can be executed as a whole and the arrival at the terminal state need only be tested when the goal is satisfied (i.e., the only choice point in applying the macro-operator is when its entire body has executed and a decision must be made whether the subgoal is satisfied or whether to iterate again). Therefore all the operators in the body of the macro-operator can be applied directly without any decision testing. This is one benefit of "homogeneous" macro-operators[33]. To see how this benefit results from having "homogeneous" macro-operators, see Chapter 6.

Another benefit arises when a terminal state implies the macro-operator's goal. Any variablized parameters in the macro-operator can be maximized to achieve the goal, since it is guaranteed to be the only goal of this macro-operator. For instance if an object, **X**, is equal to **0** in the precondition list of the macro-operator, is equal to **8** in the goal list of the macro-operator, and can be incremented by **1** thru **4** by a variablized argument of an operator in the macro-operator; then that variablized argument should be maximized to **4** to achieve the goal in fewer states. This allows the variablized parameters to be maximized for any macro-operator associated with either a preparation or critical subproblem. This heuristic can cause problems when solving the remainder subproblems because the terminal state can be passed over, see Chapter 9. A simple planning engine could have been added to the **Sojourner** problem solver to aid in the location of the terminal state in the remainder subproblem. But since this **only** can occur in remainder subproblems (or the subproblem which contains this problem's actual terminal state) it has presently not been included in the system.

[33]A "homogeneous" macro-operator is a macro-operator whose domain is a "homogeneous" subproblem.

3.1.3. The State Space Search Problem Solver

Breadth-first state space search is used by my system in two different ways. One way is when no problem reduction schemas or macro-operators *should* be applied to some subproblem, a straight forward breadth-first search of the entire state space of that subproblem is performed until a state is generated which implies the subgoal of the subproblem. The other way is when a macro-operator *should* be applied (i.e., the initial and terminal states of the new problem imply the invariant) but the macro-operator is not applicable (i.e., the initial state does not imply the preconditions) then a breadth-first search is performed until a state is generated which implies the subgoal of the subproblem or implies the preconditions of the macro-operator. Notice that both of these breadth-first searches only search the subproblem's state space (i.e., they only generate states which imply this subproblem's invariant).

3.1.4. The Combined Use of Problem Reduction Schemas and Macro-operators

How the entire problem solver solves a new problem is now illustrated. Suppose that the initial state of the new problem occurred in the preparation subproblem and the terminal state of the new problem occurred in the remainder subproblem of a stored problem reduction schema[34]. If there is no macro-operator associated with the preparation subproblem, then the problem solver would recurse on this new problem and look for an applicable problem reduction schema. If the macro-operator associated with the preparation subproblem is applicable in the initial state, then its loop body is executed. If the macro-operator is not applicable then the system breadth-first searches for another state and then attempts to apply the macro-operator from the new state. At the end of each iterative application of the loop body, the macro-operator's subgoal is checked to see if it is implied by the new state. If the new state does not imply the subgoal, then the macro-operator iterates again. If it did imply the subgoal, then the transition operator from this totally defined state to the partially defined initial state of the critical subproblem is be chosen, bound, and executed.

[34]Bear in mind that this is just an illustration of one possible use of the problem reduction schema. Other possibilities are where the initial and terminal states fall respectively in the preparation and critical, the critical and remainder, the preparation and preparation, etc. The only restriction is that the subproblem in which the terminal state occurs does not precede the subproblem in which the initial state occurs. This is why they are referred to as problem reduction schemas instead of problem reduction operators (i.e., they can be applied in many ways).

Now if there is no macro-operator associated with the critical subproblem, a solution must be found for this subproblem via state space search. If the macro-operator associated with the critical subproblem is applicable in this particular critical subproblem initial state, then its loop body is executed. If the macro-operator is not applicable then the system breadth-first searches for another state and then attempt to apply the macro-operator from the new state. At the end of each iterative application of the loop body, the macro-operator's subgoal is checked to see if it is implied by the new state. If the new state does not imply the subgoal, the macro-operator iterates again. If it implies the subgoal, then the transition operator from this totally defined state to the partially defined initial state of the remainder subproblem is chosen, bound, and executed.

Now if there is no macro-operator associated with the remainder subproblem, then the problem solver would recurse on this new problem and look for an applicable problem reduction schema. If the macro-operator associated with the remainder subproblem is applicable in this particular remainder subproblem initial state, then its loop body is executed. If the macro-operator is not applicable then the system breadth-first searches for another state and then attempts to apply the macro-operator from the new state. The loop body is now executed iteratively. Between every operator in the loop body, the terminal state of the new problem is checked to see if it is implied by the new state. If the new state does not imply the subgoal, then the macro-operator executes another operator. If it does imply the subgoal, then the problem is solved.[35]

Sojourner combines a search of the problem reduction space with a search of the state space. It has been suggested elsewhere [??] that the use of multiple representations allows problems to be solved more easily. Not much work has yet been done in this area [???]. My system gives support to the theory that using multiple representations can aid in problem solving.

[35]As can be seen, when the given problem's terminal state is the subproblem's terminal state in the final subproblem there is the possibility of overshooting and therefore missing the terminal state. Overshooting the goal can **only** occur in remainder subproblems or the final subproblem. When this happens, the system must resort to searching for the terminal state in this subproblem's state space. A simple planning extension could have been added to lower the search cost in these cases.

3.2. Introduction to the Derivation of Problem Reduction Schemas and Macro-operators

The intuition behind the derivation of problem reduction schemas is to focus on the hardest subgoal. The problem can then be reduced into the preparation subproblem, the critical subproblem, and the remainder subproblem. The *preparation subproblem* solves the problem to a state where the hardest subgoal can be solved. The *critical subproblem* solves the hardest subgoal. The *remainder subproblem* solves the rest of the problem's subgoals while leaving the hardest subgoal satisfied throughout.

The technique on which my problem reduction process is based is called *critical reduction.* There is some portion of the solution which is solved first[36]. All solutions to this problem must pass through this portion of the solution graph (i.e., the narrows of the graph, see [Amarel 68]). This frequently corresponds to the most constrained portion of the solution graph (i.e., the hardest subgoal). The critical reduction technique uses this critical portion of the graph to divide the problem's solution into the preparation subproblem (i.e., preparing to solve the critical subproblem), the critical subproblem, and the remainder subproblem (i.e., solving the rest of the problem). This technique is then recursively applied to the preparation and remainder subproblems.

Any problem which can be decomposed in this way has invariants which are *inherent* to each of its subproblems (i.e., some subgoal has its initial value in the preparation subproblem, its final value in the remainder subproblem, and neither of these values in the critical subproblem). Intuitively if something was achieved in the critical subproblem, then some object must have its old value at some point and its new value at some point. So there are some set of states (i.e., at least one) where the object's old value is invariant, some set of states (i.e., at least one) where the object's new value is invariant, and some set of states (i.e., possibly none) where the object having neither value is invariant. So if these invariants can be derived, they can then be used to determine the critical subproblem, and thereby defines the critical reduction. This method of using invariants to define the

[36]The hope is that it was solved first for a reason as opposed to by coincidence. Throughout this thesis, "solved first" refers to the "planning order" as opposed to the execution order. "Planning order", refers to the specific order which my system would plan the re-enactment of this solution path. The method for deriving this order is given in Chapter 5.

critical reduction is called *invariant reduction.*

Another benefit of deriving problem reduction schemas is that they can be used as a means of determining when it is good to create macro-operators. The problem reduction process described above recurses until no further problem reduction schemas can be formed. When the problem reduction process fails, it signifies that no subgoal's solution can be separated from all the other subgoals' solutions (i.e., the remaining portion of the solution path solves the rest of the subproblems together as a single entity). What this indicates is that a "homogeneous" portion of the solution has been found (i.e., that this portion of the solution is only solving this one group of subgoals and they should be treated as a single subgoal). This signals that a macro-operator should be created for solving this portion of the problem.

3.3. An Overview of the Derivation of Problem Reduction Schemas and Macro-operators

A flow diagram of the derivation (i.e., learning) process can be seen in Figure 3-8 and its corresponding ALGOL description is given in Figure 3-9. The input to the derivation system is a problem description and its associated solution path. The output of the system is a parse tree of problem descriptions[37] and their associated macro-operators. At each interior node of this parse tree, a problem reduction schema is represented, which decomposes the problem at this node into three subproblems, any of which can be empty[38]. At each leaf node of this parse tree which is not an empty problem, a macro-operator or a single state space operator is represented, whose domain[39] is the problem represented at this node. The problems represented at parse tree nodes are partial descriptions (i.e., sets of problems). The two major subprocesses of this derivation system are as follows:

1. Derivation of Problem Reduction Schemas

[37]The relationship between a parse tree of problem descriptions and problem reduction schemas is described at the end of this section.

[38]An empty subproblem is defined as having an initial state that implies its terminal state.

[39]A macro-operator's domain is the subproblem in which the macro-operator should be applied. Remember that when macro-operator *should* be applied and when it *can* be applied are two separate concepts.

2. Derivation of Macro-operators

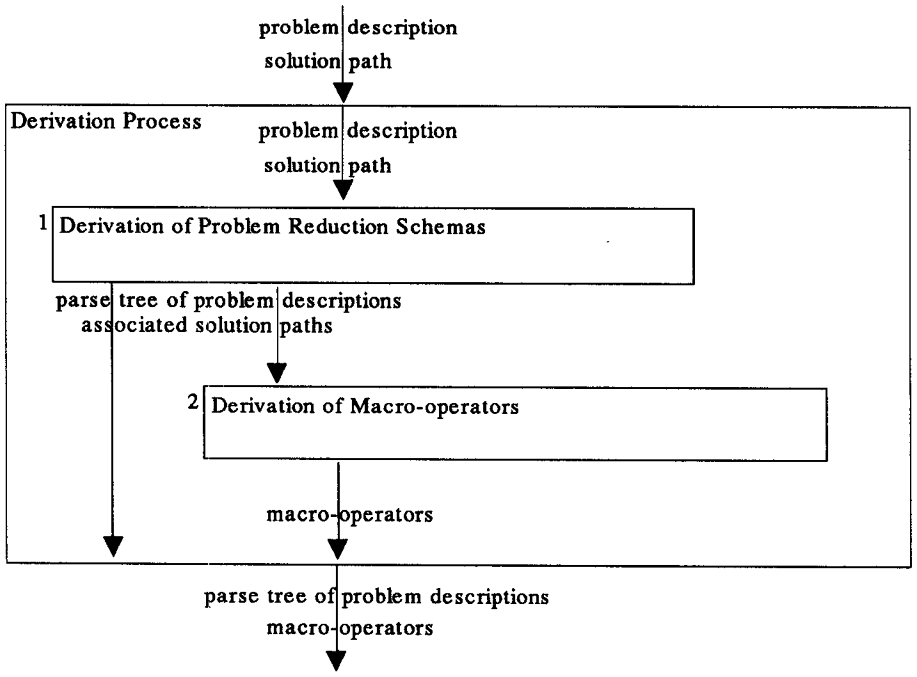

problem description
solution path
Derivation Process
problem description
solution path
1 Derivation of Problem Reduction Schemas
parse tree of problem descriptions
associated solution paths
2 Derivation of Macro-operators
macro-operators
parse tree of problem descriptions
macro-operators

Figure 3-8: Flow Diagram of the Derivation Algorithm

The flow diagram will now be examined step by step.

```
Derivation-Process(PROB-DESC,SOLUTION,REDUCTIONS,MACROS)
begin
  Derive-Problem-Reduction-Schemas(PROB-DESC,SOLUTION,REDUCTIONS,SOLUTIONS),
  Derive-Macro-Operators(PROB-DESC,SOLUTIONS,REDUCTIONS,MACROS),
end
```

Figure 3-9: Algorithm for the Derivation Process

3.3.1. Derivation of Problem Reduction Schemas and Macro-operators

A problem description and its associated solution path (i.e., training instance) are given as the inputs to the system. They are sent first to the Derivation of Problem Reduction Schemas process.

3.3.1.1. Process 1: Derivation of Problem Reduction Schemas

The Derivation of Problem Reduction Schemas process is responsible for deriving the parse tree of problem descriptions. The input to this process is the problem description and its associated solution path. The output of this process is a parse tree of problem descriptions and their associated portions of the original solution path. This process always succeeds. The parse tree of problem descriptions and their associated solution paths are now sent to the Derivation of Macro-operators process. The Derivation of Problem Reduction Schemas process is described in Section 3.3.2.

3.3.1.2. Process 2: Derivation of Macro-operators

The Derivation of Macro-operators process is responsible for deriving the macro-operators. It accomplishes this by creating iterative macro-operators when the derivation of problem reduction processes fails (which signals a "homogeneous"[40] portion of the solution path). The input to this process is a parse tree of problem descriptions and their associated solution path. The output of this process is a macro-operator for each leaf node of the parse tree whose associated solution path can be viewed as an iterative macro-operator. When this process has been applied to every leaf node of the parse tree of problem descriptions, the entire parse tree of subproblem descriptions and their associated macro-operators are output by the system. The Derivation of Macro-operators process is described in Section 3.3.3.

Now a more detailed look will be taken at the processes which derive the problem reduction schemas and the macro-operators. The M&C problem domain will be used to illustrate these processes. The description of the M&C problem class used in this chapter can be seen in Figure 3-1. Specifically, the 8-4 M&C problem (i.e., where the number of missionaries and cannibals, **n**, is 8 and the capacity of the boat, **k**, is 4) and the problem reduction schemas and macro-operators which can be derived with the training instance given in Figure 3-2 will be examined.

[40]An homogeneous subproblem is only concerned with solving a single subgoal, or a group of subgoals which should be solved as a single entity.

3.3.2. Derivation of Problem Reduction Schemas

A flow diagram of the problem reduction subsystem can be seen in Figure 3-11 and its corresponding ALGOL description is given in Figure 3-12. This is a more detailed description of the Derivation of Problem Reduction Schemas process discussed previously in section 3.3.1.1. Specifically it is a description of invariant reduction which is the method used to automate critical reduction.

The intuition behind invariant reduction is that invariants can be used to identify subproblems of the original problem. Then the critical reduction grammar can be matched to these subproblems therefore assigning them to a parse. Then the entire process can be repeated using the subproblems as the new problems. The *critical reduction grammar* is given in Backus-Naur form in Figure 3-10 and is based upon the preparation - critical - remainder decomposition discussed earlier.

```
PROB   → PREP OP CRIT OP REMAIN
PROB   → PREP OP REMAIN
PROB   → SETOP
PROB   → {}41
OP     → op
PREP   → PROB
REMAIN → PROB
CRIT   → SETOP
SETOP  → SETOP OP
SETOP  → OP
```

Figure 3-10: Critical Reduction Grammar

The Derivation of Problem Reduction Schemas process takes as input the problem description and its associated solution path (i.e., training instance). The output of this process is a parse tree of problem descriptions and their associated solution paths (i.e., subportions of the training instance).

[41] This represents an empty subproblem.

The Derivation of Problem Reduction process is itself made up of eight subprocesses. These subprocesses are as follows:

1. Derivation of the Critical State Component Literal
2. Derivation of the Invariants
3. Decomposition of the Solution Path
4. Recursion
5. Inter-Reduction Generalization
6. Intra-Reduction Generalization
7. Derivation of Subgoals
8. Reformulation of Subgoals

The flow diagram will now be examined step by step.

3.3.2.1. Process 1: Derivation of the Critical State Component Literal

The Derivation of the Critical State Component Literal process finds the state component literal[42] of the problem which is solved first (i.e., is assigned its final value defined by the terminal state) and remains solved throughout the rest of the solution path[43]. This state component literal is called the critical state component literal. The critical state component literal is derived through a backward search from the terminal state until the state component literal with the above attribute is found. The critical state component literal will be used to divide the problem and its solution path into three subproblems and their associated solution paths. The input to this process is the problem description and its associated solution path. The output of this process is the critical state component literal which is used by the Derivation of the Invariants process. If some state component literal is not solved first (i.e., more than one state component literal is solved simultaneously), then the process currently fails and the entire problem becomes the critical subproblem (i.e., the preparation and remainder subproblems are empty) and the flow of control

[42]A state component literal can be viewed at this point as an object in the state description language. It is formally defined in Chapter 4.

[43]This is the method for deriving the "planning order", mentioned previously. It is defined in more detail in Chapter 6.

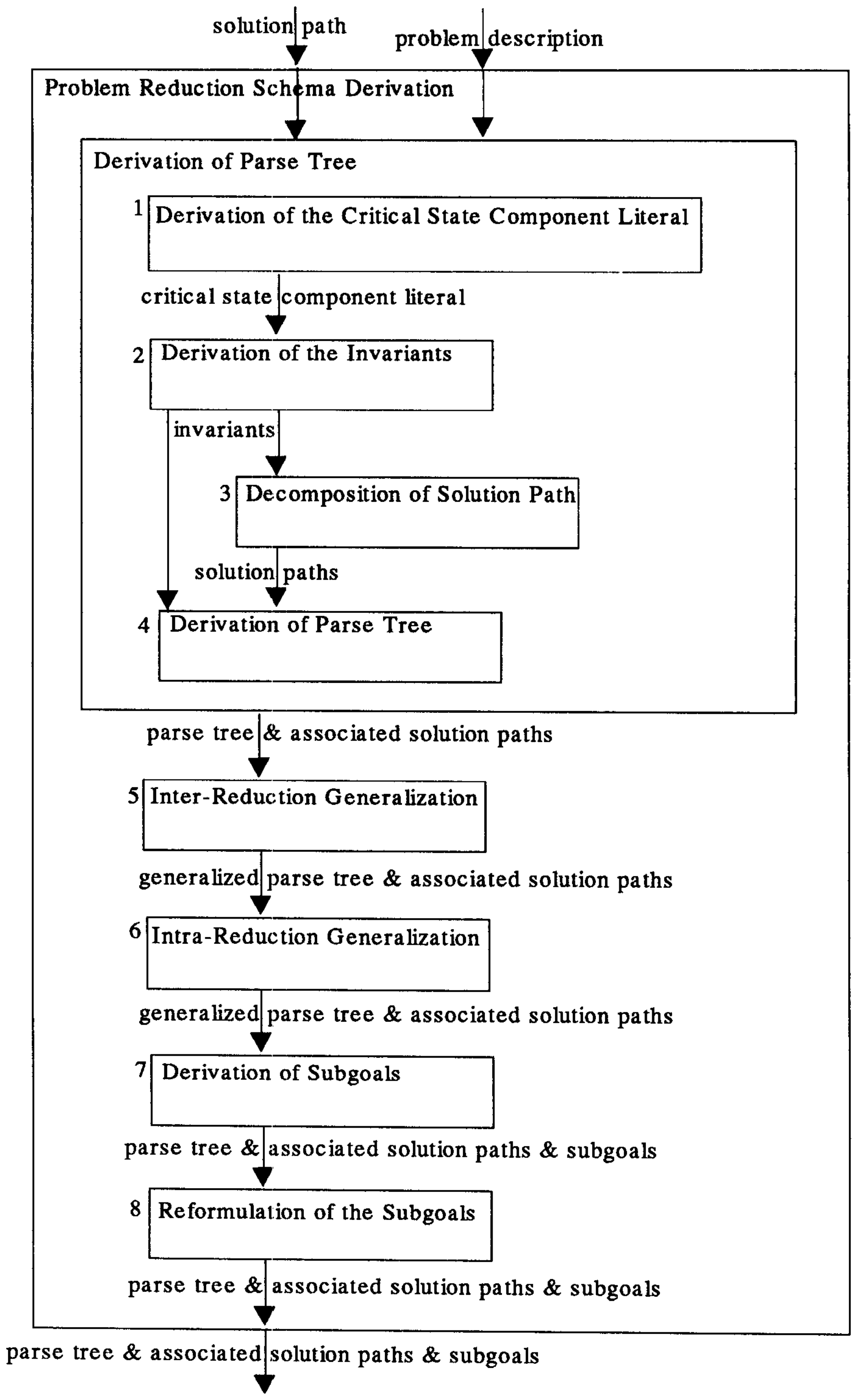

solution path
problem description
Problem Reduction Schema Derivation
Derivation of Parse Tree
1 Derivation of the Critical State Component Literal
critical state component literal
2 Derivation of the Invariants
invariants
3 Decomposition of Solution Path
solution paths
4 Derivation of Parse Tree
parse tree & associated solution paths
5 Inter-Reduction Generalization
generalized parse tree & associated solution paths
6 Intra-Reduction Generalization
generalized parse tree & associated solution paths
7 Derivation of Subgoals
parse tree & associated solution paths & subgoals
8 Reformulation of the Subgoals
parse tree & associated solution paths & subgoals
parse tree & associated solution paths & subgoals

Figure 3-11: Flow Diagram of Problem Reduction Schema Derivation Process

```
Derive-Problem-Reduction-Schemas(PROB-DESC,SOLUTION,REDUCTIONS,SOLUTIONS)
begin
  Derive-Parse-Tree(PROB-DESC,SOLUTION,PARSE-TREE-1,SOLUTIONS)
  Inter-Reduction-Generalization(PARSE-TREE-1,SOLUTIONS,PARSE-TREE-2)
  Intra-Reduction-Generalization(PARSE-TREE-2,SOLUTIONS,PARSE-TREE-3)
  Derivation-of-Subgoals(PARSE-TREE-3,SOLUTIONS,PARSE-TREE-4),
  Reformulation-of-Subgoals(PARSE-TREE-4,REDUCTIONS)
end

Derive-Parse-Tree(PROB-DESC,SOLUTION,PARSE-TREE,SOLUTIONS)
begin
  Derive-Critical-State-Component-Literal(PROB-DESC,SOLUTION,CRIT-STATE-COMP-LIT),
  Derive-Invariants(PROB-DESC,SOLUTION,CRIT-STATE-COMP-LIT,INVARIANTS),
  Decomposition-of-Solution-Path(INVARIANTS,SOLUTION,SOLUTIONS),
  .
  for each new solution path
  .
  Derive-Parse-Tree(PROB-DESC-I,SOLUTION-I,PARSE-TREE-I,SOLUTIONS-I)
  .
  compose parse-trees and union solution sets
  .
end
```

Figure 3-12: Algorithm for Problem Reduction Schema Derivation Process

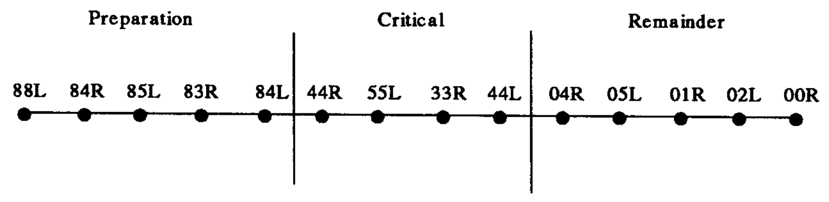
goes to the Derivation of Macro-operator process. If a critical state component literal is found, then it along with the problem description and the solution path, are now sent to the Derivation of the Invariants process. In the 8-4 M&C problem with our training example, the critical state component literal is **ML**.

3.3.2.2. Process 2: Derivation of the Invariants

The Derivation of the Invariants process derives the invariants for the three subproblems. These invariants are used throughout the Derivation of the Problem Reduction Schemas and Macro-operators processes. The invariants are derived as follows. The remainder invariant is the critical state component literal set to its value in the terminal state. The preparation invariant is the critical state component literal set to its value in the initial state. The critical invariant is the constraint that the critical state component literal not have the values specified in the preparation invariant or the remainder invariant. The input to this process is the critical state component literal and the solution path. The output of this process is the invariants for the three subproblems. This process never fails because these three invariants always exist (i.e., any operator which achieves something will have inherent invariants). In the basis case there is one state (i.e., the initial state) in the preparation subproblem, there is one state (i.e., the terminal state) in the remainder subproblem, and there is no states in the critical subproblem (i.e., a single state space operators moves directly from the initial state to the terminal state). The three invariants, the problem description, and the solution path are now sent to the Decomposition of the Solution Path process. In the 8-4 M&C problem with our training example, the preparation, critical, and remainder invariants are **ML(s)=8**, **ML(s)≠8** $\wedge$ **ML(s)≠0**, and **ML(s)=0** respectively. This set of invariants is not very useful at the moment since their values are very specific. These invariants will be generalized into a more useful form in Process 5 and Process 6.

3.3.2.3. Process 3: Decomposition of the Solution Path

The Decomposition of the Solution Path process derives the subportions of the solution path which correspond to the three subproblems (i.e., the preparation solution path, the critical solution path, and the remainder solution path). The solution path must be decomposed so that the previous processes can recurse on the preparation and remainder subproblems. The decomposition is derived by an examination of each state on the solution path. The invariants are used to test these states to determine in which subproblem each state belongs. Notice that each state must belong to one and only one subproblem, but all the states within a subproblem may not be "inter-connected". Two states in a subproblem are "inter-connected" if there exists a path between the two states which only contains states from the same subproblem. At this point a test must be done to assure that all the states within one subproblem are "inter-connected". The inputs to this process are the invariants and the solution path. The outputs of this process are the three subportions of the solution path corresponding to the three invariants. It is possible for any one of the solution paths to be empty which corresponds to an empty subproblem. This process can fail if the states within a subproblem are not "inter-connected". If this process fails, the entire problem becomes the critical subproblem (i.e., the preparation and remainder subproblems are empty) and the flow of control goes to the Derivation of Macro-operator process. In the basis case there are no states in the critical subproblem, but the decomposition still succeeds. Notice that any subproblem can be empty. The three solution paths, the invariants, and the problem description are now sent to the Recursion process. In the 8-4 M&C problem the three subportions of the solution path are shown in Figure 3-13.



Where XYZ : X = # of missionaires
Y = # of cannibals
Z ∈ {L,R} the bank of the river which the boat is on

Parameters : n = 3
k = 2

Figure 3-13: 8-4 Missionaries and Cannibals Problem Solution Path with Decomposition

3.3.2.4. Process 4: Recursion

At this point the Recursion process causes the "creation of the parse tree" processes to be recursed on for the preparation and remainder subproblem descriptions and their associated solution paths. The results of all these recursive calls is a parse tree of problem descriptions. When this recursion has terminated, the entire parse tree of problem descriptions and their associated solution paths are sent to the Inter-Reduction Generalization process. In the 8-4 M&C problem when the recursive step calls the "creation of the parse tree" processes on the preparation and remainder subproblems, they both fail (i.e., the parse tree of problem descriptions has a depth of 1). The recursive step does succeed in the Towers of Hanoi (TofH) problem class[44]. An example of this can be seen in the Appendix D.

3.3.2.5. Process 5: Inter-Reduction Generalization

The Inter-Reduction Generalization process generalizes the invariants at each level of recursion in the parse tree so that they correspond to the problem class instead of just this particular problem. A problem class consists of the set of problems which differ only in their initial and terminal states and the value of their problem parameters. Problem parameters are used to group a set of similarly structured but differently sized state space graphs together. For instance the problem parameters in the M&C problem class are the number of missionaries and cannibals (i.e., **n**) and the capacity of the boat (i.e., **k**).

The Inter-Reduction Generalization process is accomplished by generalizing over the multiple reduction levels of the parse tree of problem descriptions. Specifically this generalization is derived by using back-propagation over a proof that the remainder invariants, at each level of remainder recursion in the parse tree, are serializable (i.e., the invariants of the right most branch at each level of the parse tree are serializable). Serializability has been proven to be a necessary and sufficient requirement for the critical reduction technique to succeed, see Section 5.3. This requirement of the problem solver is used to pre-define a bias for the generalization by creating a proof schema for serializability, which is given to the system. Therefore the generalization (through back-propagation) is performed over a theory of the problem solver as opposed to a theory of the problem domain. For details on the use of proof schemas and their annotations see Chapter 7.

The input to this process is the parse tree of problem descriptions. The output of this process is a parse tree of problem descriptions which contains generalized invariants. If the invariants cannot be generalized, than the original parse tree is returned. The parse tree of problem descriptions with the

[44]The Towers of Hanoi problem class consists of three pegs A, B, C and **n** disks of different sizes {1, 2, . . . n}. Disks can be stacked on pegs; they can be moved from peg to peg one at a time. Only the top disk on a peg can be moved, but it can never be placed on top of a smaller disk. Initially, all **n** disks are at their specified initial peg locations; with the smallest disk at a peg on top, then next smallest disk at a peg next to the top and so on. Find the shortest sequence of operators for transferring all the disks to their specified destination peg locations.

generalized invariants, and their associated solution paths are now sent to the Intra-Reduction Generalization process. In the 8-4 M&C problem the invariants cannot be generalized by this method because the parse tree of problem descriptions has a depth of 1. The next process (i.e., Intra-Reduction Generalization) is able to generalize these invariants. Chapter 7 describes the generalizations performed by this process. Appendix D also contains an example of this type of generalization in the TofH domain.

3.3.2.6. Process 6: Intra-Reduction Generalization

The Intra-Reduction Generalization process generalizes the invariants at each level of recursion in the parse tree of problem descriptions. Each level of recursion in the parse tree of problem descriptions, by definition, partitions the set of states in the state space graph by means of the invariants. The Intra-Reduction Generalization process generalizes the three invariants at every level of recursion by back-propagating over a proof tree of the definition of a partition. This process also uses the back-propagation over proof schemas method. The method for using proof schemas is described briefly in Section 3.3.2.5 above and more completely in Chapter 7. The input to this process is the parse tree of problem descriptions. The output of this process is a parse tree of problem descriptions which has generalized invariants. If this process cannot generalize any of the invariants, then it returns the original parse tree. This new parse tree of problem descriptions and their previously derived associated solution paths are now passed to the Derivation of Subgoals process. In the 8-4 M&C problem the three invariants **ML(s)=8**, **ML(s)≠8** $\wedge$ **ML(s)≠0**, and **ML(s)=0** are generalized to the invariants **ML(s)=n**, **ML(s)≠n** $\wedge$ **ML(s)≠0** $\wedge$ **ML(s)=CL(s)**, and **ML(s)=0**, respectively. The **8** has been generalized to an **n** in all the invariants and the additional constraint, **ML(s)=CL(s)**, has been added to the critical invariant. Besides the variablization of the value **8**, the critical invariant has actually become more specialized!

3.3.2.7. Process 7: Derivation of the Subgoals

The Derivation of the Subgoals process derives subgoals for the subproblems at each level of recursion in the parse tree. When the problem reduction parse was derived, the exact subgoals from the subproblem's solution path were used. The subgoals in the final problem reduction schemas must be more general, if it is to be applicable in a large number of situations.

New generalized subgoals must be derived for the subproblems. These new subgoals are used as a partial description of the set of terminal states for their associated subproblems. They are derived by a one step back-propagation of the subsequent subproblem's invariant through the connecting variablized operator (i.e., the transition operator) to this subproblem's invariant. The back-propagation in this process only consists of the collection of the following formulae: the invariant at the state being back-propagated from, the invariant at the state being back-propagated to, and the description of the variablized operator being back-propagated through. The next process actually processes the back-propagation. No subgoal can be derived for the remainder subproblems because its purpose is not to move into a next subproblem but to achieve the particular terminal state of this particular subproblem.

The input to this process is the parse tree of problem description, and their associated solution paths. The output of this process is the set of formulae defining the subgoals for the subproblems at each level of recursion in the parse tree. This procedure never fails because subgoals for these subproblems can always be derived via back-propagation over the solution path. The parse tree of problem descriptions, and their associated subgoals and solution paths are now sent to the Reformulation of Subgoals process. In the 8-4 M&C problem the invariant for the preparation subproblem is **ML(s)=n**. The invariant for the critical subproblem is **ML(s)≠n, ML(s)≠0, ML(s)=CL(s)**. The operator between the critical subproblem and the preparation subproblem has the precondition list and change list respectively: **[ML(s_x)≥MT, CL(s_x)≥CT, B(s_x)='L', 0<MT(s_x)+CT(s_x)≤k]** and **[ML(s_y)=ML(s_x)-MT, CL(s_y)=CL(s_x)-CT, B(s_y)='R']**. The subgoal derived for the preparation subproblem is **[ML(0[45])≠n], ML(0)≠0, ML(0)=CL(0), B(1)='L', MT+CT≤k, 0<MT+CT, CL(1)≥CT, ML(1)≥MT, B(0)='R', CL(0)=CL(1)-CT, ML(0)=ML(1)-MT, ML(1)=n**. This subgoal is just the collection of the previously mentioned formulae with their states bound. Instead of using the explicit states the system represents states using a unique state number.

[45]The states are represented here as time stamps which can be seen as expanding into a full state description.

3.3.2.8. Process 8: Reformulation of the Subgoals

Preparation Subgoal

goal([ML(0) $\neq$ n, ML(0) $\neq$ 0, ML(0) = CL(0), B(1) = 'L', MT+CT $\leq$ k,
0 < MT+CT, CL(1) $\geq$ CT, ML(1) $\geq$ MT, B(0) = 'R',
CL(0) = CL(1)-CT, ML(0) = ML(1)-MT, ML(1) = n])

Preparation Reformulated Subgoal

goal(*or*([B(s) = 'L', ML(s) = n, ***max***(1, n-k) $\leq$ CL(s) $\leq$ ***min***(n, n+k-2, n+(k/2)-1), n $\geq$ 2],
[B(s) = 'L', ML(s) = n, CL(s) = n, n $\geq$ 2, k $\geq$ 2])),

Figure 3-14: Subgoal and Reformulated Subgoal for the Preparation Subproblem

The Reformulation of the Subgoals process reformulates the subgoals so that 1) they are testable in a problem state and 2) they are simplified for more efficient pattern matching. Testable in a problem state means that no other state besides the state being back-propagated to is mentioned and no operator arguments are mentioned. The need for this type of processing arises from the fact that up to now back-propagation over many-to-one operators has always been done by giving the system an explicit set of inverse operators[46]. Since this research's basic premise is reformulation, it should not require the initial formulation to meet this strict constraint. Thus a method has been devised for doing back-propagation over many-to-one operators which is a combination of a subset of the Fourier-Motzkins algorithm and some preprocessing and simplification transformations. This method is called the Variable Elimination algorithm. For a detailed description of this algorithm, see Chapter 8. The input to this process is the subgoals derived in the previous process. The output of this process is the reformulated subgoals. This process never fails because it is complete and sound. If the subgoals were already testable and cannot be simplified, then the original subgoals are returned. In the 8-4 M&C problem the subgoals derived are shown in Figure 3-5 as the terminal state descriptions for the preparation and critical subproblems. The preparation subgoal shown above is reformulated into *or*(**[ML(s)=n, CL(s)$\geq$max(1,n-k), CL(s)$\leq$min(n,n+k-2,n+(k/2)-1),**

[46]These operators are not strictly inverses in the mathematical sense, because the forward operators are not restricted to being one-to-one.

B(s)="L", n≥2, k≥1], [ML(s)=n, CL(s)=n, B(s)="L", n≥2, k≥2]). The original subgoal and the reformulated subgoal are shown together in Figure 3-14. Notice that all the variables in the reformulated subgoal are testable in a single state.

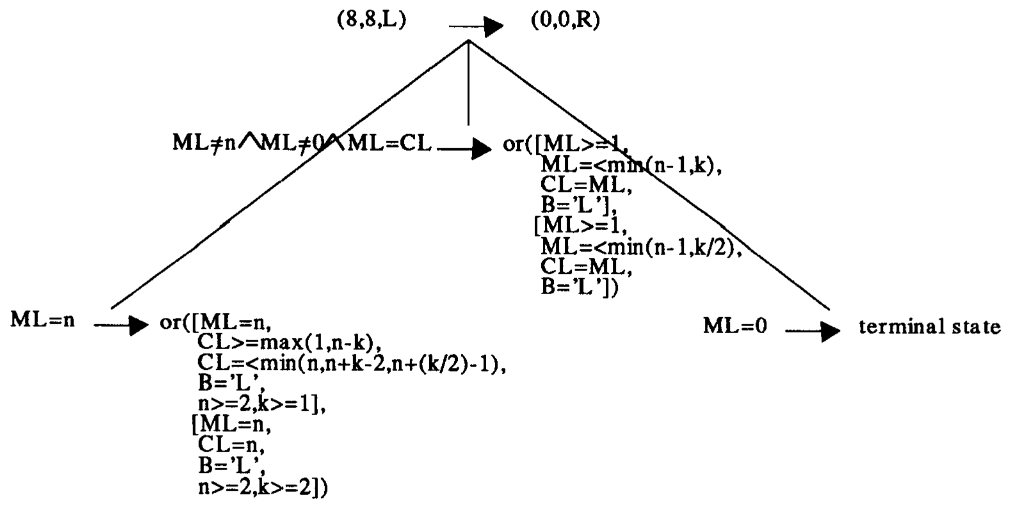

(8,8,L) → (0,0,R)
ML≠n∧ML≠0∧ML=CL → or([ML>=1, ML=<min(n-1,k), CL=ML, B='L'], [ML>=1, ML=<min(n-1,k/2), CL=ML, B='L'])
ML=n → or([ML=n, CL>=max(1,n-k), CL=<min(n,n+k-2,n+(k/2)-1), B='L', n>=2,k>=1], [ML=n, CL=n, B='L', n>=2,k>=2])
ML=0 → terminal state

Figure 3-15: Parse Tree of Problem Descriptions for the Elementary 8-4 M&C Problem

A parse tree of problem descriptions has been created by the three subproblems at each level of recursion. The parse tree created by the system for the elementary 8-4 M&C problem can be seen in Figure 3-15. The three subproblems contain the same problem description[47] as the original problem except that their initial states are the respective invariants and their terminal states are the respective subgoals. The subgoal of the final subproblem is left as a variable which can be filled in by the terminal state of a new problem actually being solved using this problem reduction schema. This parse tree of problem descriptions and their associated solution paths are returned by this subsystem.

The parse tree of problem descriptions actually contains the problem reduction schemas. At each interior node in the parse tree a problem reduction schemas is represented. The leaves of the parse tree represent either empty subproblems, subproblems whose associated solution paths

[47]A problem description contains the state description and the operator descriptions. For a complete definition see Chapter 4.

consists of a single state space operator, or subproblems whose associated solution paths consists of multiple state space operators and therefore should have macro-operators derived for them.

3.3.3. Derivation of the Macro-operators

A flow diagram of the Derivation of Macro-operators process can be seen in Figure 3-16 and its corresponding ALGOL description is given in Figure 3-17. This is a more detailed description of the Derivation of Macro-operators process discussed in the Section 3.3.1.2. The Derivation of the Macro-operator process takes as input the parse tree of problem descriptions and their associated solution paths. This process is repeated for each leaf node of the parse tree of problem descriptions which has an associated solution path which consists of multiple state space operators. This process produces an iterative macro-operator for traversing through that subproblem towards its subgoal. It is a generate and test process based upon generating a candidate macro-operator and then testing it to verify that it is in fact iterative. The Derivation of Macro-operator process is itself made up of three subprocesses. These subprocesses are:

1. Derivation of a Candidate Repetition

2. Derivation of a Candidate Macro-operator

3. Verification of a Candidate Macro-operator

The flow diagram will now be examined step by step.

3.3.3.1. Process 1: Derivation of a Candidate Repetition

The Derivation of a Candidate Repetition process finds a candidate repetition. The next process attempts to make it into a macro-operator. A candidate repetition is a sequence of variablized operators. The candidate repetition is derived by a partial match on the sequence of operators in the solution path (i.e., subportions of the solution path are partially matched against the rest of the solution path). This allows variablization of any argument which has different values on different iterations of the loop. The preconditions of the operator to be variablized are then used as constraints on the value of this argument. The input to this process is the problem description and its associated solution path. The output of this process is a candidate repetition. If this process has tried all possible repetitions including the entire operator sequence and has been unable to form an iterative macro-operator, then it does not create any macro-operator. The candidate repetition, the

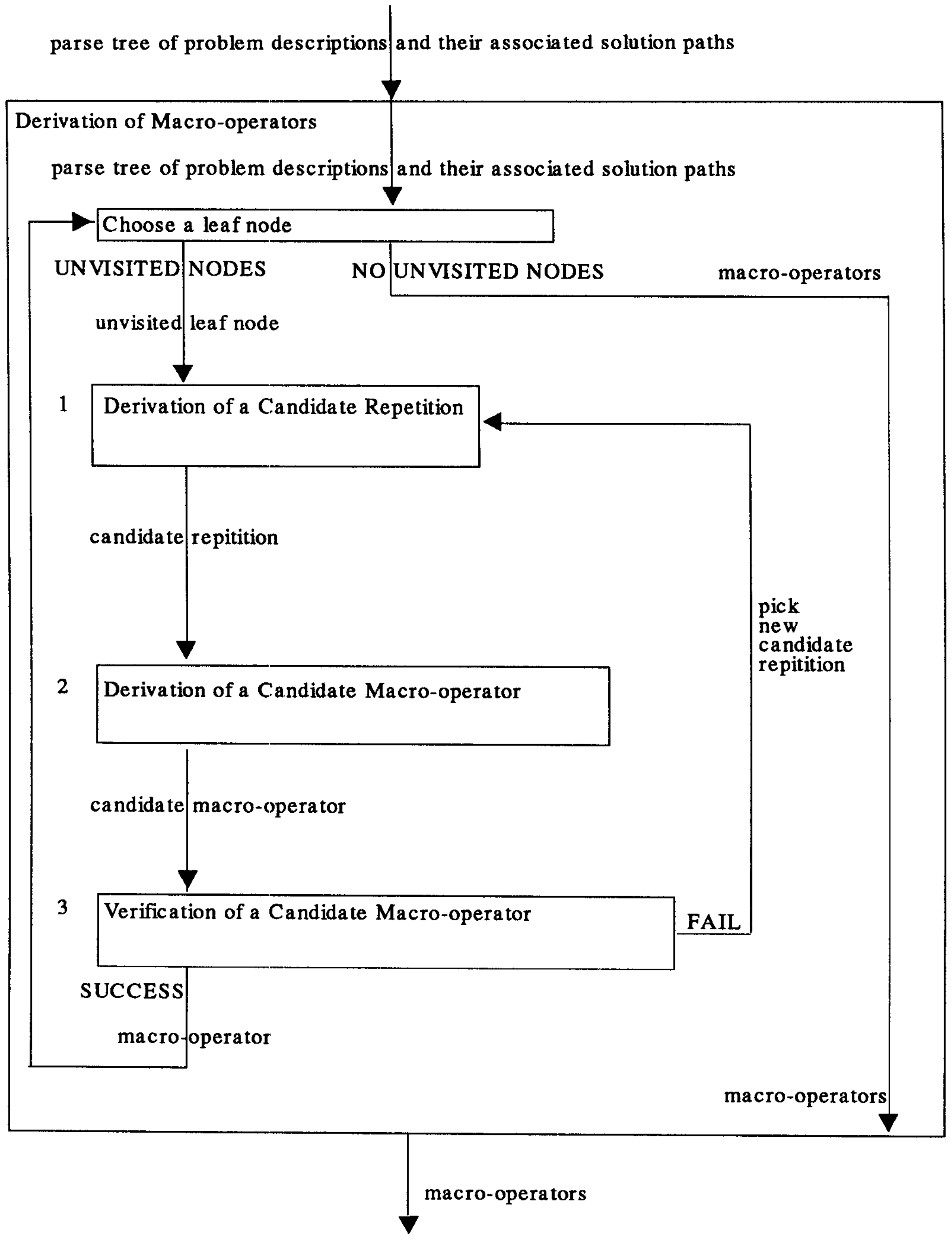

parse tree of problem descriptions and their associated solution paths
Derivation of Macro-operators
parse tree of problem descriptions and their associated solution paths
Choose a leaf node
UNVISITED NODES
NO UNVISITED NODES
macro-operators
unvisited leaf node
1
Derivation of a Candidate Repetition
candidate repitition
pick new candidate repitition
2
Derivation of a Candidate Macro-operator
candidate macro-operator
3
Verification of a Candidate Macro-operator
FAIL
SUCCESS
macro-operator
macro-operators
macro-operators

Figure 3-16: Flow Diagram of Macro-operator Derivation Process

problem description, and the solution path are sent to the Derivation of a Candidate Macro-operator process. The macro-operators derived for the 8-4 M&C problem are given in Figure 3-18. The candidate repetitions are given in the *operators*, *arguments*, *start*, and *definitions* slots.

```
Derive-Macro-operators(PROB-DESC,SOLUTIONS,PARSE-TREE,MACROS)
begin
  .
  LABEL-1:
  choose an unvisited leaf node and associated solution path else goto LABEL-3
  .
  LABEL-2:
  derive-candidate-repetition(PROB-DESC,SOLUTION,CAND-REPET)
  derive-candidate-macro-operator(PROB-DESC,CAND-REPET,CAND-MACRO-OPERATOR)
  verify-candidate-macro-operator(PROB-DESC,CAND-MACRO-OPERATOR,SUCCESS)
  if SUCCESS=true
    then goto LABEL-1
    else goto LABEL-2
  LABEL-3:
  .
  collect macro-operators
  .
end
```

Figure 3-17: Algorithm for Macro-operator Derivation Process

3.3.3.2. Process 2: Derivation of a Candidate Macro-operator

The Derivation of a Candidate Macro-operator process turns the candidate repetition into a candidate macro-operator. A candidate macro-operator must contain all the preconditions necessary for its application. The candidate macro-operator's preconditions are derived by obtaining a set of weakest preconditions for the candidate macro-operator via back-propagation by the Variable Elimination algorithm mentioned in Section 3.3.2.8. The input to this process is a candidate repetition and a problem description. The output of this process is a candidate macro-operator. This procedure never fails because if there exists a repetition, then a candidate macro-operator can always be derived. The candidate macro-operator, the problem description, and the solution path are now sent to the Verification of a Candidate Macro-operator process. In the 8-4 M&C problem

Preparation macro-operator

```
invariant(ML(s)=n),
precondition([B(s)='L',  ML(s)=n  1≤CL(s)≤ n,  k≥ 1]),
goal(or([B(s)='L',  ML(s)=n,  n≥2,  max(1, n-k)≤CL(s)≤min(n, n+k-2, n+(k/2)-1)],
        [B(s)='L',  ML(s)=n,  n≥2,  CL(s)=n,  k≥2])),
operators([trlr,trrl]),
arguments([[0,x1],  [0,1]]),
start(1),
definitions([0 < x1 ≤ min(k, CL(s))])
```

Critical macro-operator

```
invariant([ML(s)≠ n,  ML(s)≠ 0,  ML(s) = CL(s)]),
precondition([B(s) = 'R',  CL(s) = ML(s),  1≤ ML(s)≤ n-1.5,  k≥1]),
goal(or([B(s) = 'L',  1≤ML(s)≤ min(n-1, k),  CL(s) = ML(s)],
        [B(s) = 'L',  1≤ML(s) ≤ min(n-1, k/2),  CL(s) = ML(s)])),
operators([trrl,trlr]),
arguments([[2,2],  [1,1]]),
start(2),
definitions([])
```

Remainder macro-operator

```
invariant([ML(s) = 0]),
precondition([B(s) = 'R',  0≤CL(s)≤ n-1,  ML(s) = 0,  k ≥1]),
goal(terminal state),
operators([trrl, trlr]),
arguments([[0,1],  [0,x2]]),
start(1),
definitions([0 < x2≤ min(k, Cl(s))])
```

Figure 3-18: Macro-operators for the Missionaries and Cannibals Problem Class

the candidate macro-operators, which are derived, are shown in Figure 3-18. The *invariant*, *precondition*, and *subgoal* slots have been added to change the candidate repetition into a full-fledged candidate macro-operator.

3.3.3.3. Process 3: Verification of a Candidate Macro-operator

The Verification of a Candidate Macro-Operator process verifies that a candidate macro-operator can be repeated at least once. The verification is done by deriving the postconditions of the loop and proving that they imply the preconditions of the loop. The input to this process is the candidate macro-operator and the problem description. The output of this process is a verified macro-operator. If this process fails, the system looks for another candidate repetition. In the 8-4 M&C problem the preparation, critical, and remainder macro-operators are verified.

The parse tree of problem descriptions, which contains the problem reduction schemas and their associated macro-operators are returned by the system. This chapter gave an overview of the derivation and use of the problem reduction schemas and iterative macro-operators. The next chapter gives the definitions which will be needed in the more formal explanations of the derivation of problem reduction schemas and iterative macro-operators in the following two chapters.