INFORMATION TO USERS

This was produced from a copy of a document sent to us for microfilming. While the most advanced technological means to photograph and reproduce this document have been used, the quality is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help you understand markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure you of complete continuity.

2. When an image on the film is obliterated with a round black mark it is an indication that the film inspector noticed either blurred copy because of movement during exposure, or duplicate copy. Unless we meant to delete copyrighted materials that should not have been filmed, you will find a good image of the page in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed the photographer has followed a definite method in "sectioning" the material. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For any illustrations that cannot be reproduced satisfactorily by xerography, photographic prints can be purchased at additional cost and tipped into your xerographic copy. Requests can be made to our Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases we have filmed the best available copy.
JAPPINEN, HARRY JUHANI
A PERCEPTION-BASED DEVELOPMENTAL SKILL ACQUISITION SYSTEM.

THE OHIO STATE UNIVERSITY, PH.D., 1979
PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark ✓.

1. Glossy photographs ________
2. Colored illustrations ________
3. Photographs with dark background ________
4. Illustrations are poor copy ________
5. Print shows through as there is text on both sides of page ________
6. Indistinct, broken or small print on several pages ✓ throughout ________
7. Tightly bound copy with print lost in spine ________
8. Computer printout pages with indistinct print ________
9. Page(s) ________ lacking when material received, and not available from school or author ________
10. Page(s) ________ seem to be missing in numbering only as text follows ________
11. Poor carbon copy ________
12. Not original copy, several pages with blurred type ________
13. Appendix pages are poor copy ________
14. Original copy with light type ________
15. Curling and wrinkled pages ________
16. Other ____________________________________________________________________

University Microfilms International

300 N. ZEEB RD., ANN ARBOR, MI 48106 (313) 761-4700
A PERCEPTION-BASED DEVELOPMENTAL SKILL ACQUISITION SYSTEM

DISSERTATION

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of the Ohio State University

By

Harry Juhani Jappinen, dipl ins, M.Sc.

* * * * *

The Ohio State University

1979

Reading Committee:

Prof. S. Chandrasekaran
Prof. Richard Jagacinski
Prof. Norman Sondheimer
Prof. Lee White

Approved By

Adviser
Department of Computer and Information Science
ACKNOWLEDGEMENTS

I am extremely thankful to Dr. B. Chandrasekaran for his valuable guidance and consistent encouragement throughout the course of this work. He was kind enough to devote his valuable time, amidst his numerous engagements, to provide constructive criticism and continuing interest throughout this research.

I will remain ever grateful to Dr. Richard Jagacinski, Dr. Norman Sondheimer, and Dr. Lee White for their encouragement and helpful suggestions.

I would like to thank the staff of Ohio College Library Center and particularly the Instruction and Research Computer Center, the Ohio State University, for their friendship and financial support.

Special thanks are due to the United States Educational Foundation in Finland for granting me a Fulbright scholarship for one year of graduate studies in the United States. Wihuri Foundation, Helsinki Finland, provided me a generous personal grant for this dissertation research thus relieving many financial problems, for which I am ever grateful.

I wish to finally offer my gratitude to Sarah Ann Orrahood for helping with the English language and the partial typing of this dissertation.
VITA

August 21, 1943 . . . . . Born - Lohja, Finland

1972 . . . . . . . . . . dipl ins Degree
Dept. of Electrical Engineering
Technical University of Finland
Helsinki, Finland

Finnish State Computer Center
Helsinki, Finland

1974-1975 . . . . . Special Researcher
The Ministry of Finance
Helsinki, Finland

1976 . . . . . . . . . . M.Sc. Degree
Dept. of Computer and
Information Science
The Ohio State University
Columbus, Ohio

1976-1977 . . . . . Graduate Research Associate
Ohio College Library Center
Columbus, Ohio

1977-1979 . . . . . Graduate Research Associate
The Instruction and Research
Computer Center
The Ohio State University
Columbus, Ohio
PUBLICATIONS

1. "FAS-TO-COBOL Translator"
   Dipl ins Degree Thesis, April, 1972
   Dept. of Electrical Engineering
   Technical University of Finland
   Helsinki, Finland

2. "Improving the Efficiency of Computer Centers"
   (in Finnish)
   Hallinto 3, 1974
   Helsinki, Finland

FIELDS OF STUDY

Major Field: Computer and Information Science
   a. Artificial Intelligence (Prof. B. Chandrasekaran)
   b. Computational Linguistics (Prof. N. Sondheimer)
   c. Computer Architecture (Prof. M. T. Liu)
TABLE OF CONTENTS

ACKNOWLEDGEMENTS .............................................. ii
VITA .................................................................. iii
LIST OF TABLES ...................................................... viii
LIST OF FIGURES ..................................................... ix

Chapter
I. INTRODUCTION ..................................................... 1
   I.1 BACKGROUND AND MOTIVATION ......................... 1
      I.1.1 Inferred vs. Direct Behavior ....................... 2
      I.1.2 Planning vs. Skill Usage ............................ 3
   I.2 PROPOSED SKILL SYSTEM ................................... 7
      I.2.1 Formation of Skills: Advice-taking ................. 7
      I.2.2 Syntax and Semantics of Skills .................... 9
   I.3 SKILLED BEHAVIOR: AN EXAMPLE ....................... 13
   I.4 PSYCHOLOGICAL PERSPECTIVE ......................... 19
   I.5 ROBOTICS IN ARTIFICIAL INTELLIGENCE ............... 20
   I.6 OUTLINE OF CHAPTERS .................................... 25

II. OVERVIEW OF THE SYSTEM .................................... 27
   II.1 IMPLEMENTATION AND SIMULATION .................... 28
      II.1.1 The World, Sampo, and Primitives ............... 28
      II.1.2 Memory and Recalling .............................. 37
      II.1.3 Natural Language and Parsing .................... 43
      II.1.4 Development ........................................ 46
      II.1.5 System Structure and Simulation Environment .... 50
   II.2 BEHAVIORAL SIMULATION ............................... 58
      II.2.1 Initial Setting ...................................... 58
      II.2.2 Task #1 .............................................. 59
      II.2.3 Task #2 .............................................. 67
      II.2.4 Task #3 .............................................. 72
III. CONCEPTUAL MODEL

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>III.1 BEHAVIORAL ABSTRACTION</td>
<td>84</td>
</tr>
<tr>
<td>III.1.1 Logical Taxonomy of Concepts</td>
<td>86</td>
</tr>
<tr>
<td>III.1.2 Perceptual Memory</td>
<td>90</td>
</tr>
<tr>
<td>III.1.3 Propositional Memory</td>
<td>92</td>
</tr>
<tr>
<td>III.1.4 Orienting Maps</td>
<td>93</td>
</tr>
<tr>
<td>III.1.5 Requests, Questions, Claims, and Answers</td>
<td>95</td>
</tr>
<tr>
<td>III.1.6 Skills</td>
<td>95</td>
</tr>
<tr>
<td>III.1.7 Perceiving</td>
<td>99</td>
</tr>
<tr>
<td>III.1.8 Recalling</td>
<td>101</td>
</tr>
<tr>
<td>III.1.9 Monitoring</td>
<td>102</td>
</tr>
<tr>
<td>III.1.10 Fact Manipulation</td>
<td>103</td>
</tr>
<tr>
<td>III.2 DEVELOPMENTAL ABSTRACTION</td>
<td>110</td>
</tr>
<tr>
<td>III.2.1 Skill Modification</td>
<td>111</td>
</tr>
<tr>
<td>III.2.2 Skill Acquisition</td>
<td>114</td>
</tr>
<tr>
<td>III.2.3 Perceptual Development</td>
<td>115</td>
</tr>
<tr>
<td>III.2.4 Orienting Development</td>
<td>115</td>
</tr>
</tbody>
</table>

IV. COMPUTATIONAL MODEL

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>IV.1 BEHAVIORAL ABSTRACTION</td>
<td>118</td>
</tr>
<tr>
<td>IV.1.1 Logical Taxonomy of Concepts</td>
<td>119</td>
</tr>
<tr>
<td>IV.1.2 Perceptual Memory</td>
<td>120</td>
</tr>
<tr>
<td>IV.1.3 Propositional Memory</td>
<td>120</td>
</tr>
<tr>
<td>IV.1.4 Orienting Maps</td>
<td>122</td>
</tr>
<tr>
<td>IV.1.5 Requests, Questions, Claims, and Claims</td>
<td>123</td>
</tr>
<tr>
<td>IV.1.6 Skills</td>
<td>123</td>
</tr>
<tr>
<td>IV.1.7 Perceiving</td>
<td>131</td>
</tr>
<tr>
<td>IV.1.8 Recalling</td>
<td>133</td>
</tr>
<tr>
<td>IV.1.9 Monitoring</td>
<td>137</td>
</tr>
<tr>
<td>IV.1.10 Fact Manipulation</td>
<td>142</td>
</tr>
<tr>
<td>IV.2 DEVELOPMENTAL ABSTRACTION</td>
<td>149</td>
</tr>
<tr>
<td>IV.2.1 Skill Modification</td>
<td>149</td>
</tr>
<tr>
<td>IV.2.2 Skill Acquisition</td>
<td>165</td>
</tr>
<tr>
<td>IV.2.3 Perceptual Development</td>
<td>171</td>
</tr>
<tr>
<td>IV.2.4 Orienting Development</td>
<td>171</td>
</tr>
</tbody>
</table>

V. DISCUSSION AND CONCLUSIONS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>V.1 DEMARCATION BETWEEN SKILLED AND PLANNED BEHAVIOR</td>
<td>172</td>
</tr>
<tr>
<td>V.1.1 Exception Handling and Error Recovery</td>
<td>176</td>
</tr>
<tr>
<td>V.1.2 Planning</td>
<td>179</td>
</tr>
<tr>
<td>V.2 APPLYING THE PROPOSED SYSTEM</td>
<td>183</td>
</tr>
<tr>
<td>V.2.1 Where Does the Proposed System Apply?</td>
<td>183</td>
</tr>
<tr>
<td>V.2.2 How to Apply the Proposed System?</td>
<td>184</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>II.1</td>
<td>Sampo's Movements</td>
<td>30</td>
</tr>
<tr>
<td>II.2</td>
<td>Sampo's Perceptual Experts</td>
<td>37</td>
</tr>
<tr>
<td>II.3</td>
<td>Examples of Recalling</td>
<td>42</td>
</tr>
<tr>
<td>II.4</td>
<td>Sentence Types</td>
<td>43</td>
</tr>
<tr>
<td>II.5</td>
<td>Elementary Skills Taken for Granted</td>
<td>51</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>II.1</td>
<td>Simulated House</td>
<td>31</td>
</tr>
<tr>
<td>II.2</td>
<td>Example of a Use of a Reference-variable</td>
<td>42</td>
</tr>
<tr>
<td>II.3</td>
<td>Example of Parsing</td>
<td>44</td>
</tr>
<tr>
<td>II.4</td>
<td>System Organization</td>
<td>56</td>
</tr>
<tr>
<td>II.5</td>
<td>MONITOR Organization</td>
<td>57</td>
</tr>
<tr>
<td>II.6</td>
<td>LTOC in the Beginning of Simulation</td>
<td>74</td>
</tr>
<tr>
<td>II.7</td>
<td>LTOC After Task #1</td>
<td>75</td>
</tr>
<tr>
<td>II.8</td>
<td>LTOC After Task #2</td>
<td>76</td>
</tr>
<tr>
<td>II.9</td>
<td>Task #1</td>
<td>77</td>
</tr>
<tr>
<td>II.10</td>
<td>Task #2</td>
<td>79</td>
</tr>
<tr>
<td>II.11</td>
<td>Task #2 cont'd</td>
<td>81</td>
</tr>
<tr>
<td>II.12</td>
<td>Task #3</td>
<td>82</td>
</tr>
<tr>
<td>III.1</td>
<td>Skill as a Procedure</td>
<td>108</td>
</tr>
<tr>
<td>III.2</td>
<td>Perception as a Procedure</td>
<td>108</td>
</tr>
<tr>
<td>III.3</td>
<td>MONITOR as a Procedure</td>
<td>108</td>
</tr>
<tr>
<td>III.4</td>
<td>FACT-MANIPULATION as an Updating Procedure</td>
<td>109</td>
</tr>
<tr>
<td>III.5</td>
<td>FACT-MANIPULATION as an Answering Procedure</td>
<td>109</td>
</tr>
<tr>
<td>III.6</td>
<td>SKILL-MODIFICATION as a Procedure</td>
<td>116</td>
</tr>
<tr>
<td>III.7</td>
<td>SKILL-ACQUISITION as a Procedure</td>
<td>116</td>
</tr>
<tr>
<td>III.8</td>
<td>Perceptual Development as a Procedure</td>
<td>117</td>
</tr>
<tr>
<td>III.9</td>
<td>Orienting Development as a Procedure</td>
<td>117</td>
</tr>
<tr>
<td>IV.1</td>
<td>LTOC Before Skill Modification Examples</td>
<td>162</td>
</tr>
<tr>
<td>IV.2</td>
<td>LTOC After Example 3</td>
<td>162</td>
</tr>
<tr>
<td>IV.3</td>
<td>LTOC After Example 4</td>
<td>163</td>
</tr>
<tr>
<td>IV.4</td>
<td>LTOC After Example 5</td>
<td>163</td>
</tr>
<tr>
<td>IV.5</td>
<td>LTOC After Example 6</td>
<td>164</td>
</tr>
<tr>
<td>IV.6</td>
<td>LTOC After Example 8</td>
<td>164</td>
</tr>
<tr>
<td>IV.7</td>
<td>LTOC Before Skill Acquisition Examples</td>
<td>169</td>
</tr>
<tr>
<td>IV.8</td>
<td>LTOC After Example 9</td>
<td>169</td>
</tr>
<tr>
<td>IV.9</td>
<td>LTOC After Example 10</td>
<td>170</td>
</tr>
</tbody>
</table>
I. INTRODUCTION

I.1 BACKGROUND AND MOTIVATION

The traditional approach to the design of robot behavior in Artificial Intelligence can be stated as follows: for a given task a problem solver produces an ad hoc plan and then a monitor interprets and executes the plan and continuously checks its correctness. Control information for a robot's behavior is provided by symbolic representations of world states (a world model). Since the problem solver can freely manipulate the world model, no matter how complex a task, future states of the world can be projected in the planning phase, and the plan can be so organized that an optimal behavioral pattern results from it. On the other hand, unless the robot's subworld is static and strongly restricted, total reliance on a world model may result in serious efficiency problems since even the minutest details of behavior need to be explicitly determined in the plan. Furthermore, the maintaining of a consistent world model may pose harsh problems.

In contrast, humans are capable not only of the above kind of behavior -- we may call it inferred behavior -- but also of reacting directly, and a part of human knowledge of the world is stored in action form. Everyday human behavior
is a mixture of inferred and direct behavior. A plan determines only gross behavior while leaving control details in the care of earlier developed, task-specific action-schemes. Action-schemes receive control information directly from the trustworthy environment by using perceptions.

This work studies some aspects of direct behavior. We hope to gain insight into its advantages, disadvantages, and limitations of its use in robotics; and to determine how direct behavior might be interfaced with inferred behavior. We will first contrast inferred behavior with direct behavior. We will then set the two modes of behaving in the computational context and call them planned and skilled behavior, respectively. We will further outline the proposed skill system and provide an example of how the proposed system performs a task. At the end of this introduction we will give a short review of AI robotics.

1.1.1 Inferred vs. Direct Behavior

At least as far as human behavior is concerned, basically two means of creating a behavioral pattern for a given task exist:

1. A plan for behavior for the task is first inferred; then the plan is interpreted and executed.
(2) An earlier formed and stored (developed) task-specific action-scheme is activated. The action-scheme will then directly respond to stimuli from the environment and guide overt behavior.

A move in a board game, a trip, and the repair of a malfunction in a car represent paradigmatic examples of behavior of the first type. Riding a bicycle, walking along a familiar route, and skilled typing represent examples of the other. In the majority of cases, behavior seems to consist of a mixture of both.

Since knowledge representations and inference techniques are, among others, in the focal point of Artificial Intelligence research, it has followed naturally that inferred behavior has received the most attention in AI robotics.

1.1.2 Planning vs. Skill Usage

We set now the two modes of behaving in the computational context. Hence, to repeat, behavior for a given task can be created by:

(1) A problem solver, manipulating a symbolic representation of world states and operators to transform the states (world model), produces an operator sequence (plan) as a behavioral solution to
the task. A monitor then interprets and executes the plan and continuously checks its correctness.

(2) An earlier formed and stored task-specific skill is activated. In the skill, perception procedures, directly probing the states of the immediate surroundings, and recalling memory, probing recorded perceived experience, control movements. Skills do not employ inference on a symbolic world model.

From now on the terms 'plan', 'planning', and 'planned behavior' refer to the operational definition in (1), and 'skill' and 'skilled behavior' refer to the definition (2).

Traditionally, AI robotics has focused on planning. Research problems have centered around the building of efficient problem solvers (e.g., [Nilsson], and [Sacerdoti, 1974]), the constructing and maintaining of consistent and complete world models (e.g., [Fahlman], [Fikes et al.], and [Sacerdoti, 1977]), and the recognition of inconsistencies and errors in plans and recovering from them at execution time (e.g., [Fikes et al.], [Hayes], [Sacerdoti, 1977], and [Sussman]). Section 1.5 provides a review of AI robotics. Since this work studies skilled behavior, we have to provide solutions to the following research problems definition (2) calls for:

(1) Syntax and semantics for well-defined skills

(ii) Design of a skill formation process
Design of a system to activate skills for expressed tasks

This work addresses (i) and (iii). As will be shortly seen, we will replace the skill formation process by advice-taking from a human master.

In what way, then, are planned and skilled behavior different? Everyday human behavior is a mixture of the two modes of behaving: hardly any behavioral patterns are performed without at least an elementary preceding plan; hardly any behavior is a result of planning which has explicitly determined even the most elementary motor movements. Rather, in everyday human behavior, planning seems to use skills as operators: a plan determines a gross behavior while leaving control details in the care of skills at execution time. Good reasons exist for this division of labor: both modes contribute to optimality of behavior, albeit differently.

Planned behavior scores high in the task complexity dimension due to the power for predicting future events a symbolic representation provides. We will return to this aspect shortly. In skilled behavior, control is turned over to the immediate environment resulting in both high efficiency and strong reliability of control information. Control information is highly reliable since skills directly perceive states of the world without intermediary representations. Furthermore, skilled behavior is insensitive to the growth of numerical complexity in the
world. Since skills only take into account perceivable aspects of the immediate surroundings at execution time, it does not matter how complex the world is outside the perceptual range of skills or what the world will be like in future moments. Contrast this with the way planned behavior is sensitive to the complexity growth in the world. An increase in complexity in the world necessarily results in an increase in complexity in a world model which, in turn, results in efficiency problems in the planning phase. Another problematic result from the use of a world model is the need for checking whether the believed states of the world the model proclaims are still valid during the execution time of the plan. We will return to this consistency problem later in this introduction.

Because skills use perceptions for the control purpose and do not employ symbolic representations of the world, it follows that only present states of the immediate surroundings can effect skilled behavior. Projection of future states requires the use of a symbolic representation which skills do not possess. No such structural restrictions exist in planning. Particularly, planning can, in inferring any one step in a plan, project future events of the task and perform the step in a way that leads into most favorable consequences for the future steps. Hence, if a task is complex in such a manner that a solution deems projection of future events, planned behavior is superior to skilled behavior.
We will return to the demarcation between skilled and planned behavior in Conclusion. Since skilled behavior in some respect is superior to planned behavior, a study of a skilled system seems worth pursuing. A marriage of a skilled system with a planning mechanism represents an interesting future research problem.

1.2 PROPOSED SKILL SYSTEM

1.2.1 Formation of Skills: Advice-taking

A computational account to cover all different means of the human skill formation is undoubtedly an exceedingly complex task and is not attempted here. So much seems clear that imitation, verbal instructions, and learning from errors, all outside the range of planning, play important roles in human skill acquisition. Initially planned behavior, of course, can transform into a skill if repeated sufficiently often.

The branch of psychology called genetic psychology attempts to explain mental functions by their formation. Piaget argues, that prior to the existence of symbolic functions, formation of action-schemes in the child is performed by two mechanisms: assimilation and
accommodation. Assimilation modifies and incorporates reality data into action-schemes; accommodation modifies the schemes to fit reality. Piaget further contends that these action-schemes form the basis for later abstracted conceptual knowledge [Piaget and Inhelder].

In this work the simple solution we provide to skill formation is advice-taking. The crucial design feature of the system is a well-defined narrow message channel, a mini language, between a robot and its human master for requesting and transmitting advice. In case the robot lacks competence to perform a requested task, it recognizes the need for new advice, prompts its human master, and incorporates advice the master provides into the existing store of competence. Since new advice always hierarchically combines the robot’s primitive competence and/or earlier given advice, the robot’s competence with respect to task complexity gradually increases, and a need for new advice gradually decreases.

Advice-taking focused on skills consists of two separate functions which naively represent assimilation and accommodation: skill modification and skill acquisition. If the system does not possess competence for a given task, it first attempts to modify existing skills to cover the competence that is lacking. For any modification candidate the system asks the master to either confirm or reject the proposal. If skill modification attempts fail, the system acquires a new skill by asking the master to define the
skill using primitives and existing skills according to well-defined rules. The system further possesses the following characteristics which we regard as necessary conditions for an open-ended system: 1) the complexity of advice does not increase in the course of development, and 2) making structural changes to the system is never necessary to the master.

1.2.2 Syntax and Semantics of Skills

The main contribution of this work is a formal definition of skills and the design of a system to communicate with a human master and execute skills. In this section we describe skills; the structure of the proposed system is described in later chapters.

We present three types of skills which are hierarchically built from lower level defining skills or from the defined skill itself (recursion). The lowest level skills, which cannot be further decomposed, are called primitive skills. A primitive skill consists of a single movement coupled with a goal-state recognizer to discontinue the movement. The defining skills of a skill can be of any type including the primitive skill type. The skill types are:

**Type 1!**

<precondition> --- > <defining skill> <defining skill> ...
**Type 2:**

(EXCLUSIVE <precond> ----> <def skill> <def skill> ...  
<precond> ----> <def skill> <def skill> ...  
...)

**Type 3:**

(REPEAT-UNTIL <goal state> <def skill> <def skill> ...)

**Primitive skill:**

<movement> UNTIL <goal state>

<precondition> represents a boolean conjunctive normal form of perceptions or their negations.
<goal state> represents a boolean expression of perceptions.

Any perception represents a mapping, performed by an expert procedure, from the states of the robot's immediate surroundings onto the binary truth value. Recalling memory can also exist as a term in a precondition. Recalling represents a mapping from the content of the memory onto the binary truth value. We omit a discussion of recalling in this introductory chapter.

Omitting details, the semantic interpretations of the skill types are given below. Chapter IV describes both the syntax and semantics of the skill types in detail.

**Type 1:** If the precondition evaluates TRUE, perform the defining skills sequentially.

**Type 2:** Perform that one of the exclusive skills whose precondition evaluates TRUE.
**Type i**: Until the goal state evaluates TRUE, repeatedly perform the defining skills.

**Primitive skill**: Discontinue the (activated) movement when the goal state evaluates TRUE.

Clearly, the proposed skills do not capture the richness of human skills. Particularly, perceptions are rigidly used in the proposed skills only either allowing or preventing execution of defining skills. In human skills perceptions can flexibly influence the manner in which actions are performed. For example, depending on a perceived distance, a human can jump differently.

Earlier we argued that skills, if compared with plans, greatly reduce requirements for consistency checkings and corrective actions. Having defined skill types, we can return to this issue. To understand what is meant by consistency checking and corrective actions, consider planned behavior in which a planning mechanism manipulates a symbolic world model and produces a plan which does not incorporate perceptions for control. Control information in such a plan is based on believed states of the world. A monitor which executes the plan must then in some manner check that those states of the world model on which the plan is based are still consistent with reality at execution time. If the monitor recognizes discrepancies, it has to call for corrective actions. Obviously, then, the more detailed the states of the world model used in creating a
plan are, the more vulnerable such a system is in this regard. Since skills mostly utilize perceptions (actual states of the world) for the control purpose, sources for inconsistency are greatly reduced. The inconsistency problem is not, however, totally eliminated in the proposed system because the robot possesses a limited world model, called the Orienting Map, to expand the robot’s perceptual abilities for orienting purpose. In case the robot’s subworld consists of a house, the Orienting Map is composed of the adjacency relation of the rooms and their floor locations. Appendix E contains the Orienting Map of the simulation in Chapter II. Should a workman structurally change the house (say, by removing doors) without updating the robot’s Orienting Map, inconsistency problems in the above sense might result.

Skills of the proposed system further possess an important mechanism for dynamically creating subgoals. Refer to the Type 1 skill definition and its interpretation in 1.2.2. In a precondition each of the perceptions must evaluate TRUE before the defining skills are performed. Each of the perceptions in a precondition represents a binary-value difference between the satisfied precondition and the current state. If a perception evaluates TRUE, the respective difference does not exist, otherwise it exists. For each difference (perception) a unique skill, called a perceptual expectation, exists with the goal of eliminating the difference. A single skill may serve as an expectation
for several perceptions. In an attempt to execute a Type 1 skill or a chosen exclusive skill of a Type 2 skill, the perceptions of the precondition are evaluated one by one. If a perception evaluates FALSE, its perceptual expectation is called for and executed before the defining skills.

1.3 SKILLED BEHAVIOR: AN EXAMPLE

The purpose of this section is to describe the execution of skills and perceptual expectations in performing a specific task. In order for the reader to fully understand the example he has to be exposed to some technical details. We show a single skill which is general enough to guide the robot from a room in which it happens to be located into any target room. The chosen route will have the minimum distance in terms of the number of rooms crossed. First we need some preliminary concepts. As we mentioned earlier, the robot possesses a limited world model, called the Orienting Map. The Orienting Map is composed of the properties ADJ and FLOOR attached to the room names. Values of the properties are the names of adjacent rooms and floor name, respectively. Appendix E describes the Orienting Map.

Any reference to the Orienting Map (and other constituents of memory) is established by calling the function procedure RECALL with proper arguments. Below we
exhibit some example calls.

'Recall the floor of room R101':

(RECALL ?FLOOR FLOOR R101)
evaluates TRUE and the variable $FLOOR$ receives the value (FLOOR1).

'Recall the rooms adjacent to room R101':

(RECALL ?ROOM ADJ R101)
evaluates TRUE and the variable $ROOM$ receives the value (R102 R105 ELEVATOR).

'Recall the rooms adjacent to the rooms the variable $ROOM$ refers to':

(RECALL ?ROOM ADJ $ROOM$)
evaluates TRUE, if called after the previous recalling, and
the variable $ROOM$ receives the value (R101 R102 R103 R104 R105 R106) (that is, the adjacent rooms of the adjacent rooms of R101).

The robot's primitives, from which all skills are
hierarchically built, represent movements, perceptions, and
recallings. The above examples suffice for an explanation
of recalling. Since movements are not shown on the
hierarchical level of the example skill, we do not discuss
them here. Perceptions represent expert function procedures
which map the state of the robot's immediate surroundings
onto the binary truth value. Below are examples of calls of
three perceptual experts and their interpretations. Notice
that 'perception' here does not only imply 'directly aware
of' but also 'directly aware of and having the named
property'.

(BE-IN R101)
evaluates TRUE if the robot is located in room R101. We discuss in Chapter II how this expert might be implemented in reality. This expert perceives any room; that is, R101 can be replaced by any room name.

(BE-IN-ADJ-TO R101)
evaluates TRUE if the robot is in a room whose name occurs in the ADJ property of R101. Applies to all room names.

(BE-ON-FLOOR-OF R101)
evaluates TRUE if the robot is in a room whose FLOOR property is identical to the FLOOR property of R101. Applies to all room names.

The skill, representing a Type 2 skill, for going into any named room is given below in its stored, general form. GO-INTO-ABLE represents a general concept name, generated by the system itself, to denote all things that are 'go-into-able'. Obviously, at least rooms are such things.

[SKILL1] 'The skill for going into a named room' ::= 
\( \text{EXCLUSIVE} \ ((\text{BE-IN GO-INTO-ABLE})) \rightarrow (\text{DONE}) \)
\( ((\text{NOT (BE-IN GO-INTO-ABLE)}) \)
\( (\text{BE-ON-FLOOR-OF GO-INTO-ABLE}) \)
\( (\text{BE-IN-ADJ-TO GO-INTO-ABLE}) \)
\( \rightarrow (\text{GO TO GO-INTO-ABLE-DOOR}) \)
\( (\text{OPEN GO-INTO-ABLE-DOOR}) \)
\( (\text{ENTER IN GO-INTO-ABLE}) \)

[SKILL1] represents the semantic interpretation of the requests (! GO INTO <room>). Using the general reference (GO INTO GO-INTO-ABLE) or any of its instances, [SKILL1] can be later used as a defining skill in higher level skills. The skill utilizes four defining skills indicated by their
surface forms. DONE represents the empty skill whose execution causes control to be immediately returned. The definitions of the other defining skills are shown in Appendix A.

Recall that for each perception there exists a unique skill, a perceptual expectation, for solving the problem of evaluating the perception TRUE. Depending on how the robot is initially located, execution of the above skill may call the following two perceptual expectations:

'The perceptual expectation for
(BE-ON-FLOOR-OF GO-INTO-ABLE) ::= 
  ; Recall the floor of the room and go to the floor
  [EXP1] (((RECALL ?FLOOR FLOOR GO-INTO-ABLE))
        ---> (GO TO $FLOOR))

'The perceptual expectation for
(BE-IN-ADJ-TO GO-INTO-ABLE) ::= 
  ; Recall the rooms adjacent to the target room
  ; and go into one of those
  [EXP2] (((RECALL ?ROOM ADJ GO-INTO-ABLE))
        ---> (GO INTO $ROOM))

Both expectations represent simple Type 1 skills consisting of a single defining skill. Notice how the defining skills use indirect reference. Incidentally, the defining skill of [EXP2] represents an instance of the example skill [SKILL!]. Hence, execution of [SKILL!] may be recursively called.
we have now all ingredients for solving the problem of going into a named room. Let us assume that the robot is initially located in R105, and we request (! GO INTO R103). The request will trigger the fetching and binding of [SKILL1] to the task instance (details omitted). The skill to be performed is shown below. We have indicated with T. and F the evaluations of perceptions (see also Figure II.1). The number to the left indicates the position of an executable skill definition in the stack TASKLIST counted from the bottom to the top.

[1] (EXCLUSIVE (((BE-IN R103)) F

> (DONE))

(((NOT (BE-IN R103)) T

(BE-ON-FLOOR-OF R103) T

(BE-IN-ADJ-TO R103)) F

> (GO TO R103-DOOR)

(OPEN R103-DOOR)

(ENTER IN R103))

Since the precondition of neither of the exclusive skills evaluates TRUE, the perceptual expectation [EXP2] will be executed first to solve the sub-problem of going into a room adjacent to R103. [EXP2] is selected because the precondition of the second exclusive skill is closer to the satisfied state. The top of the task stack becomes:

[2] (((RECALL ?ROOM ADJ R103)) T

> (GO INTO $ROOM))
Recalling evaluates TRUE (adjacent rooms of R103 can be recalled) and binds the variable $ROOM to (R102). Since the whole precondition of [2] evaluates TRUE, its defining skill is executed. [2] will be replaced by the fetched and bound instance of [SKILL1].

\[2'] (EXCLUSIVE (((BE-IN $ROOM)) F
\(\longrightarrow\) (DONE))
(((NOT (BE-IN $ROOM)) T
(BE-ON-FLOOR-OF $ROOM) T
(BE-IN-ADJ-TO $ROOM)) T
\(\longrightarrow\) (GO TO $ROOM-DOOR)
(OPEN $ROOM-DOOR)
(ENTER IN $ROOM))

The whole precondition of the last exclusive skill of [2'] evaluates TRUE (R105, the room the robot is initially located, is adjacent to R102, the value of $ROOM). Had the robot been located in a room still further away from the target room out on the same floor, [EXP2] would have been called a sufficient number of times. A reference-variable, $ROOM in this case, represents a stack preserving previous recalls. Since the whole precondition of the second exclusive skill evaluates TRUE, skill definitions of its defining skills are in turn fetched, bound, and performed (using indirect reference). After performing [2'] the robot is located in R102 and [1] becomes under execution. Since the robot now is located in a room adjacent to R103, the target room, the whole precondition of the second exclusive skill of [1] now evaluates TRUE, and the robot moves into R103. Had the robot initially been located on a floor other
than FLOOR1, the expectation [EXP1] would have been invoked first. The defining skill of [EXP1] would cause the robot first to go into the elevator, to push the first floor button, and to exit from the elevator.

1.4 PSYCHOLOGICAL PERSPECTIVE

The branch of psychology called genetic psychology attempts to explain mental functions by their development in the child. Theories have been set forth to the effect that high-level symbolic cognitive functions have developed from concrete action-structures which themselves lack symbolic functions. For example:

When we say ... that thinking is an advanced form of skilled behavior, what we mean is that it has grown out of earlier established forms of flexible adaptation to the environment and that the characteristics that it possesses ... can best be studied as they are related to those of its earlier forms [Bartlett, p. 199].

Our use for 'skill' seems to be, at least partly, in agreement with Bartlett's. To us, you may recall, skilled behavior means directly recognizing and searching states of the immediate surroundings without a use of symbolic representations. If by 'thinking' Bartlett generally means use of symbolic representations, then in his use 'skilled behavior' obviously lacks such representations.
In genetic psychology the term 'action-schemes' is used to denote complex concrete structures in the child to analyze and understand the world prior to the ability to use symbolic functions, including a language.

... everyone agrees in recognizing the existence of an intelligence before language. Essentially practical—that is aimed at getting results rather than stating truths—this intelligence nevertheless succeeds in eventually solving numerous problems of actions ... by constructing a complex system of action-schemes and organizing reality in terms of spatio-temporal and causal structures. In the absence of language or symbolic function, however, these constructions are made with the sole support of perceptions and movements and thus by means of a sensori-motor coordination of actions, without the intervention of representation or thought [Piaget and Inhelder].

It is tempting to view our skilled system in such a Piagetian framework. The skills, then, represent action-schemes and the robot's overall competence represents (presymbolic) sensori-motor level of intelligence. If such correspondence is a viable one, a possible, ambitious future research problem would be the exploration of abstraction of conceptual structures from the robot's skill base.

1.5 ROBOTICS IN ARTIFICIAL INTELLIGENCE

The paradigmatic robot system in Artificial Intelligence has shown neither interest in developmental aspects of behavior nor skilled behavior but, instead, has
centered directly around adult planned behavior (see, however, [Sussman]). The major concerns have been how to organize conceptual knowledge about the world to form an adequate and complete world model, how to build a powerful and efficient problem solver, and how to make the two interact in a robot system. To quote McCarthy and Hayes:

... we regard the construction of intelligent machines as fact manipulators as being the best bet both for constructing artificial intelligence and understanding natural intelligence ... we shall say that an entity is intelligent if it has an adequate model of the world, ... if it is clever enough to answer a wide variety of questions on the basis of this model, if it can get additional information from the external world when required, and can perform such tasks in the external world as its goals demand and its physical abilities permit [McCarthy and Hayes].

In the paradigmatic robot system a problem solver, manipulating a conceptual world model, produces a plan (a symbolic representation of movements) whose execution will hopefully result in the robot’s achieving the expressed goal state of a given task. In a separate execution phase a monitor executes the plan, checks its consistency and correctness, and calls for corrective actions if needed.

A popular method for problem solving has been means-ends analysis [Newell and Simon] in which repeatedly the most important difference between goal states and the current state is searched and reduced. Operators which transform states are organized according to the differences they reduce. Each operator has a precondition which must be satisfied before the operator can be applied. After the
selection of the operator which will best reduce the most prominent difference, satisfying its precondition becomes a subgoal. The process is repeated until a proper operator sequence is found.

Search among operators easily leads to a combinatorial explosion in purely syntactic approaches. A gradual, hierarchical refinement of plans from top down along a semantic dimension represents one way of decreasing the planning time. [Sacerdoti, 1974] discusses the increase in problem solving power with the utilization of a hierarchy of abstraction spaces. He augments preconditions along the semantic dimension 'criticality'. A plan is formed on the highest criticality level first; then the plan is refined on the next level, and so on.

Another popular approach in applying a hierarchy to decrease complexity of a task is to break the goal into independent conjunctive subgoals. Since in reality many such subgoals interact, an extra methodology must then be provided to check interactions and order subgoals accordingly. The testbed for solutions to this subgoal interaction problem has been the stack-of-blocks problem: from an arbitrary initial setting of labeled blocks produce a stack where the blocks occur in the specific order (e.g., [Sacerdoti, 1975] and [Tate]).

[Sussman] provides an interesting developing alternative by relaxing the correctness requirement of produced plans and, instead, letting the system learn from
the errors it makes. Subgoals are assumed independent, and only if a produced plan fails, the independence assumption is rechecked, the plan is debugged, and the system learns. In the course of development the system creates a collection of critics from encountered errors.

When a plan has been produced, a monitor receives and executes it. In [Fikes et al.] the interface between the problem solver and the monitor comprises a so-called triangle table which represents reasons and effects of each step in the plan. In [Sacerdoti, 1977] the interface consists of a hierarchical procedural net.

The monitor must be equipped with the capability to correct possible inconsistencies between the believed states of the world the plan is based on and the real states of the world at execution time. Inconsistencies may arise, for instance, from an inadequate world model, from the dynamic nature of surroundings, or from the robot's inability to properly predict consequences of its own actions (the so-called frame problem [McCarthy and Hayes]). To indicate the range of required corrective actions after an inconsistency is recognized, [Hayes] proposes two separate data structures to be used for representing a plan: a tree of subgoals and a decision graph. If a subgoal fails during the execution phase, the decision graph indicates the effects of the failure and the extent to which new planning is required (see also [Fikes et al.] and [Sacerdoti, 1977]).
In the planning-based robotics, creation of a symbolic representation of the world represents a hard design problem, although so obvious that it is not usually mentioned. [Fahlman] describes a computer program for generating plans for building simple structures out of bricks and wedges. He reports: "... the extreme difficulty [compared to the planning system] of programming the modelling system, especially the stability test." What difficulties would lie ahead in forming a world model for large and complex task domains or for an open-ended task domain is so far a question without answers.

[Sacerdoti, 1977] and [Winograd] provide, with different design goals, elegant restricted planning-based robot systems. Winograd achieves impressive results in natural language competence in his simulated robot system. He restricts and idealizes the robot's environment: "... it is precisely by limiting the subject matter to such a small area that we can address the general issues of how language is used in a framework of physical objects, events, and a continuing discourse" [Winograd]. The emphasis of the system is on the analysis and synthesis of natural language sentences. The knowledge about the physical world is represented by assertions and procedures in PLANNER-language [Sussman and Winograd]. The world model, in addition to being extremely limited, is idealized in that its contents are always assumed to be consistent with reality. Sacerdoti has developed a system which from top down hierarchically
generates a correct plan. The planning process makes the conjunctive subgoal assumption and then, using what he calls constructive criticism, orders subgoals to take into account inherent interactions between them. A plan is represented as a procedural net allowing intelligent monitoring of the execution of the plan and recovering from errors in it.

We have only discussed Artificial Intelligence research which has a direct bearing on creating behavior. We have not dealt with a large body of research on vision, concept formation, learning, and natural language processing, among others.

I.0 OUTLINE OF CHAPTERS

The representation of this work has been basically divided into three somewhat overlapping chapters in addition to the Introduction and the Conclusion. Chapter II provides an informal representation of a simulated implementation of a skilled system. The hypothetical world is simulated on a graphics terminal. The underlying principles of the advice-taking system are shown mainly using examples. Chapter III discusses a utopian system, capable of developing on its own, on a conceptual level avoiding the use of computational terms. Chapter IV molds the Conceptual Model into the computational realm; the result is called The Computational Model. Some features of the Conceptual
Model for which no computational counterpart yet is found are simply rejected in the Computational Model, and some such developmental features are replaced by advice-taking. The simulation of Chapter II is based on the Computational Model. The Conclusion will, in the light of concrete examples from the simulation, return to a discussion on the demarcation between skilled and planned behavior.

A hasty reader might just read Chapter II and then the Conclusion. A more patient reader might go through Chapters II and IV and the Conclusion. If the reading of a computational constituent or an interrelation calls for conceptual flesh around it, the reader can browse the respective sections of the Conceptual Model. The structures of Chapters III and IV are identical. A persistent reader might just follow the ordering of the chapters in his tedious task.
II. OVERVIEW OF THE SYSTEM

This chapter provides an informal overview of the proposed principles for building an advice-taking skilled robot system and describes a simulator. The following chapters deal with the principles in greater depth: Chapter III on the conceptual level and Chapter IV on the computational level in a formal manner.

The first section of this chapter describes one possible implementation of a developing system and, in due course, the constituents of the system and their interrelations are displayed. A justification and an exact description of these constituents are left for the following chapters. The second section demonstrates the behavioral power of the implemented system by exposing the reader to tasks the robot can perform in a simulated world. The section describes how the robot acquires and uses its skills, and how the robot’s taxonomy of individuals in the environment develops in parallel with skill acquisition.
II.1 IMPLEMENTATION AND SIMULATION

II.1.1 The World, Sampo, and Primitives

As a reminder, this work describes the principles of an advice-taking system that, starting with a complete set of primitives, empowers the robot with increasing practical competence to cope with concrete tasks. Since we establish our goal as the achievement of an open-ended system, we do not attempt to formalize any particular level of the robot's competence. Rather, we aim at formalizing the mechanism of progression by defining skills in such a manner that the acquisition of new skills does not increase in complexity as development progresses.

The primitives of the proposed robot represent motor organs which cause movements, and perceptions which control activations and deactivations of movements. Furthermore, to furnish the robot with a means for improving behavioral competence through experience, a procedure RECALL has been devised to retrieve recorded perceptual experience for control purposes.

Any implementation of the proposed developmental system calls for the following three steps:

(1) Define the sub-world and the robot's tasks.
(2) Define an optimal set of motor organs and perceptions for the robot. The set of primitives should be complete so that necessary behavioral patterns for conceivable tasks could be created using those primitives according to well-formed rules.

(3) Build the motor organs in the robot's body, implement effective procedures to correspond to the chosen perceptions, and implement the proposed principles.

The system would then take charge of development by interacting with its human master in an appropriate manner. Notice that neither the motor organs nor the perceptions need to correspond to those that the human possesses.

To demonstrate the proposed principles and their power, we simulate a hypothetical implementation. The robot of this simulation is called Sampo. The hypothetical world consists of a 3-floor house with structurally identical floors with an elevator connecting the floors (see Figure II.1). Doors open and close by sliding along rails if a push-down switch, located by the doors, is pushed. The rooms contain a number of movable and fixed objects. At the moment of simulation, three kinds of movable objects — tables, chairs, and baskets — exist with each object displayed as a circle with the initial letter of the object name inside the circle. Sampo is also displayed as a circle with an angle inside the circle indicating the direction of
Sampo's vision (see Figure II.1).

On the back and front Sampo has attached grasping mechanisms and on the front a hand for pushing buttons. The grasping mechanism on the back is used for carrying objects, and the mechanism on the front, for clearing possible obstacles out of the way. While keeping its vision in a forward direction, Sampo can move forward, backward, right, and left. It can further turn its body and vision both to the left and to the right. Sampo's vision is fixed in the body in such a way that when the body turns then its vision also turns.

Sampo's elementary movements are displayed in Table II.1. Implementation of these movements with respective motor organs represents an engineering task with which we shall not be concerned here.

| MOVE-FORWARD   |
| MOVE-BACKWARD |
| MOVE-RIGHT    |
| MOVE-LEFT     |
| TURN-RIGHT    |
| TURN-LEFT     |
| WAIT          |

Table II.1 Sampo's Movements
Figure II.1: Simulated House
We turn now to Sampo's perceptions, the remaining set of primitives. Specialized perceptual expert procedures perform the act of perceiving. Each of the experts is a function that responds TRUE or FALSE to a call depending upon whether the situation holds in the environment where the expert possesses proficiency. Since an expert responds only if it is called, it follows that only anticipated perceptions can occur. Consequently, Sampo's behavior is efficient with regard to perceptual processing since perceptions other than those that have relevance in the ongoing task are ignored. But it also follows that exceptional events in the surroundings are also ignored, for instance, if the house where Sampo is located catches fire, Sampo does not recognize it.

Sampo possesses a perceptual expert for recognizing the current room. The call and the factual 'belief' implied by the respective percept are:

\[(\text{BE-IN } \text{<room>}) ; \quad \text{'(I am) located (now) in <room>}'\]

We envision this expert implemented in reality by, for example, using discriminative colors in standard form patches. According to this proposal, each room has a certain color patch of a standard form on the walls. Perceiving an anticipated room then transforms into calling the procedure dedicated to recognizing a patch of the standard form in the visual field, transforming the color into the respective room name, and comparing the name with
the argument in the call. A similar perception but without a specific anticipation also exists. This expert evaluates TRUE if Sampo is located in any room and stores the trace of the specific percept of a room:

(\text{BE-IN~\text{ROOM}}) ~; ~'(\text{I am) located (now) in \langle\text{room}\rangle}~\\

Sampo also has an expert for visually perceiving individual doors. We use the convention for naming doors whereby a name is of the form: \langle\text{room}\rangle-\text{DOOR}'. Sampo can further visually perceive both an open and a closed elevator door. The calls of the experts and the respective factual 'beliefs' of the percepts are:

(\text{SEE~\langle\text{room}\rangle-\text{DOOR}}) ~; ~'(\text{I) see (now) the door of \langle\text{room}\rangle}~\\

(\text{SEE\text{ELEVATOR-DOOR-OPEN}}) ~; ~'(\text{I) see (now) an open elevator door}~\\

(\text{SEE\text{ELEVATOR-DOOR-CLOSED}}) ~; ~'(\text{I) see (now) a closed elevator door}~

Using a similar schema as above, we also imagine the doors in the house marked using color patches, but the shape would differ from the patches on the walls. The color of a door to a room would be identical to the color of the room. Perceiving a specific door would similarly transform into first looking for the patch and then comparing the respective name with the name given in the call.
One part of Sampo's memory, composed of one or more **orienting maps**, models the spatial organization of the environment. A single orienting map, call it the Orienting Map, will suffice for the chosen subworld. The map represents the adjacency relation of the rooms and their floor locations. For example, the room R103 introduces these two entries in the map (see Figure II.1): (ADJ R102) and (FLOOR FLOOR1). The interpretation of the entries is: 'R103 is adjacent to R102 and it is on the 1st floor'. Appendix E contains the complete Orienting Map of this implementation. The following three perceptual experts first call (BE-IN <room>) and then check appropriate properties in the Orienting Map. Hence, elementary inferences are allowed in our notion of 'perception'.

(BE-IN-ADJ-TO <room>);
'(I am now) in a room adjacent to <room>'

(BE-ON-FLOOR-OF <room>);
'(I am now) on the floor of <room>'

(BE-ON <floor>); '(I am now) on <floor>'

Sampo possesses perceptual experts for visually recognizing each of the objects it might be asked to manipulate. Furthermore, a perceptual expert exists for visually recognizing if any obstructing obstacle is located in front of Sampo. In this simulation, then, Sampo also possesses the following perceptual experts:
Furthermore, any object that can be visually perceived independent of context can also be perceived in context with respect to the relation 'in front of'. That is, Sampo also possesses the following perceptual experts:

(SEE <object1> IN-FRONT-OF <object2>);
'(I) see (now) <object1> in front of <object2>'

(SEE OBSTACLE IN-FRONT-OF <object>);
'(I) see (now) an obstacle in front of <object>'

<object1> is defined to be in the relation 'in front of' with <object2> if Sampo's body would touch <object1> before <object2> in case Sampo moves straight toward <object2>.

Sampo needs some faculties for tactile perception. Such an implemented expert is a procedure which recognizes whether the body is touching its right, left, front, or back side. The procedure also recognizes if something obstructs Sampo's turning to the right or to the left.

(BODY-SENSE <direction>);
'(My) body touches (now) in the <direction>'

<direction> can be: FRONT, BACK, LEFT, RIGHT, LEFT-TURN, or
RIGHT-TURN. An appropriate arrangement of sense switches in Sampo's body would in reality represent an effective realization of this expert. Sampo also possesses an expert for sensing if something is being carried:

(SENSE-CARRYING); '(I am now) carrying (something)'

This expert could, again, easily be realized using sense switches installed in the carrying mechanism. Furthermore, Sampo has a perceptual expert for being by a named object:

(BE-BY <object>); '(I am now) by <object>'

Finally, Sampo possesses a perceptual expert for sensing when continuous turning around evolves into a half or full circle. This expert is needed, for example, to avoid a possible indefinite revolving in cases when something is looked for by turning around, but the anticipated object does not appear in the visual field:

(SENSE-OPPOSITE-TURNING-POINT); '(I) have (continuously) turned 180 degrees'

(SENSE-START-TURNING-POINT); '(I) have (continuously) turned 360 degrees'

This expert could be implemented in reality again using sense switches around the body. Sampo's perceptual experts are collected in Table II.2.
In this chapter we discuss how Sampo's perceptual experience can be used to control its behavior. Sampo possesses collections of declarative, symbolic descriptions that we call the Memory since they record Sampo's perceptual experience (with the exception of LTOC). The Memory is composed of the following functionally distinct parts:

1. The **Perceptual Memory** (a list of symbolic representations of percepts)

2. The **Propositional Memory** (a set of abstracted non-spatial propositions of percepts)

3. **Orienting maps** (sets of abstracted spatial propositions of percepts)
The recording of perceptions in the Memory happens as follows. A call of a perceptual expert returning TRUE indicating a satisfied perception will be appended in the Perceptual Memory as a trace of the percept. From a list of such traces, abstractions of selected properties for both the Propositional Memory and orienting maps occur as concurrent processes. Abstractions not involving considerations of locomotion are stored in the Propositional Memory while those involving such considerations are stored in the orienting maps. These abstraction procedures are initially given procedures. Elsewhere we speculate, without concreteness, about abstracting conceptual structures from skills, from the robot's action-schematism, as a far-reaching developmental step. These two abstraction conceptions should be kept separate: the one, discussed here, works on perceptual experience; the other, on action experience. As an example of perception-based abstraction, if Sampo first perceives being in room R101 and then sees a table, the traces (see Table II.2):

\[(\text{BE-IN R101})\]

(SEE TABLE)

will be consecutively stored in the Perceptual Memory. From
those traces the abstraction procedure for extracting the property LOCATION gathers the proposition:

\[(\text{TABLE LOCATION R101})\]

and stores it in the Propositional Memory. In this writing, LOCATION is the only implemented abstraction procedure. Notice that the Perceptual Memory, being inconsistent with reality (for example, someone moves the table in the above example from R101), does not result in severe inconsistency problems in this skill approach, since skills can be so easily defined that they anticipate such inconsistencies.

Orienting maps contain geographical information about the surroundings. As mentioned earlier, a single such map, called the Orienting Map, is used in this simulation to describe the spatial organization of the house, that is, the adjacency relation of the rooms and their floor locations. We have not implemented procedures for abstracting propositions for the Orienting Map, instead, we assume that the master initially describes the Orienting Map. In the Conceptual Model in Chapter III, we will discuss problems in abstracting orienting maps. See Appendix E for the Orienting Map of this simulation.

The Logical Taxonomy of Concepts (LTOC) represents a developing discrimination hierarchy for individuals in Sampo's known environment. LTOC is organized as a member-set-superset tree as indicated by the properties SONS and FATHER attached to concept names. SONS of a concept
name indicate its members; FATHER indicates the superset of the concept. The initially given concept name SOMETHING forms the 'root' of LTOC and names of individuals in the surroundings form 'leaves' of the tree. The root and leaves are connected with classificatory concept names created in the developmental process. See Figure II.6 for an example of an instance of LTOC. LTOC provides scopes for skills: a skill applies to all individuals that are descendants of the concept names in the skill. For example, if a skill contains the concept name ENTER-IN-ABLE then the skill applies to room R101, but the skill does not apply to chairs (see Figure II.6). LTOC defines a logical taxonomy in the sense that it classifies concepts using the manners in which they participate in actions as the basis for similarities and dissimilarities between them. For example, LTOC of Figure II.8 indicates that a chair and a door are different in that the former can be picked up, left by other objects, and so on, but the latter cannot. We informally discuss in the simulation how LTOC develops in parallel with skills. Notice that LTOC does not develop from perceptual experience as do the other constituents of the Memory.

We have discussed how contents of the Memory are produced, and now we will explain the ways the Memory is utilized. All references to the Memory are established by calling the procedure RECALL with proper arguments. If recalling is successful, RECALL will return the value TRUE and bind a reference-variable to the extension (a set of
names of individuals) in the LTOC which satisfy the **intension** (the description of the individuals in the call). If the extension is empty, **RECALL** returns the value **FALSE**. A reference-variable is indicated by a prefix of '?>' in binding (when the extension is to be established) or '>$' in referring (when the extension is referred to). Only those names of individuals which are descendants of siblings of a reference-variable are considered in establishing the extension. For example, if $ROOM and the names of the rooms are sons of ENTER-IN-ABLE in LTOC, ?ROOM will restrict a search to the names of rooms when an extension is established. Basically, a call is of the form:

```plaintext
(RECALL ?<ref-var> [<intension> [EXCEPT <intension2>]])
```

Its interpretation is: Return those names of individuals in LTOC that are siblings or descendants of siblings of <$reference-variable> and (optional) which satisfy <intension>, but reject (optional) the ones that satisfy <intension2>. The use of a reference-variable roughly corresponds to the use of a pronoun in natural language communication.

Intensions, described in detail in the Computational Model, can refer to traces of percepts in the Perceptual Memory, propositions in the Propositional Memory, or propositions in the Orienting Map. An intension can also refer to another reference-variable thus establishing a double indirect reference. A few examples in Table II.3
illustrate the use of RECALL.

(RECALL ?ROOM)
- recall all rooms

(RECALL ?ROOM $ROOM)
- recall the rooms $ROOM refers to

(RECALL ?ROOM EXCEPT BE-IN FIRST)
- recall all rooms besides the one you were in first

(RECALL ?ROOM BE-IN FIRST)
- recall the room you were in first

(RECALL ?ROOM ADJ R101)
- recall the rooms adjacent to room R101

(RECALL ?ROOM ADJ $ROOM)
- recall the rooms adjacent to rooms $ROOM refers to

Table II.3 Examples of Recalling

Computationally, a reference-variable represents a push-down stack with recalling pushing the stack and a reference pointing to the top of the stack. An explicit pseudo-skill RELEASE is designated for popping values from a stack. An example below illustrates the use of reference-variables. A skill is defined for finding a chair in case the robot has stored the location of a chair in its Propositional Memory.

(((RECALL ?ROOM LOCATION-OF CHAIR)))
(GO INTO $ROOM)
(PICK UP CHAIR)
(RELEASE $ROOM))

Figure II.2 Example of a Use of a Reference-variable
II.1.3 Natural Language and Parsing

Communication between the master and Sampo is conducted in restricted English. Admissible natural language sentences fall into the three distinct categories shown in Table II.4.

<table>
<thead>
<tr>
<th>Type</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>(! &lt;imperative sentence&gt;)</td>
</tr>
<tr>
<td>Claim</td>
<td>(: &lt;declarative sentence&gt;)</td>
</tr>
<tr>
<td>Question</td>
<td>(? &lt;interrogative sentence&gt;)</td>
</tr>
</tbody>
</table>

Table II.4: Sentence Types

The three sentence types call for different kinds of responses and, hence, they are dealt with differently in this system. Questions will not be discussed at all in this introductory chapter; later chapters will conduct a tentative, incomplete discussion on the question-answering capabilities of a robot of the proposed kind.

The master uses claims, bypassing Sampo's own perceptual experience, to incorporate into Sampo's Memory facts of the world. For example, in this simulation the Orienting Map of the house is described in this manner using claims such as:

( : R101 IS ADJACENT TO R102 )
Since Sampo's behavior is of primary interest to us, the major concern in natural language processing focuses on requests. A received request is first parsed into a deep case structure which is then used to retrieve an appropriate skill to represent the semantic interpretation of the request. We have restricted requests so that they can only contain verbs, nouns, and prepositions (or other qualifiers). The selection of deep cases is solely based on pragmatic considerations: the cases are: OBJECT, AGENT, INSTR, LOCAT, DESTIN, TRAJECT, and TIME. Figure II.3 shows an example of parsing.

We have not yet implemented a real parser; instead, the master provides parsing results for requests in much the same fashion as he provides skill definitions. Sampo stores deep case structures and applies them when appropriate. Sampo can perform some elementary generalizations on parsing results (not discussed here). This 'rote learning parsing' is not meant to represent any serious substitute for a real parser. Appendix D contains parsing results of the system.

(! GO INTO R101)
   I
   V
PARSER
   I
   V
   ((VERB GO) (DESTIN (PREP INTO) (NOUN R101)).)

Figure II.3: Example of Parsing
Skills are stored as ordered pairs:

\[
(\text{<deep structure schema>} \text{<skill definition>})
\]

A deep structure schema represents a generalized deep case structure. Verb and preposition indicators in a schema may carry lists of verbs or prepositions (or qualifiers), and noun indicator may carry any created discriminatory concept of LTOC. In contrast, in a deep case structure of a request, VERB indicates a single verb, PREPs indicate single prepositions, and NOUNs indicate names of individuals ('leaves' of LTOC). A schema applies to a deep case structure of a request if the verb and prepositions of the latter are identical to or members of the respective elements in the schema, and the nouns of the latter are identical to or descendants of the respective nouns in the schema. If a schema applies to a request, the bound skill definition (general concept names in the skill definition replaced by names of individuals in the request) represents the semantic interpretation of the request. Following is an example of a deep structure schema which applies to the deep case structure of the request in Figure II.3 assuming that R101 exists as a son of GO-INTO-ABLE in LTOC:

\[
((\text{VERB (GO MOVE)}) \text{ (DESTIN (PREP (IN INTO))))})
\]

\[
(\text{(NOUN GO-INTO-ABLE)})
\]
A response to a request, then, calls for parsing the request into its deep case structure, retrieving, employing pattern matching, the respective skill definition, binding the generalized concepts in the skill definition to their instances displayed in the request, and performing the bound skill.

II.1.4 Development

While a utopian robot would be one whose perceptual faculties and orienting maps also develop, we have only implemented development with regard to skills and LTOC. Even this restricted development is implemented in the form of advice-taking. All other constituents are initially given. The two procedures which participate in developing skills and LTOC are called SKILL-MODIFICATION and SKILL-ACQUISITION. In the framework of Piagetian psychology, these procedures naively represent assimilation and accommodation, respectively. This section briefly discusses these two procedures in an informal fashion, section II.2 conducts a simulation on development of both skills and LTOC. Chapters III and IV contain a more thorough discussion of the subject matter.

SKILL-MODIFICATION procedure is invoked if Sampo receives a request for which no skill can be found. The procedure applies three types of heuristics in an attempt to
generalize any of the existing skills to cover the pending request. For each candidate SKILL-MODIFICATION only suggests a generalization and asks advice; it is the master's duty to confirm or reject the proposal. The heuristics are (assume that LTOC is the one in Figure II.7):

(1) A skill is a candidate for modification if its deep structure schema otherwise applies to the request but the respective verbs are different. If the master confirms the modification suggestion, the verb of the request is added to the list of verbs in the schema. For example, assuming that the master regards requests (!GO INTO <room>) and (!MOVE INTO <room>) to be semantically equivalent, the deep structure schema:

\[ ((\text{VERB GO}) \ (\text{DESTIN} \ (\text{PREP INTO}) \ (\text{NOUN GO-INTO-ABLE}))) \]

is a successful candidate for modification with respect to the following deep case structure:

\[ ((\text{VERB MOVE}) \ (\text{DESTIN} \ (\text{PREP INTO}) \ (\text{NOUN R101}))) \]

and the schema is modified into:

\[ ((\text{VERB (GO MOVE)}) \ (\text{DESTIN} \ (\text{PREP INTO}) \ (\text{NOUN GO-INTO-ABLE}))) \]

(2) A skill is a candidate for modification if its deep structure schema otherwise applies but prepositions are different in a single instance. Again, if the master confirms the proposal, the preposition of the request is added to the respective preposition list in the schema. For example, assuming that the master regards requests (!GO INTO <room>) and (!GO IN <room>) to be semantically equivalent, the deep structure schema:

\[ ((\text{VERB (GO MOVE)}) \ (\text{DESTIN} \ (\text{PREP INTO}) \ (\text{NOUN GO-INTO-ABLE}))) \]

is a successful candidate for modification with respect to the following deep case structure:

\[ ((\text{VERB GO}) \ (\text{DESTIN} \ (\text{PREP IN}) \ (\text{NOUN R101}))) \]

and the schema is modified into:

\[ ((\text{VERB (GO MOVE)}) \ (\text{DESTIN} \ (\text{PREP (INTO IN)}) \ (\text{NOUN GO-INTO-ABLE}))) \]
(3) A skill is a candidate for modification if an increase of generality of one concept makes the skill applicable. Increasing generality of a concept corresponds to replacing it with one of its ancestors in LTOC. This modification instance may generate new concepts in LTOC if the existing ones do not capture the proper level of generality with respect to the request (see below).

To amplify the third heuristic, without going into details as several rules are involved, let us run through an example. Assume that a skill has been acquired for the request (! GO INTO R101) and the room name R101 has been incorporated in LTOC as a new name of an individual. Assume further that no other room names exist in LTOC. If the request:

(! GO INTO R105)

is received, SKILL-MODIFICATION suggests the existing skill be used. After receiving acceptance from the master, SKILL-MODIFICATION creates a new concept GO-INTO-ABLE and generalizes the skill to the form:

(! GO INTO GO-INTO-ABLE)

The procedure further attaches GO-INTO-ABLE into LTOC with two sons: R101 and R105. This example depicts only one simple instance of generalization, variations of the theme will be given in the simulation in the next section and particularly in Chapter IV. In short, SKILL-MODIFICATION modifies skills and progressively associates concepts in LTOC. Association occasionally creates new concept names.
**SKILL-ACQUISITION** is the procedure which receives control if a skill does not exist for a request and **SKILL-MODIFICATION** fails in its attempts to modify existing skills. **SKILL-ACQUISITION** asks the master to define the new skill using the existing ones according to well-formed rules. In the above example, **SKILL-ACQUISITION** created the first skill for going to room R101 and attached R101 in LTOC. In short, **SKILL-ACQUISITION** acquires skills and progressively differentiates concepts in LTOC.

Initially, Sampo only possesses a single concept SOMETHING in its LTOC. From that seed **SKILL-MODIFICATION** and **SKILL-ACQUISITION** progressively mold an increasingly elaborate hierarchy of concepts, the former by gathering lower level concepts under higher level ones, and the latter by splitting higher level ones into lower level ones.

How Sampo might perform even some elementary steps of skill acquisition on its own poses a difficult question for future research. As it stands now, Sampo is capable of being taught, but lacks abilities to learn new skills. It can, however, modify existing skills to some extent.
II.1.5 System Structure and Simulation Environment

We simulate Sampo's behavior on a cathode ray memory screen. At any moment during the simulation, the terminal displays Sampo's current floor location. Excluding graphics interface, the system has been programmed in MTSLISP-language. The interface with the terminal has been programmed as a collection of FORTRAN function routines. Each of the primitives (see Table II.1 and Table II.2.) represents a FORTRAN function routine: a movement changes Sampo's position on the screen, and a perceptual expert simulates the respective perception.

We want to omit detailed manipulation of the hand and the grasping mechanisms because such considerations would lead to a lengthy discussion of peripheral issues. Instead, we take the elementary skills listed in Table II.5 for granted and implement them as FORTRAN subroutines. This side-stepping does not, however, represent any relaxation of our theoretical position. Those skills, too, we argue, could be defined using our well-formed rules, provided that proper perceptual experts and motor organs exist.
(GRASP-BACK <pick-up-able>)
Turn around and grasp in the back <pick-up-able> you were facing.

(GRASP-FRONT <pick-up-able>)
Grasp in front <pick-up-able> you were facing.

(UNGRASP-BACK <pick-up-able>)
Lower <pick-up-able> on the back and ungrasp.

(UNGRASP-FRONT <pick-up-able>)
Lower <pick-up-able> on the front and ungrasp.

(OPENS <door>)
Push the button of <door>.

(CALL-ELEVATOR)
Push the elevator button.

(PUSH-BUTTON <floor#>)
Push the button of <floor#>.

(FIX-START-TURNING-POINT)
Remember the orientation of your visual field.

Table II.5 Elementary Skills Taken for Granted

The overall system organization is shown in Figure II.4. The main procedure (SAMPO) receives sentences from the master. FACT-MANIPULATION procedure would, if it fully existed, take care of updating the Memory according to the master’s claims and answering questions. As we mentioned earlier, we have not implemented question-answering. If a sentence represents a request, the request is parsed and a skill for the request is retrieved, bound, and stored in the list TASKLIST. If such a skill cannot be fetched, SKILL-MODIFICATION attempts to modify any of the existing skills to cover the pending request. If SKILL-MODIFICATION succeeds, the modified skill replaces the old one in the
collection of skills, and the bound instance of the skill definition is placed in TASKLIST. If SKILL-MODIFICATION fails, SKILL-ACQUISITION procedure asks the master to define a new skill. The received skill is stored in the collection of skills, and the bound instance of it is placed in TASKLIST. When the master types NIL signaling the end of the task definition, TASKLIST is transferred to MONITOR for execution.

Before outlining the gross structure of MONITOR, we need briefly to describe the three forms of skills: ORDINARY, EXCLUSIVE, and REPEAT-UNTIL. An ORDINARY skill defines a linear sequence of lower level skills. An EXCLUSIVE skill, offering multiple tactics for a task, represents a list of ORDINARY skills, one of which will be selected for execution. ORDINARY and EXCLUSIVE skills are not strikingly different because an ORDINARY skill actually represents a trivial EXCLUSIVE skill. Finally, REPEAT-UNTIL skills provide a means for creating iterative behavior.

An ORDINARY skill contains as a precondition for execution a conjunctive normal form boolean expression of calls to perceptual experts or RECALL (both, you may recall, evaluate TRUE or FALSE upon a call). 'T' as a precondition signals the always TRUE precondition. If a precondition is satisfied, the list of defining skills is executed sequentially. If the precondition evaluates FALSE, the perceptual expectation of the first perception that forced the expression to fail will be executed first. For every
perceptual expert and its negation, there exists a unique perceptual expectation, a skill, whose execution will hopefully result in successful perceiving. The mapping between perceptions and perceptual expectations is many-to-one, that is, a single perceptual expectation may represent an expectation of many perceptions. As an example of an ORDINARY skills, the skill definition for the request (! GO INTO ELEVATOR) is as follows:

```plaintext
(((BE-IN-ADJ-TO ELEVATOR))
(GO TO ELEVATOR-DOOR)
(CALL ELEVATOR)
(TURN-RIGHT UNTIL (SEE ELEVATOR-DOOR))
(WAIT UNTIL (SEE ELEVATOR-DOOR-OPEN))
(ENTER IN ELEVATOR))
```

In this case the precondition consists of a single perception which evaluates TRUE if Sampo is located in a room adjacent to the elevator. If Sampo is not so located, the precondition evaluates FALSE, and the perceptual expectation skill for that perception will be performed first. (A skill for going to a room adjacent to the elevator represents an obvious expectation for this perception.) This skill implicitly assumes that (! GO INTO ELEVATOR) is never called when Sampo is already located in the elevator.

An EXCLUSIVE skill contains two or more ORDINARY skills for a task. The topmost of the ORDINARY skills whose precondition evaluates TRUE is chosen for execution or, if none evaluates TRUE, the one is chosen that carries the greatest number of consecutive perceptions TRUE counted from
the beginning of the expression. For example, the following EXCLUSIVE skill represents a skill definition with two tactics for the request (! GO INTO R202):

(EXCLUSIVE (((BE-IN R202))
 (DONE)))
 ((NOT (BE-IN R202))
 (BE-ON-FLOOR-OF R202)
 (BE-IN-ADJ-TO R202))
 (GO TO R202-DOOR)
 (OPEN R202-DOOR)
 (ENTER IN R202)))

The first tactic, the trivial one, is selected if Sampo already is located in R202. This tactic simply proclaims that nothing need be done. The other tactic, containing three perceptions as a precondition, is chosen in other cases. If Sampo is not located on the floor of R202 (the second floor), the respective perceptual expectation is executed first (obviously a skill for going to the second floor). If Sampo is not located in a room adjacent to R202, the respective expectation is called for execution (obviously a skill for going to a room adjacent to R202). Both the skill and respective perceptual expectations are stored in a general form applying to any room name. Notice how the triggering of perceptual expectations dynamically creates subgoals for moving Sampo to a room adjacent to the goal room.

A REPEAT-UNTIL skill contains a condition-state to discontinue the execution of an iteration. A sequence of defining skills are circularly executed until the condition evaluates TRUE. The condition is tested prior to activation
of each of the defining skills. For example, the following skill represents a skill definition for the request (! SEARCH FOR CHAIR):

(REPEAT-UNTIL (OR (SEE CHAIR)
    (RECALL $ROOM))
    (REMEMBER OTHER ROOMS)
    (GO INTO $ROOM)
    (LOOK FOR CHAIR))

The skill is stored in a general form applying to any object. Figure II.5 depicts the structure of MONITOR. In the course of simulation in section II.2 we will show several additional examples of the three skill types.
Figure II.4: System Organization
Figure II.5: MONITOR Organization
II.2 BEHAVIORAL SIMULATION

In this section we demonstrate Sampo's growing competence in dealing with practical tasks. We simulate both acquisition and generalization of skills and shaping of LTOC.

II.2.1 Initial Setting

The simulation begins at the point where Sampo already possesses the skills for the requests below. Some of the skills have already been subject to generalization (S1, S2, and S6). In actuality, this general status of a skill is indicated in the deep structure schema of the skill; here we indicate, for the sake of clarity, the scope of a skill in its surface form. These skills, in turn, have been defined using other, still lower level, skills. Appendix A provides detailed definitions of all skills in this simulation in their final form. Figure II.6 shows the status of LTOC at the moment the simulation begins.

(S1) (! CIRCUMVENT OBSTACLE TO GO-TO-ABLE)

Since there is an obstacle on the way to a GO-TO-ABLE (this skill is invoked only in such a situation), try first to go around to the left, then to the right, and finally, if neither succeeds, remove enough obstacles to free the way.
(S2) (! ENTER IN ENTER-IN-ABLE)

When facing the door of an ENTER-IN-ABLE and seeing the door open, move forward until arrive there. Move forward any obstacles blocking the doorway.

(S3) (! GO INTO ELEVATOR)

When being in a room adjacent to the elevator, go to the elevator door. Call the elevator. When see the elevator door open, enter in elevator.

(S4) (! GO TO FLOOR!)

Go into the elevator. Push the button of the first floor. Exit from the elevator when the door has opened.

(S5) (! LOOK FOR TABLE)

Turn around until either see a table or have turned the full circle.

(S6) (! GO TO GO-TO-ABLE)

If see the named GO-TO-ABLE and no obstacles, move straight to the target object. If see the GO-TO-ABLE and see obstacles, move first straight to the obstacles, circumvent the obstacles, and then move straight to the target object.

When the simulation begins, Sampo is located in room R105 with its vision facing to the right.

II.2.2 Task #1

As the first task, let us assume that the master wants a table in room R202 to be moved by a wall in R204. The master issues the first request:
No skill is found for the request. SKILL-MODIFICATION proposes (S3) be used as the skill for the request, but the master rejects the proposal. SKILL-ACQUISITION then asks advice from the master. SKILL-ACQUISITION displays the existing skills, the primitives, and the skill building rules, if needed. The master defines the skill as shown below. It is important that skills are structurally general even if their applicability range is specific at the moment of definition.

; the skill for going into R202
; (S7) (! GO INTO R202) ::= 
; (EXCLUSIVE
; ; if already in R202
; ; then o.k.
; ; (((BE-IN R202))
; (DONE))
; ; if not in R202 but on the floor of R202
; ; and in an adjacent room
; ; then go to the separating door, open it, and enter
; ; (((NOT (BE-IN R202))
; (BE-ON-FLOOR-OF R202)
; (BE-IN-ADJ-TO R202))
; (GO TO R202-DOOR)
; (OPEN R202-DOOR)
; (ENTER IN R202)))

The remaining task for SKILL-ACQUISITION is to locate the new name R202 in its proper place in LTOC. The procedure always attempts to locate a new name as a son of
the least general generated concept name. If no proper father is found, the new name will become a son of SOMETHING. In this case, LTOC contains only two created concept names (GO-TO-ABLE and ENTER-IN-ABLE) which both are equally general. SKILL-ACQUISITION asks the master’s advice concerning applicability of such skills which contain the general concept names. Since the master confirms that (S2) applies to R202 but (S6) does not, R202 becomes a son of ENTER-IN-ABLE (see Figure II.7).

Sampo studies the defining constituents of the received skill. DONE is a primitive empty skill causing immediate return of control, OPEN is an elementary skill (Table II.5), and ENTER IN is now defined by (S2) (after locating R202 as a son of ENTER-IN-ABLE). The remaining defining skill, (GO TO R202-DOOR), is not well defined with a new name R202-DOOR. SKILL-MODIFICATION suggests (S4) or (S6) be used to cover (GO TO R202-DOOR), too. Since the master confirms the latter alternative, R202-DOOR becomes a son of GO-TO-ABLE (see Figure II.7).

After checking the defining skills Sampo would similarly make sure that perceptual expectations for the used perceptions exist. We assume that perceptual expectations for the perceptions (BE-ON-FLOOR-OF R202) and (BE-IN-ADJ-TO R20) are not yet known. Sampo asks advice and the master defines them as shown below. Recall that perceptual expectations, marked here as (En), are skills. A reading of a definition of a perceptual expectation roughly
is: After performing this-and-this action sequence, the perception (whose expectation is being defined) might evaluate TRUE. From now on we omit discussion on acquisition and generalization of perceptual expectations; the method is analogous to that of skills. Appendix C shows perceptual expectations of this simulation in their final form.

; how to perceive the floor of R202
; (E1) (BE-ON-FLOOR-OF R202) ::= 
; if recall the floor of R202
; then go to that floor
; 
; (((RECALL ?FLOOR FLOOR-OF R202))
 (GO TO $FLOOR)
 (RELEASE $FLOOR))

; how to perceive an adjacent room to R202
; (E2) (BE-IN-ADJ-TO R202) ::= 
; if recall the adjacent rooms
; then go into the closest one
; 
; (((RECALL ?ROOM ADJ R202))
 (GO INTO $ROOM)
 (RELEASE $ROOM))

In studying (E1) Sampo recognizes $FLOOR as being a new name (it is a reference-variable, all right, but its logical status is not yet known). SKILL-MODIFICATION suggests (S4) or (S6) be used as the skill for (GO TO $FLOOR). Since the master accepts (S4) and rejects (S6), SKILL-MODIFICATION generates a new concept name GO-TO-ABLE-2 (GO-TO-ABLE
already exists with a different logical status), attaches $FLOOR and FLOOR1 as sons of it (see Figure II.7), and generalizes (S4) into the form:

\[(S4') (\text{! GO TO GO-TO-ABLE-2})\]

In the same manner the defining skill (GO INTO $ROOM) engenders a new concept name GO-INTO-ABLE, assigns R202 and $ROOM as sons of it, and generalizes the skill (S7) into:

\[(S7') (\text{! GO INTO GO-INTO-ABLE})\]

This generalization has not, however, been finished yet. Since R202 now has two fathers (ENTER-IN-ABLE and GO-INTO-ABLE), SKILL-MODIFICATION checks whether the fathers are, indeed, aliases or whether, as a consequence of the last performed generalization, either of the fathers increases its generality. If sons of both ENTER-IN-ABLE and GO-INTO-ABLE apply to exactly the same skills, the concepts are aliases. ENTER-IN-ABLE has three sons: ELEVATOR, R105, and R202; GO-INTO-ABLE has two sons as a consequence of the recent generalization: R202 and $ROOM. Since all sons of GO-INTO-ABLE apply to (S2) (the only skill containing ENTER-IN-ABLE), but ELEVATOR does not apply to (S7') (the only skill containing GO-INTO-ABLE), ENTER-IN-ABLE becomes the father of GO-INTO-ABLE (see Figure II.7).
The master requests next:

(! TAKE TABLE)

No skill is found and the modification fails. SKILL-ACQUISITION asks the master to define the new skill. Since TABLE is an existing name, no modifications occur in LTOC.

; the skill for taking a table after looking for it
; (S8) (! TAKE TABLE) ::= 
; always, look for it and pick it up
; ((T)
  (LOOK FOR TABLE)
  (PICK UP TABLE))

The first of the defining skills is exactly (S5). For the second one no skill can be found and modification attempts fail. SKILL-ACQUISITION asks the master to define the skill.

; the skill for picking up a table
; (S9) (! PICK UP TABLE) ::= 
  (EXCLUSIVE 
  ; if not see it 
  ; then it cannot be picked 
  ; ((NOT (SEE TABLE))
    (DONE))
  ; otherwise, go to it and grasp it in on the back
  ; ((T)
    (GO TO TABLE)
    (GRASP-BACK TABLE)))
(S9) is defined totally using existing skills. The master then requests:

(! GO INTO R204)

Since R204 is a new name, no skill is immediately found for it. SKILL-MODIFICATION suggests (S7') be used and, because the master accepts the proposal, R204 is set as a son of GO-INTO-ABLE. No modification needs to be done with respect to the skill itself. The master's last request for the task is:

(! LEAVE TABLE BY WALL)

No skill is found and modification attempts fail. SKILL-ACQUISITION asks the master to define the skill. The procedure further assigns WALL as a son of SOMETHING (no skill containing created concept names applies to WALL).

; the skill for leaving a table by a wall
; (S10) (! LEAVE TABLE BY WALL) ::= 
; (EXCLUSIVE 
; ; if not carrying anything 
; ; the o.k.
; ; (((NOT (SENSE-CARRYING)))
; (DONE))
; ; if have a load and not see anything in the front 
; ; then move forward to a wall, turn until load touches 
; ; and ungrasp the load 
; ; (((SENSE-CARRYING)
; (NOT (SEE OBSTACLE)))
; (MOVE-FORWARD UNTIL (BODY-SENSE FRONT))
; (TURN-RIGHT UNTIL (BODY-SENSE RIGHT-TURN))
; (UNGRASP-BACK TABLE)))
Sampo has understood all given requests in the sense that respective skills have been retrieved, bound, and organized as a list in TASKLIST. The master then issues NIL to signal the end of the task definition. Sampo performs the task as shown in Figure II.9. Some cryptic remarks attached in the figure explain Sampo's overt behavior. The state of LTOC after the definition and execution of the task is shown in Figure II.7. The figure indicates how by executing the task Sampo has become acquainted with new room and door names. Let us now briefly look at the covert side of Sampo's behavior. Since the first request was (! GO INTO R202), the first skill intended for execution (the top of TASKLIST) is:

(EXCLUSIVE

(((BE-IN R202))
(DONE))

(((NOT (BE-IN R202))
 (BE-ON-FLOOR-OF R202)
 (BE-IN-ADJ-TO R202))
 (GO TO R202-DOOR)
 (OPEN R202-DOOR)
 (ENTER IN R202)))

The skill contains two competing tactics; one of those will be selected for execution. The perceptual expert (BE-IN R202) is called; it evaluates FALSE since Sampo is located in R105. But (NOT (BE-IN R202)) evaluates TRUE. The second tactic will hence be selected. Since (BE-ON-FLOOR-OF R202) evaluates FALSE (Sampo is on the first floor), its perceptual expectation (E1) will be executed
prior to the skill. The top of TASKLIST becomes:

(((RECALL ?FLOOR FLOOR-OF R202))
 (GO TO $FLOOR)
 (RELEASE $FLOOR))

RECALL binds the reference-variable $FLOOR into
(FLOOR2) (the set of names that constitute the property
FLOOR of R202) and returns TRUE. Since the whole
precondition evaluates TRUE, it is replaced by (T) to
indicate that it has been satisfied. Its first defining
skill (GO TO $FLOOR) is removed from TASKLIST and parsed,
and the respective skill (S4) is fetched, bound, and pushed
to the top of TASKLIST. Monitoring of the task thus
continues. When a movement (a call to a motor organ)
appears in a defining skill, it is performed.

II.2.3 Task #2

For this task, let us assume the master is located with
Sampo in R105 and wants a basket in that room by a wall, but
the master does not know the location of a basket. The
master requests:

(! FIND BASKET)

Since no skill is found and modification fails,
SKILL-ACQUISITION asks the master to define the new skill.
SKILL-ACQUISITION assigns the new name BASKET as a son of
GO-TO-ABLE after the master has confirmed the applicability of (S6) to BASKET.

; the skill for finding a basket
; (S11) (! FIND BASKET) ::= 
; perceive the current room
; look for a basket in the current room
; then remember the rooms of the house
; and find a basket in those rooms
; 
; (((BE-IN ROOM))
 (LOOK FOR BASKET)
 (REMEMBER-ROOMS)
 (FIND BASKET IN ROOMS)
 (RELEASE $ROOM))

Sampo studies the defining skills. Since (REMEMBER-ROOMS) and (FIND BASKET IN ROOMS) are unknown, SKILL-ACQUISITION asks for the master to define them.

; the skill for remembering the rooms of the house
; (S12) (! REMEMBER-ROOMS) ::= 
; recall all room names
; 
; (((RECALL ?ROOM))
 (DONE))
the skill for finding a basket in rooms
(S13) (! FIND BASKET IN ROOMS) ::= 
  (EXCLUSIVE 
  if see a basket 
  then o.k. 
  (((SEE BASKET)) 
    (DONE)) 
  if recall the location of a basket 
  then go into that room 
  look for a basket there first 
  then start searching (if not found) 
  (((RECALL ?ROOM LOC-OF BASKET)) 
    (GO INTO $ROOM) 
    (RELEASE $ROOM) 
    (LOOK FOR BASKET) 
    (SEARCH FOR BASKET)) 
otherwise, search for a basket 
=((T) 
  (SEARCH FOR MOVABLE-THING)))

One of the defining skills need to be acquired:

the skill for searching a basket
(S14) (! SEARCH FOR BASKET) ::= 
  until either see a basket 
  or there are no more rooms left 
  repeatedly, remember the other rooms 
  go to the closest one 
  and look for a basket there 
  (REPEAT-UNTIL (OR (SEE BASKET) 
    (NOT (RECALL $?ROOM $ROOM))) 
  (REMEMBER-OTHER-ROOMS) 
  (GO INTO $?ROOM) 
  (LOOK FOR BASKET))
; the skill for remembering other rooms
; (S15) (! REMEMBER-OTHER-ROOMS) ::= 
; recall the last recalled room names
; except the last visited one
; (((RECALL ?ROOM $ROOM EXCEPT BE-IN LAST))
 (DONE))

Hence the request (! FIND BASKET) has become well
defined. The master next requests:

(! PICK UP BASKET)

Sampo does not find a skill for the request, and
SKILL-MODIFICATION proposes (S9) be used on the grounds that
TABLE and BASKET are sons of the same father (GO-TO-ABLE).
Since the master confirms the applicability, the procedure
creates a new concept name PICK-UP-ABLE and sets TABLE and
BASKET as sons of it. Again SKILL-MODIFICATION has to
decide the relationship between PICK-UP-ABLE and GO-TO-ABLE
because TABLE and BASKET are sons of both. The master
advises that doors are not PICK-UP-ABLEs, hence GO-TO-ABLE
represents a more general concept than PICK-UP-ABLE (see
Figure II.8). The skill (S9) is generalized into the form:

(S9') (! PICK UP PICK-UP-ABLE)

The master then requests:

(! COME BACK)
Sampo does not find a skill and modification attempts fail. SKILL-ACQUISITION asks the master to define the skill.

; the skill for coming back
; (SI6) (! COME BACK) ::= 
; recall the first visited room and go into that room

(((RECALL ?ROOM BE-IN FIRST))
( GO INTO $ROOM)
( RELEASE $ROOM))

The master requests:

(! LEAVE BASKET BY WALL)

The skill (S10) will be generalized into the form shown below, and a new concept name LEAVE-ABLE is created and located in LTOC as an alias of PICK-UP-ABLE (exactly the same skills apply to their sons).

(S10') (! LEAVE LEAVE-ABLE BY WALL)

Then the master issues NIL indicating the end of the task definition. Sampo's overt behavior with comments is shown in Figure II.10. Figure II.8 shows the state of LTOC after the definition and execution of the task.

Sampo should recall, based on its own actions, that a basket is now located in R105. To demonstrate this, let us first ask it to go to another room and again find a basket.
The simulation of the task is shown in Figure II.11.

II.2.4 Task #3

As still another task, let us assume that the master wants an additional chair placed in room R206 by the table. The master requests:

(! BRING CHAIR INTO R106 BY TABLE)

Sampo does not find a skill and SKILL-ACQUISITION asks the master to define one (below). For the defining skills of (S17) that have not yet been exhibited, see Appendix A. Assignment of R106 and R106-DOOR as sons of GO-INTO-ABLE and GO-TO-ABLE represent the only modifications in LTOC. The simulation of the task is shown in Figure II.12. Appendix A shows skills in their most general form, Appendix B describes the respective LTOC of the skills, and Appendix C exhibits the perceptual expectations.
the skill for bringing a chair into R106 by a table

(S17) (! BRING CHAIR INTO R106 BY TABLE) ::= 
  (EXCLUSIVE
  
  if are in the target room
  then recall an adjacent room, go there first
  and then start bringing
  
  (((BE-IN R106)
    (RECALL $ROOM ADJ R106))
  (GO INTO $ROOM)
  (RELEASE $ROOM)
  (BRING CHAIR INTO R106 BY TABLE))

otherwise remember all rooms except the target room
look for a chair in the current room first
then find it in the remaining rooms
pick up a chair, go into the target room
and leave the chair there by the table

((T)
  (REMEMBER-ROOMS EXCEPT R106)
  (LOOK FOR CHAIR)
  (FIND CHAIR IN ROOMS EXCEPT R106)
  (RELEASE $ROOM)
  (PICK UP CHAIR)
  (GO INTO R106)
  (LEAVE CHAIR BY TABLE)
Figure II.6: LTOC in the Beginning of Simulation
Figure 11.7: LTUC After Task #1
FIGURE II.8: LTOC After Task #2
Step 1

Turns around to look for R101-DOOR (R101 adjacent to the elevator). Goes to the door.

Step 2

Looks for the elevator door. Obstacles:
First goes straight.
Attempts to circle to the left. Succeeds.

Step 3

The elevator has been called. The door has opened. Has gone into the elevator. Pushes the 2nd floor button.
Step 4
The elevator is on the 2nd floor.
Exits from the elevator.
Turns around to look for R202-DOOR.
Goes to it.

Step 5
Enters R202.
Turns around to look for a table.
Obstacles!
Attempts to circle to the left.
To no avail.
Attempts to circle to the right.
To no avail.

Step 6
Removes an obstacle.

Step 7
Goes to the table.
Grasps it on the back.
Attempts to turn around to look for R204-DOOR.
Cannot turn!
Forward to get free space.
Turns around to look for R204-DOOR.
Goes to it.

Step 8
Pushed the button.
The door has opened.
Step 9
Moves forward.
Obstacle!
Grasps it on the front.
Forward until in R204.
Releases the obstacle.

Step 10
Turns to the left until sees free way to a wall.
Forward until by the wall.
Turns to the left until the table touches the wall.
Releases the table.
DONE.

Task '2

Figure II.10

Step 1
Turns around to look for a basket.
A basket not in the room.
Turns around to look for R101-DOOR
(decides to go into R101 next).
Goes to the door.

Step 2
Enters the room.
Turns around to look for a basket.
A basket not in the room.
Turns around to look for R102-DOOR
(decides to go into R102 next).
Goes to the door.
Step 3
Enters the room. 
Turns around to look for a 
basket. 
A basket not in the room. 
Turns around to look for 
R103-DOOR 
(Decides to go into R103 next). 
Goes to the door.

Step 4
Enters the room. 
Turns around to look for a 
basket. 
Succeeds!

Step 5
Has gone to the basket. 
Has grasped it on the back. 
Pushes the door button.

Step 6
Has entered R102. 
Has gone to R105-DOOR. 
Has pushed the door button. 
The door is open.

Step 7
Has seen straight ahead a free 
way to a wall. 
Has moved forward until comes 
to the wall. 
Turns around until the basket 
touches the wall. 
Releases it.
DONE.
Step 1
Has moved into R103.
Should remember basket in R103.

Step 2
Has looked for a basket in R117.
Leaves the room.

Step 3
Goes straight to R103-DOOR.

Step 4
Enters the room.
Step 5
Looked for the basket.
Went to the basket.
Grasped it on the back.
Goes to R102-DOOR.

Step 6
Looked for R103-DOOR.
Goes to it.

Step 7
Enters the room.
Goes to a wall.
Turns until basket touches.
Ungrasps the basket.
DONE.

Figure II.12
Task #3

Step 1
Looks for a chair first
in the current room.
Goes to R101-DOOR.
Step 2
Looks for a chair.
Sees one.
Goes to it.

Step 3
Grasps the chair on the back.
Looks for R105-D005..
Goes to it.

Step 4
Enters the room.
Looks for R105-D005..
Goes to it.

Step 5
Opens the door.

Step 6
Enters the room.
Goes by the table.
Leaves the chair.
DONE.
This chapter presents the Conceptual Model of an agent. We do not yet call the artifact a robot because the agent possesses some utopian developmental features with no clear computational solution. This sketchy Conceptual Model shows the constituents of the underlying system and their structural relationships, but does not discuss them in detail. A more detailed look is taken when the Computational Model is discussed in Chapter IV. The Computational Model will present a restricted realization of this utopian Conceptual Model. In the Computational Model some developmental features of the agent are replaced by advice-taking from a human master.

The constituents of the Conceptual Model can be studied along two independent dimensions. They can be viewed against the passive-active dimension, which, because the symbol manipulation paradigm for explaining the 'mind' of the agent is assumed, boils down to the question of whether or not a constituent is a manipulator or a manipulatee. The constituents which manipulate are henceforth called procedures, and those which are manipulated are called declarations.
The other classifying dimension naturally follows from the importance this paper attaches to development. Traditionally, research on robotics has almost entirely dealt with constituents that explain a robot's behavior from the moment a task is given to its successful end. Let us call this view the Behavioral Abstraction (BA) of an agent. In this paper the constituents which explain why and how the constituents in the BA have developed are also of great importance. This view is called the Developmental Abstraction (DA) of the agent. The constituents of the BA explain a single behavioral pattern; the constituents of the DA explain the progression of the agent's competence.

The following notational convention has been devised to depict a simple schematic representation for the conceptual model. Procedures are represented as arrows and declarations as rectangles. An arrow exits those declarations which provide its input and points to those which serve its output. If a declaration is not manipulated on every activation of a procedure, the corresponding arrow is shown as a broken line.
III.1 BEHAVIORAL ABSTRACTION

The BA consists of the following declarations: the Propositional Memory, the Perceptual Memory, orienting maps, and Logical Taxonomy of Concepts. All of these comprise an umbrella declaration called the Memory. The BA also contains the master-agent interface declarations: requests, questions, claims, and answers. Naturally, since the agent 'perceives', some sort of sense data also belongs to the class of declarations. The abstraction further consists of perception procedures, RECALL procedure for retrieving recorded perceptual experience, skills (composite procedures of motor movements, perception procedures, and RECALL procedure), and MONITOR procedure for monitoring execution of skills. Furthermore, BA contains procedures for updating the Memory by the master-given facts, answering the master's questions, and some lower level utility procedures. Each of the constituents, except the master-agent interface, is discussed in this chapter in its own section.

III.1.1 Logical Taxonomy of Concepts

We have explained in great length earlier in this work how this agent does not possess a comprehensive symbolic world model to be manipulated in a planning phase. Instead, the agent solves problems directly perceiving states of the
world in utilizing skills. The constituents which become closest to the idea of a world model are orienting maps (described later) and Logical Taxonomy of Concepts (LTOC). We locate LTOC inside the Memory although it develops from action experience, as will be seen, whereas the other parts of the Memory develop from perceptual experience.

The agent's environment initially represents a chaos to the agent. In the beginning, from the agent's viewpoint, the environment does not contain any structure. Gradually, individuals of the world begin to depart from the chaos through acts the agent learns to perform on them. ('Individual' in this context refers to a thing in the environment that possesses perceptual invariance, can be named, and which the agent can be requested to act upon. The syntactic categories 'proper name' and 'noun' contain individuals. The names of individuals are assumed to be strictly labels. Hence, JOHN is a name of an individual, and so are R101 and TABLE.) For example, when the agent has learned (possesses skills) to go into R101 and pick up a TABLE, its 'understanding' of the world has been structured to the effect that R101 is a 'go-into-able' and TABLE is a 'pick-up-able'. It may further attempt to apply the two skills the other way around and further increase the 'understanding' that R101 is not a 'pick-up-able' and TABLE is not a 'go-into-able'.
Obviously, the contention that skills should carry with them the names of applicable and non-applicable individuals is not feasible. Instead, a dedicated constituent (declaration), external to skills, provides a more handsome solution. LTOC represents such a classificatory constituent. It is organized as a hierarchical member-set relation. It forms a tree whose root represents the concept SOMETHING, which has a number of member sets, which in turn have member sets and so on, until each branch of the tree terminates on names of individuals. Leaves of the tree may be either ordinary nouns (denote several individuals in the environment, say TABLE) or proper names (denote a single individual, say JOHN). All classifying concept names, located between leaves and the root of LTOC, are generated in the developmental process (by the SKILL-MODIFICATION procedure) to reflect applicabilities of skills. Hence, for instance, the concept name GO-INTO-ABLE has as its descendants in LTOC (denotes) the names of individuals in the surrounding to which the skill for 'going into' applies. (Obviously these individuals are room names.)

Notice that LTOC is not to be understood as representing knowledge in the sense of a collection of abstracted intensional concepts. It does not capture any intellectual properties of concepts; it represents only an ordering of individuals based on the manner they participate in actions. In this sense LTOC represents a logical arrangement. The generated classifying concept names come
close, we believe, to what [Gibson] calls affordances: a generated concept name reveals what actions its descendant individuals 'afford'. LTOC discriminates for the purpose of providing scopes for the skills. Every living organism, so it seems, must possess such mechanism in order to survive. Organisms must, for instance, be able to discriminate food particles and its enemy organisms in order to be able to invoke different actions in their presence.

Since LTOC exhibits scopes of skills, its development must somehow parallel development of skills. The DA in the next section depicts how this development takes place. It suffices here to say that the agent starts with the given concept name SOMETHING and gradually molds it under complementary formation processes of splitting and gathering to reflect the agent's ever increasing understanding of the 'logical' order of its environment. At any moment LTOC together with skills reflects the agent's developing action-based understanding of the environment. Following the empiricist's assumption that every non-logical concept is reducible to sensation plus motor concepts via coordinating definitions, LTOC provides such a link to motor definitions.
III.1.2 Perceptual Memory

The Memory comprises, you may recall, a passive collection of functionally separable declarations. One of those declarations, LTOC, was described in the preceding section. The remaining constituents of the Memory record perceptual experience in one way or another. One part of the Memory, called the Perceptual Memory, faithfully records successful perceptions as a list of traces of percepts in the order they have occurred. The remaining constituents of the Memory utilize these traces as sources for abstractions.

Organizationally, the Perceptual Memory represents a one-dimensional list. Hence, different modalities of perception do not structurally manifest themselves. An immediate observation shows that if traces of all percepts are recorded, the content of the Perceptual Memory monotonically grows. Since the agent's memory is necessarily finite, some means of discarding traces from the Perceptual Memory must be devised. A straightforward solution is to cut the agent's lifetime into short enough intervals so that during each of the intervals all perceptual traces are recorded, and after the interval, all traces are lost. A natural time-slicing rule, adopted here, considers tasks independent from each other. When a new task is requested, the Perceptual Memory does not contain any traces from previous tasks.
It is left to the Computational Model to give a precise account of the form of traces of percepts. It suffices here to note that since perceiving is performed by perceptual expert procedures of perceptual schemas (see III.1.7 for details), a trace must at least indicate which expert did the perceiving and under what conditions. A perceptual expert procedure will evaluate TRUE or FALSE depending on whether the state of affairs on which the expert possesses expertise does or does not hold in the environment. If perceiving is successful, the trace of it will be stored in the Perceptual Memory. It follows that a trace stores information about an equivalence class of states of affairs. For example, a trace of the percept of a visual chair contains information about the equivalence class of chairs in different orientations. Should there exist a means of 'traversing' a perceptual expert the other way around — from its call to its input (a loose analogy of a cathode ray tube) — it could be said that the Perceptual Memory contains perceptual images. Notice, however, that perceptual experts in this system perform a many-to-one mapping. Hence, even if such a converse process were possible, no individual 'images' could be regenerated.
III.1.3 Propositional Memory

The Perceptual Memory, defined in the previous section, stores traces of percepts without further manipulation. The Propositional Memory, on the other hand, contains propositions which expert abstraction procedures have inferred from a content of the Perceptual Memory. Propositions can also be abstracted from the agent's own actions, or the master may have given them. A detailed account of perception-based abstraction is given in the Computational Model.

An example of a predicate which has obvious importance to the agent and should be abstracted is 'location'. It is easy to conceive how the abstraction procedure abstracts that predicate. If the agent visually perceives a table, for instance, a trace of the percept is stored in the Perceptual Memory. If the agent also perceives the current room, a trace of that percept is also stored in the Perceptual Memory. From those traces the proposition 'a table located in such-and-such room' can be abstracted and stored in the Propositional Memory.

We wish to distinguish propositions that describe a map-like spatial organization of the surroundings from other propositions. The propositions of the former type are stored in constituents of the Memory called orienting maps (discussed in the next chapter), all other propositions are stored in the Propositional Memory. Hence, the
Propositional Memory contains abstracted propositions which have importance for the agent's well being and which do not involve considerations of locomotion.

A slight departure has taken place from the initially proclaimed purist intent that the agent does not possess a world model since the Propositional Memory and orienting maps represent rudiments of a partial world model. Their use naturally augments the power of skills without severe consistency problems that are inherent in a comprehensive world model on which a problem solver totally relies. An empty or imperfect Propositional Memory does not prevent the agent from performing its duties.

III.1.4 Orienting Maps

Orienting maps resemble the Propositional Memory. An orienting map depicts the geographic organization of the agent's habitat in the form of a set of propositions. The agent may possess different orienting maps with different scales just as we may have different ordinary maps. One orienting map might describe the spatial organization of the rooms of a house, another might outline how neighboring houses are spatially interrelated, and still another might describe how cities are located in relation to each other.
Abstracting propositions contained in orienting maps necessarily involves considerations of locomotion. It may first seem unwarranted that a negation of a single feature would necessitate the establishment of two functionally separate declarations (abstracting the Propositional Memory does not involve locomotion), but both practical and pseudotheoretical reasons exist for doing so. Let us take the practical aspect first. If locomotion can be excluded from consideration, abstractions from a list of traces of percepts represent fairly straightforward tasks. Once consideration of locomotion is added, the complexity of abstraction increases by a quantum leap. Traces of percepts, the domain of abstraction, must now include information about directions, speeds, and durations of movements. Proper handling of these additional characteristics is a matter of some difficulty.

A pseudotheoretical argument for separating the Propositional Memory and orienting maps might be based on the fact that an agent without adequate orienting knowledge is 'lost' in a majority of tasks, whereas an empty Propositional Memory does not prevent the agent from acting. Contents of orienting maps and the Propositional Memory seem to possess qualitatively different survival values for the agent. Propositions for the Propositional Memory can be abstracted as secondary activities with respect to primary behavioral activities, but abstracting propositions of the agent's habitat seems to require dedicated behavioral
activities. For example, that room R101 is adjacent to room R102 must be found in a separate exploratory activity. The assumption that it could be found as a side-effect of some other activity raises the question what such an activity might possibly be? Aimless wandering around the house seems to be the only such possibility. The moment we assume any primary purposeful non-orienting activity involving locomotion, it seems we also must assume the existence of an orienting map.

III.1.5 Requests, Questions, Claims, and Answers

We assume communication between the agent and its master to be conducted in a restricted natural language. At least the following four types of sentences are admissible: requests by the master to the agent to perform tasks, claims by the master to express facts about the world bypassing the agent's own perceptual 'experience', questions by the master, and answers by the agent to the master's questions.

III.1.6 Skills

Skills represent the most important procedures of the Behavioral Abstraction. Figure III.1 depicts a skill as a procedure composed, in the final analysis, of calls of
perception and RECALL procedures, and movements. When a skill is activated — it becomes an action — the perception and RECALL procedures as defined in the skill begin to control activations and deactivations of the motor organs in the skill. Both perceptions and RECALL procedures are described in more detail in the following sections. In this section we reflect on some conceptual requirements for well-formed skills. The view that motor movements and perceptions form a highly interwoven structure in skills has also been proposed by Turvey concerning human actions:

But it must be necessarily the case that, like warp and woof, perception and action are interwoven, and we are likely to lose perspective if we attend to one and neglect the other; for it is in the manner of their union that the properties of each are rationalized. After all, there would be no point of perceiving if one could not act, and one could hardly act if one could not perceive [Turvey, p. 211].

To say that perceptions and motor movements are intermixed in an orderly fashion calls for rules for such structures. The Computational Model will define exact rules for building the agent's skills. It suffices here to note that every skill requires a certain perceptual setting (and/or recalling) to be satisfied for the activation of motor organs to occur and, also, some perceptual setting to discontinue activated movements. As discussed in the Introduction, the use of perception in control seems to represent the most striking difference between skills and plans. To summarize, the first design goal for the skill structure aims at an intermixture of calls of perceptions.
The second design goal deals with the vertical structuring of skills. The degree of freedom a human muscular system possesses is, indeed, great. The agent obviously lacks such sophistication, but even in its case, the degree of freedom of its motor organs may cause serious control problems if skills are heterarchical. In such a case each skill should explicitly define control of each individual motor organ. A clear alternative is to gradually decrease degrees of freedom by imposing a hierarchy on skills. That is, the lowest level skills are exclusively defined using motor movements, perceptions, and recalling. Higher level skills may use the lower level ones as autonomous units and augment them with appropriate perceptual control structure. At each level in the hierarchy, a skill need not get involved in decisions executed on lower levels. This results in an orderly decrease in the control of degrees of freedom. Recursion may be also imposed to further decrease the explicit control need. 'Recursion' implies that in building up a skill the skill itself can be used as if it were already known. To summarize, the second major design goal for the skill structure is application of hierarchy and recursion.

The third design goal follows from the developmental nature of the agent. If the agent develops, at least its skills should develop. That observation and our attainment of a computational explanation in turn entail that skills
should possess a form that yields easy symbol manipulation. Since declarations are clearly more advantageous than procedures in this regard, skills should be located on the declaration side in the procedure-declaration dichotomy. Skills should possess, then, a dual nature: in the Behavioral Abstraction they serve as procedures and in the Developmental Abstraction (where they develop), as declarations. In LISP language both procedures and data are lists, thus making the dichotomy procedures vs. declarations purely role dependent. A list represents a procedure if it assumes such a role, but the very same list can be a declaration (data) if in turn such a role is assumed. The third design goal is then the list structure of skills. Of course, we do not claim human skills are stored as list structures.

It is a task of the Computational Model, depicted in the next chapter, to come up with a detailed description which satisfies all mentioned design goals. Finally, let us briefly look at what epistemological implications the proposed skill system bears. This discussion will be conducted in greater length in the section Knowledge. To use a Piagetian term, we view the increasing collection of skills the agent possesses as its sensori-motor level of intelligence. When the agent acquires a new skill, the agent's implicit action-based knowledge of the world increases exactly at the moment the piece of knowledge is needed and in the most applicable form.
III.1.7 Perceiving

Human perception has been the subject of vast numbers of psychological and philosophical inquiries. Our intention is not to add anything to such discussions but, rather, to achieve a functionally satisfactory faculty of perception for an artificial agent. Furthermore, an obvious additional requirement is that functionally adequate perceptual faculties should be realizable by mechanical means. Terms such as 'perceiving', 'perception', and 'percept' will from now on refer to such artificial faculties. In this Conceptual Model, you may recall, we use the dichotomy procedures vs. declarations in describing any constituent. Perceiving as a procedure has two separable functions: it controls behavioral patterns as an integral part of a skill as depicted in Figure III.1, and it loads the Memory with a list of traces of percepts representing perceived facts from surroundings as schematically shown in Figure III.2. Traces of percepts provide the source for perception-based knowledge for the agent. The other parts of the Memory, the Propositional Memory and orienting maps, are updated by abstraction procedures that manipulate these traces of percepts.

We naively mimic Neisser's cyclic model of perception where the act of perceiving is performed by (perceptual) schemas: "A schema is that portion of the entire perceptual cycle which is internal to the perceiver, modifiable by
experience, and somehow specific to what is being perceived, "it directs movements and exploratory activities that make more information available" [Neisser, p. 54]. In this context perception consists of a collection of (simple) schemas. Each of the schemas, containing a perceptual expert procedure (expert, for short) and a perceptual expectation skill (expectation, for short), has developed and will further develop perceptual abilities within its limited scope. Experts perform the act of perceiving. Since we have broken a perceptual faculty into a collection of narrowly proficient schemas, it suffices for the expert of a schema to respond TRUE or FALSE (only) depending on whether the state of affairs holds or not where the schema possesses expertise. The expert simply confirms or rejects an anticipation. An example of an expert is 'see door' which responds TRUE if an image of a door exists in the visual field at the moment of the call, otherwise the response is FALSE.

The other constituent of a perceptual schema, a perceptual expectation, is a skill that may be activated, in the case the respective expert responds FALSE, to perform such bodily movements which might result in successful perceiving. For example, an expectation for 'see door' expert might be: 'turn head until see door'. Introduction of expectations embodies the agent with execution time sub-goal creation mechanism as will be seen in more detail in the Computational Model. The cycle in a perceptual
schema is completed by perceptual development, that is described in the DA in the next section.

From the manner perceptions control skills — providing preconditions and goal conditions for motor movements — it follows that the agent always attempts to perceive some specific perception and pays attention only to such perceptions that it has anticipated. Hence, the agent's behavior is perception-controlled. Contrast this with even controlled behavior where an external state of affairs may trigger changes in the behavior of the agent regardless of its inner state at that moment.

III.1.8 Recalling

We have described how the Memory records the agent's perceptual experience in both 'raw' and inferred forms. This perceptual experience can be reflected on both for the purpose of answering questions and for controlling behavior (Figure III.1). To cover both of these functions we define the procedure RECALL which scrutinizes any constituent of the Memory (the Perceptual Memory, the Propositional Memory, orienting maps, or LTOC), responds TRUE or FALSE depending on whether recalling is successful or not, and returns the names of individuals from the Memory that were causes of a successful recalling. RECALL establishes an indirect reference to names of individuals through the use of a
reference-variable (see the Computational Model for details). In a skill definition, reference-variables can be used to refer to recalled individuals. The function of a reference-variable resembles the function of a pronoun in natural language usage. For example, assuming that LOCATION is a predicate used in the Propositional Memory, recalling can be used to establish an indirect reference to a room name that is the location of a named object. If recalling succeeds, that room name can be referred to, for instance, in requesting that the agent go there.

III.1.9 Monitoring

A master assigns tasks to the agent as a list of requests. The system procedure MONITOR parses requests, fetches respective skills, activates them, and monitors their execution until the task terminates (Figure III.3). During an ongoing task, a new task may be assigned. The agent assigns priorities for tasks; and if a new task obtains a higher priority than an ongoing task, the active task is suspended. The agent may even possess a permanent low priority task, such as 'curiosity'. It would move around the surroundings, perceive everything it can (activate all perceptual experts), and abstract propositions into both the Propositional Memory and orienting maps.
The agent is always in either a goal-seeking state or in an inactive state. Whenever the agent is active, it is pursuing a specific goal. This feature greatly reduces both ambiguity and increases efficiency in the perceptual processing. Each skill has a certain perceptual precondition which must be satisfied before the skill is activated. Similarly, any movement has a perceptual condition to terminate the movement. Each of perceptual conditions consists of calls of perceptual experts, hence, the agent, if performing a task, either attempts to perceive specific perceptual settings of a precondition or a termination condition. It follows that the agent 1) does not recognize perceptual ambiguity (the environment either satisfies or does not satisfy an activated perceptual expert), and 2) processing is efficient (only those aspects in the environment which have relevance to the activated perceptual expert need to be considered).

III.1.10 Fact Manipulation

The primary goal of this Conceptual Model centers around behavioral capabilities of the agent, by and large neglecting its question-answering abilities. In this section, however, we give an inconclusive and partly speculative account of epistemological aspects of the proposed system. We separate "knowledge through
perceiving', 'knowledge through being told', and 'knowledge through acting'. Acquisition of 'knowledge through perceiving' has already been dealt with in the sections describing the Memory. It remains to reflect on the other two types of 'knowing'. The system procedure FACT-MANIPULATION is designated for manipulating these types of 'knowing'.

We have reserved a class of natural language sentences, called claims, for the master to express facts about the world and bypass the agent's perceptual experience. FACT-MANIPULATION procedure updates the Memory based on the facts expressed in claims (Figure III.4). Once the Memory is updated, nothing separates 'knowledge of being told' from 'knowledge through perceiving' in the Memory.

'Knowledge through acting' represents the agent's practical intelligence. We argue that skills collectively represent the agent's presymbolic sensori-motor level of intelligence in the Piagetian sense. Skills logically structure and differentiate the surroundings in the sense that "To know is to transform reality in order to understand how a certain state is brought about" [Piaget, p. 15]. Clearly, abstracted conceptual knowledge represents an advanced form of knowing. Piaget believes that abstraction is drawn from a concrete action basis:

In this hypothesis the abstraction is drawn not from the object that is acted upon, but from the action itself. It seems to me that this is the basis of logical and mathematical abstraction [Piaget, p. 16].
If skills successfully represent such an action basis, a possible future research topic would be to find a computational account for this 'abstraction from actions'. In this paper the action-based knowledge the agent possesses in its skills is coined diffuse to separate it from a more common discrete knowledge that conceptual structures provide. If this Piagetian contention that high level cognitive structures have grown out of precognitive substructures is right, then the stand has strong epistemological implications. A knowledge structure that is intimately related to lower level substructures is 'richer' than knowledge structures that lack such a base.

Diffuse knowledge can provide answers both to 'how' and 'what' questions. For a question 'How is such and such an action performed?', FACT-MANIPULATION procedure scrutinizes the skill for the action and generates appropriate answers. Answers can be given in different degrees of details by moving up and down in hierarchical representations of skills. If a question is of the type 'What is such and such a concept?', it is treated in the spirit of diffuse knowledge as if it were given in the form 'How is such and such a concept used in skills?'.

Initially, the agent's knowledge sources are empty, and it can only declare its ignorance to questions. During its development, in acquiring new skills and executing existing skills, the knowledge sources increase in content as does the agent's competence in answering questions.
Some results from experimental psychology can be viewed as supporting action-based semantics. Bugelski reports how, in an association test, a group of college students respond in concrete terms even to abstract concepts.

If you say ANIMAL, ..., they report dogs and horses and, again, highly specific animals at that: their own, their neighbors— they specify breeds, colors, actions. If you say DEMOCRACY, they report a variety of imagery, practically none of which refers to governmental operations. Government by people becomes an image of a crowd at a political rally [Bugelski, p. 1006].

Bugelski even questions "... whether there is such a thing as an abstract thought or abstraction..." [Bugelski, p. 1008]. Bugelski does not claim that those findings support action-based semantics — he does not say anything about it. The experiments are argued to support the relevance of the use of images in thinking. We believe that they can also be used to support the argument that abstract concepts have action-based semantic roots. The act of voting, for example, is, according to this argument, inherently imbedded in the meaning of 'democracy'.

Menyuk, in discussing acquisition and development of language, provides support to action-based semantics in referring to a study which shows that "At about age 6 there is an 82 percent usage of definitions in terms of concrete action (for example, 'bottle' defined as 'Where you pour something out of'). By age 15 there is 33 percent usage of these types of definitions. At age 13 a definition for bottle is 'a container in which all liquids go.'" [Menyuk,
Finally, a temptation to attempt to interpret Wittgenstein overcomes the fear of committing an error in the enterprise. The view that action-schemas, or skills as we call them, participate in structuring the surrounding world for an agent can be used in an attempt to explain vagueness of concepts. Vagueness of concepts was a topic which the late Wittgenstein often returned to. Malcolm tells how Wittgenstein once talked about measurement. He considered a tribe of people who would measure lengths counting steps. If different results for the same length would come out, they would think nothing of it, claims Wittgenstein. If somebody from outside would come along with a tape measure to show accuracy, they would not be interested: "The notion of a more accurate measurement does not enter into their lives, and so the notion of the real length does not either. If we say 'They must have the notion of a real length', this is only because we imagine a more complicated life in which one method of measuring is preferred to another. But that is not their life" [Malcolm, p. 33]. Are we far wrong if we take the tribe having its concept of length strongly imbedded in its precognitive action structures, in this case measuring, in a way that does not allow understanding of accuracy. To understand accuracy one has to first abstract 'length' from any particular measuring action.
**Figure III.1:** Skill as a Procedure

**Figure III.2:** Perception as a Procedure

**Figure III.3:** MONITOR as a Procedure
Figure III.4: FACT-MANIPULATION as an Updating Procedure

Figure III.5: FACT-MANIPULATION as an Answering Procedure
III.2 DEVELOPMENTAL ABSTRACTION

While the BA exposed in the previous section explains the constituents which create and monitor appropriate behavioral patterns, the DA described in this section explains how some of those constituents have come into being through gradual development. A continuous progression exists in the sensori-motor intelligence of the agent, and the DA provides an explanation of this progression by defining procedures that represent such mechanisms.

System procedures, such as RECALL, FACT-MANIPULATION, MONITOR, and procedures which abstract from perceptual experience, are initially given and remain fixed. The developing constituents of this system are skills, the Logical Taxonomy of Concepts (LTOC), perceptual schemas, and orienting maps. Even though the contents of both the Perceptual Memory and the Propositional Memory increase during the agent's lifetime, we do not consider them to belong to the DA proper.

Development of skills and LTOC occur in parallel because the latter contains scopes of the former, that is, applicability ranges of skills are indicated in LTOC as explained in III.1.1. Two distinct, complementary procedures take part in their development. SKILL-MODIFICATION procedure attempts to expand the scopes of the existing skills to cover an ever increasing amount of actions. SKILL-ACQUISITION procedure, on the other hand,
creates a new skill if SKILL-MODIFICATION fails to digest a new requested action. Separate procedures account for both perceptual and orienting development.

III.2.1 Skill Modification

Piaget calls the first developmental period of the infant the sensori-motor level because the infant lacks the symbolic function. The child does not have representations by which he can evoke persons or objects in their absence. He solves problems of action during this period by constructing a complex system of action-schemas and organizing reality in terms of spatio-temporal and causal structures without representation of thought, claims Piaget. There is a continuous progression from spontaneous movements and reflexes to acquired habits and from the latter to intelligence. According to Piaget, first of all this mechanism consists of assimilation whereby reality data are treated and modified in such a way as to become incorporated into the structure of the subject, into the existing action-schemas. Schematism, on the other hand, needs to be modified and enriched to correspond to the demands of reality. Piaget calls this modification of internal structures fitting into reality accommodation. The reader may refer to the excellent summary [Piaget and Inhelder] for explanation of the Piagetian stand on the development of the
child.

We loosely mimic these two Piagetian processes. In our model a distinction is made between modification of existing action-schemas (skills), taken care of by SKILL-MODIFICATION procedure, and creation of totally new skills, the task of SKILL-ACQUISITION procedure. This section briefly describes SKILL-MODIFICATION; SKILL-ACQUISITION is discussed in the next section.

Collectively speaking, skills can manifest deficiencies in dealing with reality in two ways. First, an individual skill may be structurally inadequate to cope with a new action instance. For example, a general skill may exist for going into any named room. Later on the agent finds out that an access to a particular room requires special considerations, for instance because its door opens in a peculiar manner. SKILL-MODIFICATION should be able to perform these kinds of corrective modifications to skills. Another form of skill modification is enlarging the applicability of the existing skills. SKILL-MODIFICATION procedure is invoked if a request is received for which DESIRE procedure cannot retrieve a skill (Figure III.6). SKILL-MODIFICATION attempts to find an existing skill whose scope could be made wide enough to cover the current action. In case of success the skill is updated. Recall that the scope of a skill is displayed in LTOC: a skill is applicable to individuals which are descendants of the concept names in the skill definition. Making a skill more
general corresponds then to replacing concepts in the skill by more general ones.

In case SKILL-MODIFICATION succeeds and proper generalized concepts exist in LTOC, it suffices to update the skill. For example, let us assume LTOC contains the concept names PICK-UP-ABLE, HAMMER, and BOOK so that PICK-UP-ABLE is the direct generalization of the other two. If a skill for carrying HAMMER exists, and the agent receives a request to carry a BOOK, and SKILL-MODIFICATION succeeds (the very same skill can be used for both carrying BOOKS and HAMMERS), then the skill is updated for carrying PICK-UP-ABLE. It may, however, be the case that SKILL-MODIFICATION succeeds only under the conditions that a new concept is created to reflect the proper level of generality with respect to the pending skill. Then both the skill and the LTOC should be updated. The Computational Model discusses SKILL-MODIFICATION in greater detail. SKILL-MODIFICATION generalizes skills and progressively associates concepts in the LTOC. For example, refer to Figures IV.5 and IV.6 where SKILL-MODIFICATION associates all DOORS and PICK-UP-ABLEs under a created concept name LOOK-FOR-ABLE.
III.2.2 Skill Acquisition

SKILL-ACQUISITION is a procedure which receives control if a skill for a request does not exist, and SKILL-MODIFICATION fails to generalize existing skills. It is the duty of SKILL-ACQUISITION to create a new skill (Figure III.7).

Just as SKILL-MODIFICATION has two developmental roles in both associating concepts and modifying skills, so SKILL-ACQUISITION in addition to creating skills also differentiates concepts in the LTOC. When SKILL-ACQUISITION creates a new skill for a new individual of the surroundings, the agent in some sense becomes acquainted with the individual. SKILL-ACQUISITION performs this 'get-acquainted-with' function by attempting to locate the new concept name as a member of as specific a classifying concept as possible. In other words, it attempts to differentiate the least general created concept with the new name. The attempt proceeds through application tests of the existing skills. If SKILL-ACQUISITION does not succeed, the new name becomes a son of SOMETHING. For example, refer to Figures IV.7 and IV.8 where SKILL-ACQUISITION differentiates SOMETHING with ELEVATOR, the name of a new individual, after acquiring a new skill for (! ENTER IN ELEVATOR).

In the beginning the agent's LTOC contains a single concept SOMETHING. From that seed SKILL-MODIFICATION and SKILL-ACQUISITION progressively mold an increasingly
elaborate generalization tree.

III.2.3 Perceptual Development

The Perceptual Development procedure creates and develops perceptual schemas (Figure III.8). Obviously perceptual development with artificial means represents an extremely difficult undertaking. The procedure is displayed here only for the sake of completeness of the model. As will be seen, in the Computational Model all perceptual schemas will be initially given.

III.2.4 Orienting Development

Orienting development procedure creates and shapes orienting maps (Figure III.9). Recall that those maps describe the ways the environment is spatially structured. Also orienting development is mentioned here only for the sake of completeness of the model.
Figure III.6: SKILL-MODIFICATION as a Procedure

Figure III.7: SKILL-ACQUISITION as a Procedure
Figure III.8: Perceptual Development as a Procedure

Figure III.9: Orienting Development as a Procedure
IV COMPUTATIONAL MODEL

The Computational Model as described in this chapter replaces some developmental features of the Conceptual Model with advice-taking from a human master and shapes the Conceptual Model into a form that can be implemented in a computer. From here on the agent is renamed a robot to reflect its computational nature.

Particularly, the robot is only a partial realization of the agent in that the master defines skills and orienting maps, and perceptual experts are initially given, fully developed procedures. The robot, however, has limited capabilities for modifying existing skills, but even in that regard the master must confirm results of such a process. That is, the robot suggests modification instances, and the master either confirms or rejects them.

The structuring of this chapter preserves the structure of the previous chapter. If the reader wishes some conceptual flesh around the bare bones of this formal computational model, he can read the respective sections of the Conceptual Model.
IV.1 BEHAVIORAL ABSTRACTION

The Behavioral Abstraction describes those constituents and their interrelations that take part in creating proper behavioral patterns for given tasks.

IV.1.1 Logical Taxonomy of Concepts

The Logical Taxonomy of Concepts (LTOC) is the part of the Memory which discriminates individuals in the surroundings the robot has become acquainted with. Organizationally, it is a member-set-superset hierarchy (tree). LTOC does not contain any intensional knowledge other than set memberships and/or set inclusions. The root of the tree constitutes the initially given concept SOMETHING from which any 'known' concept can be reached along a single path. LTOC develops in parallel with skills in part reflecting the robot's increasing practical 'understanding' of the surroundings.

(1) LTOC is a two-way linked tree indicated with the properties FATHER and SONS attached to concept names:

(SOMETHING (SONS <concept1> <concept2> ...))

...( <concept-i1> (FATHER <concept-i>)
      (SONS <concept-i11> <concept-i12> ...))
IV.1.2 Perceptual Memory

The Perceptual Memory represents a list of traces of percepts, i.e., calls to perceptual expert procedures that have returned TRUE indicating successful perceiving. The system will set the Perceptual Memory to the empty list before execution of a new task. The ordering within the list follows the temporal ordering of respective perceptions.

(2) The Perceptual Memory is a list:

( ... <trace of a percept> ... )

e.g., ((BE-IN R101) (SEE TABLE) ...)

IV.1.3 Propositional Memory

The Propositional Memory contains such abstracted propositions, inferred from traces of percepts (a content of the Perceptual Memory), whose abstractions do not involve considerations of locomotion. Alternatively, the master may have given propositions in claim sentences (see IV.1.5), or they may have resulted from the robot's own actions.
(3) The *Propositional Memory* is a set of properties of concepts:

\[
\begin{align*}
\ldots \\
&\langle\text{concept-i} \rangle \langle\text{property-ij} \rangle \langle\text{value-ij} \rangle \\
\ldots
\end{align*}
\]

e.g., \(\text{TABLE LOCATION R101}\) is a proposition abstracted from the example of (2).

Abstraction procedures do not undergo development; they are initially given and remain fixed. The only procedures implemented so far are the ones for abstracting the property LOCATION.

(4) *Abstracting of LOCATION through perceiving* is the procedure:

IF the last element in the Perceptual Memory is (SEE \text{<object>})

THEN add the property (LOC <room>) to the property list of \text{<object>}. \text{<room>} is the room of the last trace of the form (BE-IN <room>).

e.g., The example of (2) results in the abstraction of the example of (3).
(5) Abstracting of LOCATION through acting is the procedure:

IF (GRASP <object>)

THEN remove the property (LOC <room>) from the property list of <object>. <room> is the room of the last trace of the form (BE-IN <room>)

IF (UNGRASP <object>)

THEN add the property (LOC <room>) into the property list of <object>. <room> is the room of the last trace of the form (BE-IN <room>)

IV.1.4 Orienting Maps

Orienting maps lay out spatial organizations of the surroundings. As we may have different ordinary maps, different orienting maps may exist for different purposes with different scales. A single orienting map suffices for the subworld of a house, call it the Orienting Map. In this Computational Model we assume that the master describes the Orienting Map using claim sentences. In the Conceptual Model, you may recall, orienting maps develop.

(6) The Orienting Map consists of ADJACENT and FLOOR properties of room names.

e.g., (R101 ADJ R102)

(R101 FLOOR FLOOR1)

...
IV.1.5 Requests, Questions, and Claims

The admissible set of natural language sentences, used in communication between the master and the robot, are requests for performing tasks, claims for stating facts, and questions. Sentences are in restricted English.

(7) Request ::= (! <imperative sentence>)
Claim ::= (: <declarative sentence>)
Question ::= (?) <interrogative sentence>)

e.g., (! GO INTO KITCHEN)
(: KITCHEN IS ADJACENT TO HALLWAY)
(?) WHERE IS JOHN)

IV.1.6 Skills

Skills are hierarchically built from lower level defining skills. Recalling memory and perceiving through perceptual experts provide preconditions for activating the motor organs in a skill. Perceptual experts also control termination of the active state of motor organs. Recursion may also be applied so that the skill itself can be used as a defining unit.

First we need some preliminary notions. Before any development, the robot has at its disposal a set of primitives of which all skills will be composed. The
primitives do not call for further analysis; they have been realized by best possible means, and nothing in skill definitions depends on the manner of realization.

(8) A primitive ::= 

<movement>, i.e., a call of a motor organ, 
<perception>, i.e., a call of a perceptual expert, or 
<recall>, i.e., a call of RECALL procedure (see IV.1.9) 

e.g., MOVE-FORWARD

(SEE TABLE)

(RECALL ?ROOM LOC OF TABLE)

Certain conditions of the environment initiate and terminate the robot's movements. In order to formalize admissible conditions, we define 'state'. Basically, a state is a boolean conjunctive normal form expression of calls of perceptual experts or RECALL, or it is the state which is always true. We say that the robot is in a state if the respective expression evaluates TRUE.
(9) An expression represents a *state* if it has been formed using rules 1) - 4) below.

1) If $P_i$ is <perception>, <recall>, or $T$ (the *empty perception* defined as $T(t) = \text{TRUE}$ for all $t$), then $(P_i)$ is a state.

2) If $S_i$ is a state, then $(\text{NOT } S_i)$ is a state.

3) If $S_i$ and $S_j$ are states, then $(\text{OR } S_i S_j)$ is a state.

4) If $S_i$ and $S_j$ are states, then $(S_i S_j)$ is a state (implied AND).

e.g., $(T)$

$((\text{OR } (\text{BE-IN } R_{101}) (\text{BE-IN } R_{102})))$

$((\text{RECALL } ?\text{ROOM ADJ } R_{101}) (\text{BE-IN } $ROOM$))$

Next, as a step to comprehensive skill definitions, we define 'primitive skill' as a movement coupled with termination conditions. A primitive skill reflects semi-autonomy in the sense that after external activation it contains means for terminating itself. We assume that any movement can be performed either in discrete steps or in a continuous manner. From now on we describe both syntactic rules and their semantic interpretations.
A primitive skill **:=**

SYN1: (movement UNTIL state), or
SYN2: (movement UNTIL (n STEPS))

Semantic interpretations:

SEM1: Activate movement;
    deactivate it when state evaluates TRUE.
SEM2: Perform movement for n steps.

e.g., (TURN-LEFT UNTIL ((SEE TABLE)))
    (MOVE-FORWARD UNTIL (1 STEPS))

We define the notion 'empty skill' as the means of explicitly saying when nothing needs to be done.

The empty skill **:=**

SYN3: (DONE)

SEM3: Return control immediately.

In order to be able to choose between different tactics for a given task, we need a more refined measure for the robot's being in a given state than what the binary truth value would provide.
The **closeness measure** of the robot with respect to a state expression is the number of consecutive (implicit) AND-terms, counted from the beginning of the expression, that evaluate TRUE.

e.g., Let us assume that the robot is in room R101 with a wall directly in front of it. 

```
((BE-IN R101) (SEE TABLE)) has the closeness measure 1.
```

```
(T) has closeness measure 1.
```

We are now ready to give rules for defining skills.

(13) A list represents a **skill** (definition) if it has been formed using the rules SYN4 ... SYN6 below. `<skill>` refers to a primitive skill, the empty skill, or a proper skill (one formed or being formed using these rules).

```
SYN4* ( <state>  >  (<skill1>  <skill2> ....))
```

```
SYN5* (EXCLUSIVE  (<state1>  >  (<skill11>  <skill12> ....)))
        ( <state2>  >  (<skill21>  <skill22> ....)))
```

```
SYN6* (REPEAT-UNTIL  <state> ( <skill1>  <skill2> ....))
```

SEM4: IF <state> evaluates TRUE

THEN perform <skill1> <skill2> ... in that order

IF <state> evaluates FALSE

THEN perform first the perceptual expectation of the first perception in <state> that made it FALSE (see IV.1.7 for definitions of perceptual expectations)

SEM5: IF at least one of the states evaluates TRUE

THEN choose the topmost of the competing skills whose state is TRUE and use the rule SEM4 for that skill

IF none of the states evaluates TRUE

THEN choose the topmost of the competing skills whose closeness measure is the greatest and use the rule SEM4 to that skill

SEM6: IF <state> evaluates TRUE

THEN return control immediately

IF <state> evaluates FALSE

THEN circularly perform the defining skills; discontinue execution when <state> is TRUE; test <state> before executing each of the defining skills
Example 1

; the surface form of a request
(GO INTO ELEVATOR) ***=

; the skill definition
(((BE-IN-ADJ-TO ELEVATOR))
 (GO TO ELEVATOR-DOOR)
 (CALL ELEVATOR)
 (TURN-RIGHT UNTIL (SEE ELEVATOR-DOOR))
 (WAIT UNTIL (SEE ELEVATOR-DOOR-OPEN))
 (ENTER IN ELEVATOR))

If the robot is not located in a room adjacent to the elevator, (BE-IN-ADJ-TO, ELEVATOR) is FALSE and the perceptual expectation for that perception will be executed first. For example, the following skill may represent that expectation. It simply asks the robot to go to a room adjacent to the elevator.

(((RECALL ?ROOM ADJ ELEVATOR))
 (GO INTO $ROOM)
 (RELEASE $ROOM))

Example 2

; the surface form of a request
(GO INTO R202) ***=

; the skill definition
(EXCLUSIVE (((BE-IN R202))
 (DONE))
 (((NOT (BE-IN R202))
  (BE-ON-FLOOR-OF R202)
  (BE-IN-ADJ-TO R202))
 (GO TO R202-DOOR)
 (OPEN R202-DOOR)
 (ENTER IN R202)))
If the robot already is in room R202, nothing needs to be
done. If it is not, the perceptual experts (BE-ON-FLOOR-OF
R202) and (BE-IN-ADJ-TO R202) will be called in that order.
If either returns the value FALSE, the respective perceptual
expectation will be performed first. If we assume that the
robot was located in room R101 when the request occurred,
then (BE-ON-FLOOR-OF R202) will return FALSE and the
following perceptual expectation will be executed first.
The expectation simply asks the robot to go to the floor in
question.

((RECALL ?FLOOR FLOOR-OF R202))
(GO TO $FLOOR)
(RELEASE $FLOOR))

**Example 3**

; the surface form of a request
(SEARCH FOR CHAIR) :=

; the skill definition
(REPEAT-UNTIL (OR (SEE CHAIR)
   (RECALL $ROOM))
   (REMEMBER-OTHER-ROOMS)
   (GO INTO $ROOM)
   (LOOK FOR CHAIR))

The robot will execute

(REMEMBER-OTHER-ROOMS)
(GO INTO $ROOM)
(LOOK FOR CHAIR)
in that order until it either sees a table or the reference-variable (see IV.1.8) $ROOM$ points to an empty list signalling that all rooms have been searched.

Notice how the use of perceptual expectations, as defined in SEM4, loads the system with the capability of dynamically creating subgoals at execution time.

Skills obey the general schema:

\[ \text{<situation>} \implies \text{<action>} \]

They resemble production systems [Davis and King] with the provision that percepts correspond to data and movements to actions on data.

IV.1.7 Perceiving

The Conceptual Model, you may recall, defined perceiving as activating a perceptual schema consisting of a perceptual expert and a perceptual expectation. When the truth value of a particular perception is called, the respective perceptual expert is activated. It returns TRUE or FALSE depending upon whether the anticipated state of affairs in the environment is true or not. A successful perceiving causes the trace of the percept — the call to
the perceptual expert procedure — to be stored in the Perceptual Memory. A perceptual expectation is simply a skill that may be activated (for conditions for activation, see (13)) in the case the expert responds FALSE. The expectation performs such bodily movements that after the execution, the probability for perceiving in a satisfactory manner has increased.

(14) A **perceptual expert** is a function procedure $P_i$ such that:

\[
P_i(t) = \begin{cases} 
\text{TRUE} & \text{iff state-of-affairs-i holds at the moment t} \\
\text{FALSE} & \text{otherwise} 
\end{cases}
\]

\[\text{e.g., (SEE TABLE) represents a call to a perceptual expert to visually recognize tables.}\]

(15) A **perceptual expectation** is a skill $S_i$ such that the probability of $P_i$ being TRUE increases if $S_i$ is executed provided that $P_i$ evaluated FALSE before activation of $S_i$.

\[\text{e.g., (T)} \\
\text{(TURN-LEFT UNTIL (SEE TABLE)))} \]

is a perceptual expectation for (SEE TABLE) provided that a table exists in the room.
The *Perceptual Memory acquisition* is the procedure:

IF Pi(t)=TRUE (Pi is a perceptual expert) 

THEN append the trace of Pi (its call) to the Perceptual Memory.

With respect to perceptual experts, the Computational Model of this section drastically reduces the scope of the Conceptual Model. While perceptual experts of the agent were subject to development, no such development occurs in the robot. All experts are fixed function procedures.

IV.1.8 Recalling

RECALL represents the nondevelopmental function procedure possessing the sole authority to retrieve the Memory. Any reference to the Memory must be established using RECALL. A call of RECALL has the general schema:

(RECALL <reference-variable> <intension>)

An *intension* describes the individuals in LTOC who will be bound as a set to the reference-variable when the control returns from RECALL. A value of a reference-variable will hence be a list of individuals, called the *extension* of an intension. If a returned extension is a null list, function RECALL has the value FALSE; otherwise it evaluates TRUE.
Such recalling is, undoubtedly, rather restricted since only names of individuals can be referred to. No reference can be made to memory structures. A reference-variable is prefixed with '?' in binding and with '$' in referring. Hence a schema of a call of RECALL also has the form:

(RECALL ?<ref-var> <intension>)

A gross interpretation of RECALLing is the following: Find the names of those individuals in LTOC that are siblings or descendants of siblings of $<ref-var>$ and that satisfy <intension>.

Two concept names are siblings if they have the same father in LTOC.

(17) The function procedure RECALL obeys the following syntactic and semantic conventions (brackets denote optionality):

SYN:

(RECALL ?<ref-var> ([intension] [EXCEPT <intension2>]]))
1. <intension> ::= <concept>
2. <intension> ::= $<ref-var2>
3. <intension> ::= <property> <value>
4. <intension> ::= <property> OF <concept>
5. <intension> ::= <trace of percept> [<keyword>]
   <value> ::= value of a property, or
   $<ref-var2>
   <keyword> ::= FIRST or LAST
SEM: $\text{RECALL} = \text{TRUE}$ if the extension (the set of names of siblings or descendants of siblings of $\text{<ref-var>}$ satisfying $\text{<intension1>}$ but excluding those satisfying $\text{<intension2>}$) is not empty.

$\text{RECALL} = \text{FALSE}$ if the extension is empty.

The extension is bound as the value to the reference-variable $\text{<ref-var>}$.

The extensions for each type of intensions 1.-5. above are defined separately below. Assume that in LTOC $\text{ROOM}$ and all room names are sons of GO-INTO-ABLE (see Figure II.8).

0) If no intension is given, all descendants of siblings of $\text{<ref-var>}$ are bound to the reference-variable.

*e.g.*, $(\text{RECALL} \ ?\text{ROOM})$ binds all room names to $\text{ROOM}$.

1) The individual $\text{<concept>}$ is the extension.

*e.g.*, $(\text{RECALL} \ ?\text{ROOM EXCEPT R101})$ will bind all room names except R101.

2) The extension is the intersection of descendants of siblings of $\text{<ref-var>}$ and the value of $\text{<ref-var>}$ (extension of an earlier established $\text{RECALL}$).

*e.g.*, $(\text{RECALL} \ ?\text{ROOM} \ \text{ROOM})$ will return all room names of the value of $\text{ROOM}$.

*e.g.*, $(\text{RECALL} \ ?\text{ROOM} \ \text{ROOM EXCEPT R101})$ will return all room names $\text{ROOM}$ refers to, excluding R101.
3) The extension is the set of descendants of siblings of $\text{<ref-var>}$ having $\text{<property>}$ with $\text{<value>}$, or alternatively if $\text{<value>}$ is a reference-variable, having $\text{<property>}$ with a value that is a member of the value of $\text{<ref-var2>}$. This intension can refer to the Propositional Memory or to orienting maps.

e.g., (RECALL ?ROOM ADJ R101) returns all names of the rooms adjacent to the room R101.

e.g., (RECALL ?ROOM ADJ $\text{ROOM}$) returns all names of the rooms adjacent to the rooms $\text{ROOM}$ refers to.

e.g., (RECALL ?ROOM ADJ $\text{ROOM}$ EXCEPT ADJ R101) returns all names of the rooms adjacent to the rooms $\text{ROOM}$ refers to, excluding the ones adjacent to the room R101.

4) The extension is the set of descendants of siblings of $\text{<ref-var>}$ that are properties of $\text{<concept>}$. This intension refers to the Propositional Memory or to the orienting maps.

e.g., (RECALL ?ROOM LOC OF TABLE) returns all names of the rooms in the property list of the concept TABLE.

5) The extension is the set of descendants of siblings of $\text{<ref-var>}$ in a trace of the form $\text{<trace of percept>}$ in the Perceptual Memory. Optionally, the last one of such concepts if LAST is present, or the first one if FIRST is present. This recalling refers exclusively to the Perceptual Memory.

e.g., (RECALL ?ROOM BE-IN FIRST) returns the room name occurring in the first trace of a percept of the type (BE-IN <room>).

e.g., (RECALL ?ROOM EXCEPT BE-IN FIRST) returns all room names except the one which exists in the first trace of the type (BE-IN <room>).

e.g., (RECALL ?TABLE SEE) returns table names that are in percepts of the type (SEE <table>) in the Perceptual Memory. If tables are not further individualized, the only effect of this recalling is the signal that a table has been seen.

e.g., (RECALL ?TABLE SEE LAST) returns the table name seen last.
Computationally, a reference-variable represents a push-down stack: a recalling causes the stack to be pushed, and a reference points to the top of the stack. In a skill a value of a reference-variable must be explicitly released through the use of RELEASE pseudo skill. The following example illustrates a recalling, referencing, and releasing of the value of a reference-variable.

(((RECALL ?ROOM LOC OF BASKET))
(GO INTO $ROOM)
(PICK UP BASKET)
(RELEASE $ROOM))

IV.1.9 Monitoring

MONITOR is the nondevelopmental procedure responsible for monitoring execution of requests. It parses them, retrieves appropriate skill definitions, and monitors their execution. A request is parsed into a deep case structure that is used, employing pattern matching, to retrieve the respective skill definition.
A deep case structure of a request :=

\[
(\langle \text{verbdef} \rangle \ \langle \text{casedef1} \rangle \ \langle \text{casedef2} \rangle \ \ldots)
\]

\[
\text{verbdef} := (\text{VERB} \ \langle \text{verb} \rangle)
\]

\[
\text{casedef} := (\langle \text{case} \rangle \ \langle \text{PREP} \ \langle \text{prep} \rangle \rangle \ \langle \text{NOUN} \ \langle \text{noun} \rangle \rangle)
\]

\[
\text{case} := \text{OBJECT, AGENT, INSTR, LOCAT, DESTIN, TRAJECT, or TIME}
\]

\[
\text{prep} := \text{a preposition or other qualifier}
\]

PARSER is the function procedure:

\[
\text{PARSER(}\langle \text{request-i} \rangle) = \langle \text{deep case structure-i} \rangle
\]

\[
e.g., \ \text{PARSER}((! \ \text{GO \ INTO \ R101})) =
\]

\[
((\text{VERB} \ \text{GO}) \ \langle \text{DESTIN} \ \langle \text{PREP} \ \text{INTO} \rangle \rangle \ \langle \text{NOUN} \ \text{R101} \rangle))
\]

A deep structure schema (of a request) represents a generalized deep structure such that <verb> may be a list of verbs, <prep> a list of qualifiers or prepositions, and <noun> any concept in LTOC. We say that a schema applies to a deep case structure of a request if the verb and the prepositions of the latter are identical to or members of the respective verbs and prepositions of the schema, and the concepts of the latter are descendants or identical to the respective concepts in the schema. If a schema applies to a deep case structure, the deep case structure becomes an instance of the schema.
is a deep structure schema. It applies to the deep case structure given in the example of (19) assuming that LTOC is the one in Figure IV.6.

So far 'skill' has referred to skill definitions. Every defined skill must be stored in a way that allows an easy access to it.

(20) A stored skill is an ordered pair:

\[(\text{<deep structure schema> } \text{<skill definition>})\]

\[e.g., \left((\text{VERB TAKE}) (\text{OBJECT} (\text{NOUN TAKE-ABLE})))\]

\[\text{(T)}\]

\[\text{(LOOK FOR TAKE-ABLE)}\]

\[\text{(PICK UP TAKE-ABLE))}\]

(21) A collection of skills, called SKILLS, is the list:

\[\text{(<stored skill1> <stored skill2> ...)}\]

A stored skill is general since it may apply to many requests. Naturally, then, prior to executing a request, the instance of a particular task must be fabricated through the schema into the skill definition. We say that a stored skill is bound to a request when those concepts in the schema which are generalizations of the respective instances
in the request are replaced by their instances in the skill definition.

e.g., (! TAKE CHAIR) applies to the skill in the example of (20) assuming that LTOC is the one in Figure IV.6. When the skill is bound, the skill definition becomes:

```
(((T)
 (LOOK FOR CHAIR)
 (PICK UP CHAIR))
```

We said earlier that MONITOR is composed of parsing requests, retrieving skill definitions, and monitoring their performance. So far we have defined parsing; now we define retrieval and execution of skills.

(22) RETRIEVER is the following recursive function procedure:

Fetch the first skill of SKILLS.

IF the end of SKILLS
 |
 THEN return FAILURE

IF the deep structure schema of the stored skill applies to the deep structure of the request
 |
 THEN return the bound skill definition

IF the deep structure schema of the stored skill does not apply
 |
 THEN apply RETRIEVER to the rest of SKILLS
We are now ready to define MONITOR as a composite procedure of PARSER and RETRIEVER.

(23) MONITOR is the procedure:

IF the sentence the master gives is NIL signaling the end of a discourse
THEN perform the task as defined in TASKLIST by executing its skill definitions sequentially using the semantic interpretation rules SEM1...SEM6 ((10), (11), and (13))

IF the sentence begins with a question mark (?) or a colon (:) 
THEN call FACT-MANIPULATION procedure; ask the next sentence

IF the sentence begins with an exclamation mark (!)
THEN first call PARSER with the request to obtain the deep case structure; then call RETRIEVER with the deep case structure

IF RETRIEVER returns a bound skill definition
THEN append the skill definition in TASKLIST; ask the next sentence

IF RETRIEVER returns FAILURE
THEN call SKILL-MODIFICATION (see IV.2.1); if it returns a skill definition, append it in TASKLIST; ask the next sentence

IF both RETRIEVER and SKILL-MODIFICATION return FAILURE
THEN call SKILL-ACQUISITION (see IV.2.2); append the returned skill definition in TASKLIST; ask the next sentence
By fact manipulation we mean procedures for performing the following activities:

1) Parsing claims
2) Updating the Memory according to claims
3) Parsing questions
4) Retrieving data items for answering questions
5) Generating answers for questions based on retrieved data items

Since the behavior of the robot in physical tasks rather than question answering has been considered of primary importance, the discussion on fact manipulation is exceedingly cursory. We have proposals for 1) and 2), but even those are tentative. We have crossed where the fence is the lowest. For 3), 4), and 5) we have only an informal, sketchy discussion to offer.

Claims are used for the purpose of bypassing the robot's own perceptual experience in updating the Memory. In the surface form, claims are indicated by a leading colon. We use fixed formats for claims thus substituting parsing with pattern matching.
(24) The admissible claim patterns are the following:

A. (* :property: OF <noun> IS <value>)
B. (* :property: OF <noun> IS NOT <value>)
C. (* SKILL OF <request> IS <skill definition>)

e.g., (* LOCATION OF TABLE IS R101)

The procedure for updating the Memory or SKILLS follows.

(25) CLAIM-UPDATING is the procedure:

IF type A claim
AND the concept <noun> does not have <property>
THEN add <property> with <value> to its property list

IF type A claim
AND the concept <noun> has <property>
THEN add <value> to the property list

IF type B claim
THEN remove <value> from the property list of <property> of the concept <noun>; if the value list becomes empty remove <property>

IF type C claim
AND a skill for <request> exists in SKILLS
THEN replace the stored skill definition by <skill definition>
IF type C claim
AND a skill for <request> does not exist in SKILLS
THEN add a new stored skill with the deep structure of
<request>

We do not even have a tentative formal proposal for
parsing questions and generating answers. A sound account
of these issues would require careful protocol and
linguistic analyses, enterprises which are outside the scope
of this work. We do, however, conduct a partial, informal
discussion on the topic.

What does a robot, built around the principles
propounded in this paper, know? It possesses several
functionally separable faculties, designed primarily as
tools for generating proper behavior for performing tasks.
Interestingly, as a side effect, these faculties contain
'knowledge' about the world, although partially in a diffuse
form. Notice first that the Perceptual Memory, the
Propositional Memory, orienting maps, and the Logical
Taxonomy of Concepts all contain information about the world
in an explicit form. This information could also be used
for reflecting about the world and answering questions
accordingly. Then notice that skill definitions, too,
implicitly contain 'knowledge' about the world. For
example, a skill for picking up a chair implies that chairs
are individuals that can be picked up. A skill for sitting
on a chair implies that chairs are individuals that can be
sat on, and so on. All skills that can be focused on chairs
tell something about the nature of chairs; they contain 'knowledge' about them in a diffuse form.

No attempt is made to cover the full range of different questions that might be asked. The two classes of inquiries we do comment upon are: 'knowing how' questions that focus on the ways actions are performed, and 'knowing what' questions that focus on static aspects of the surroundings.

Clearly, the knowledge source for creating answers for 'knowing how' questions is SKILLS. Without going into details, we envision that questions of this type would be parsed into a deep case structure from which the deep case structure of the respective requests could be extracted. This request, then, would be used in retrieving the respective skill definition (13). The answer-generating routine would scrutinize the skill definition, its form, preconditions, and defining skills, and then respond accordingly. If a more detailed answer is called for, the definitions of the defining skills would be retrieved and scrutinized in the same manner. The process would continue until either the desired level of detail is reached, or primitive skill definitions (10), which cannot be analyzed further, are encountered.

Let us use the example of (20) as an illustration of the outlined approach. If the master should ask the question:

( ? HOW DO YOU TAKE A CHAIR)
The following answer could be easily generated by a general answer generating procedure:

"Regardless of the situation, I always first look for a chair, and then I pick it up."

The robot is then ready to explain in more details what 'looking for a chair' or 'picking a chair up' mean by retrieving their skill definitions and repeating the procedure.

'Knowing what' questions call for a different approach. Some of these inquiries can be answered in an obvious manner. For example, questions that refer to the robot's perceptual experience can be answered by consulting the content of the Perceptual Memory. Hence the question:

( ? HAVE YOU SEEN JOHN )

evokes searching the trace of a percept of the form (SEE JOHN) in the Perceptual Memory.

For questions that focus on the contents of the Propositional Memory or orienting maps, a similar simple approach applies. For example, the question:

( ? WHERE IS JOHN )

transforms into looking for a value for the property LOC of the concept JOHN in the Propositional Memory.
"Knowing what" questions that inquire the robot's 'understanding' of concepts per se are more interesting. The only knowledge sources that carry 'knowledge' about concepts are LTOC and SKILLS. An example might provide insight into how these sources might be utilized. For example, what could the robot possibly answer to the question:

(? WHAT IS A TABLE)

That is, what connotations does the concept TABLE have to the robot? For the discussion that follows, assume that LTOC is the one in Figure II.8. The robot can consult LTOC and existing skills and find ways TABLE participates in actions. The robot can run through the stored skills, first looking for skills that contain the least general form of the pending concept, TABLE in this case, and respond accordingly. Then it could generalize the concept by using the father concept of the concept, PICK-UP-ABLE and LEAVE-ABLE in this case, and repeat the process. In this example, assuming the existence of a single skill containing TABLE in its definition, say the one for the request:

(! PUT PICK-UP-ABLE ON TABLE)

the robot would answer:

"Table is a concept that I can put pick-up-able things on."
It would then generalize TABLE and tell that it can pick a table up, leave it by objects, put it on a table, and so forth. The next generalization step would yield a new set of skills and answers. In each instance the robot is ready to explain what each sentence 'means': it can retrieve the respective skill, scrutinize the form and the defining skills, and respond accordingly. Such answers reflect the robot's functional 'understanding' of the concepts.
IV.2 DEVELOPMENTAL ABSTRACTION

The Developmental Abstraction (DA) explains how the constituents of the Behavioral Abstraction (BA) have come into existence or will do so. While BA offers an explanation of the robot's competence at any moment, DA offers an explanation for the progression of competence.

IV.2.1 Skill Modification

Skill modification in this context means a set of procedures aimed at either generalizing scopes of existing skills to cover new task domains or modifying existing skills that have manifested handicaps in some task instances. As an example of the first feature, think of a skill for carrying an ashtray. Skill modification may come to a successful conclusion that the very same skill also can be used for carrying a book or, more generally, any movable thing. As to the second skill modification feature, assume that a skill exists for carrying a movable thing. Some of objects may be so heavy that their moving requires a van. Skill modification should now be able to modify the existing skill to anticipate this requirement, if the skill in its original form lacked the feature. This Computational Model does not attempt to achieve a solution to the last mentioned types of modification instances; they are clearly more
complex features and are left for future research (see Section V). For the former feature we offer a solution which relies on the master's advice.

Skill modification is performed by the procedure SKILL-MODIFICATION, which is activated if RETRIEVER does not find an applicable skill for a request. SKILL-MODIFICATION has three kinds of heuristics at its disposal in attempting a generalization:

1) It attempts to find synonymous verbs.

2) It attempts to find synonymous prepositions.

3) It attempts to generalize concepts in skill definitions.

SKILL-MODIFICATION also plays another developmental role in developing LTOC by progressively associating new concepts. If SKILL-MODIFICATION successfully concludes that some siblings (sons of the same father) of a concept name in a skill are applicable to the skill, then obviously siblings can be associated under a higher level concept. SKILL-MODIFICATION must guarantee, however, that the skill would then apply to all remaining siblings. If that is not the case, a new concept must be generated to reflect the proper level of generality in the view of the current skill. We will give below a few examples of different skill modification instances. Created concept names reflect the skills that have induced them. A concept name GO-INTO-ABLE,
say, results from generalizing the request (! GO INTO ---).

(26) **SKILL-MODIFICATION** is the following procedure that is activated after RETRIEVER has reported FAILURE. We say that a concept is a **constant** if it has no members in LTOC. Concepts that have members are called **variables**. (All variables, except SOMETHING, are generated concept names.)

Fetch the first skill of SKILLS.

IF no skill is received

\[\text{THEN return FAILURE}\]

\[\text{; The next two rules only shape deep structure schemas of stored skills.}\]

IF the deep structure schema of a skill differs only in verbs, if it would otherwise apply (see Example 1 below)

\[\text{; THEN prompt the master for confirmation of applicability of the skill to the pending request; if the answer is affirmative, update the stored skill by adding the new verb in the list of verbs and return the skill definition if the answer is negative, apply SKILL-MODIFICATION to the rest of SKILLS}\]

IF the schema of the fetched skill differs only in that one of the prepositions is different, the schema would apply otherwise (see Example 2 below)

\[\text{; THEN ask for confirmation and either update the stored skill by adding the preposition in the list of prepositions in the schema and return the skill definition, or apply SKILL-MODIFICATION to the rest of SKILLS}\]
The rest of the rules shape skill definitions and LTOC. They do not shape deep structure schemas.

The conditions below are invoked if the schema of the skill would apply otherwise, but there is a conflict in a concept.

IF the concept in the request is new (call it ReqConst), AND the concept in the schema is a constant (call it SchConst), AND the master confirms that the skill applies:

SKILL-MODIFICATION generates a new variable of the form 'verb-prep-ABLE' (call it NewVar) and locates it in LTOC. SKILL-MODIFICATION checks further that possible ALIAS concepts of NewVar are also genuine fathers of ReqConst.

THEN set both ReqConst and SchConst as sons of the NewVar; ask the master to confirm the applicability of the skill to the father of SchConst (excluding SOMETHING);

IF the skill applies to all sons of the father (see Example 3)

THEN assign NewVar to the same location in LTOC with the father of SchConst; check that all skills that contain ALIAS concepts (occupy the same location in LTOC) do apply to ReqConst; if such a skill is found that does not apply, the respective ALIAS concept is made less general; return the generalized skill definition

IF the skill does not apply to all sons of the father (see Example 4)

THEN set the applicable sons to be sons of NewVar, set NewVar to be a new son of the father, and remove the applicable sons from the son list of the father; return the generalized skill definition
IF the concept in the request is **new** (call it ReqConst);
AND the concept in the schema is a **variable with constant members** (call it SchVar);
AND the master confirms that the skill applies (see Example 5);

; No new variable needs to be created, ReqConst will be
; a new member of SchVar.
; But again, it needs to be checked that possible ALIAS
; concepts are genuine fathers of ReqConst.

THEN set ReqConst to be a new son to SchVar;
check that ReqConst applies to all skills that contain ALIAS
concepts of SchVar;
if such a skill does not apply to ReqConst, decrease
generality of the respective ALIAS concept by setting it and
ReqConst to be sons of SchVar (and its ALIAS concepts);
return the generalized skill

---

IF the concept in the request is **new** (call it ReqConst);
AND the concept in the schema is a **variable with at least one variable members** (call it SchVar);
AND the master confirms that the skill applies (see Example 6);

; No new variable needs to be generated.
; ReqConst will be a descendant of SchVar.
; It remains to locate ReqConst in the least general
; variable that is a descendant of SchVar.

THEN find the **least** general variable of the sons such that
the new concept would be a son of it by checking
applicability of skills containing those variables, set
ReqConst to be a son of the found variable;
if no such a variable is found, set ReqConst to be a son of
SchVar;
return the generalized skill definition
IF the concept in the request already exists (call it ReqConst);
AND the master confirms that the skill applies

; The concept in the schema (SchConst) is generalized until
; either it 'covers' both ReqConst and old SchConc
; or a new variable must be generated.

THEN generalize the skill by repeatedly replacing SchConc by
its father, then the father by its father, and so on, until
the master rejects applicability of the skill; then check
applicability of the skill to the sons of the rejected
concept (call it RejConc)

IF the skill applies to a single son (see Example 7)
; It must apply to at least one son.
THEN replace SchConc by that son;
return the generalized skill

IF the skill applies to more than one son (see Example 8)
; It cannot apply to all.
; A new variable is created to gather the applicable
; sons.
THEN create a new variable (call it NewVar) and set the
applicable sons to be sons of NewVar;
set NewVar to be a son of RejConc replacing the
applicable sons;
return the generalized skill

IF none of the rules above applies
THEN call SKILL-MODIFICATION with the rest of SKILLS

E.g., Some examples of skill modification are given below.
We assume that LTOC has already developed to the stage shown
in Figure IV.1 and that the shown skills already exist in
the shown forms.
LTOC1: (SOMETHING (SONS DOOR PICK-UP-ABLE ENTER-IN-ABLE))
LTOC2: (PICK-UP-ABLE (SONS CHAIR TABLE))
LTOC3: (LEAVE-ABLE (ALIAS PICK-UP-ABLE))
LTOC4: (ENTER-IN-ABLE (SONS R101 ELEVATOR))

SKILL1: (((VERB ENTER) (DESTIN (PREP IN)) ; schema
  (NOUN ENTER-IN-ABLE)))
  < ... ENTER-IN-ABLE ...>) ; skill def.

SKILL2: (((VERB PICK) (OBJECT (PREP UP)
  (NOUN PICK-UP-ABLE))))
  < ... PICK UP PICK-UP-ABLE ...>)

SKILL3: (((VERB GO) (DESTIN (PREP INTO) (NOUN R101)))))
  < ... R101 ...>)

SKILL4: (((VERB TAKE) (OBJECT (NOUN CHAIR))))
  < ... CHAIR ...>)

SKILL5: (((VERB LEAVE) (OBJECT (NOUN LEAVE-ABLE))
  ; schema
  (DESTIN (PREP IN) (NOUN ELEVATOR)))
  < ... LEAVE-ABLE ... ELEVATOR ...>)

SKILL6: (((VERB LOOK) (OBJECT (PREP FOR) (NOUN DOOR)))
  < ... DOOR ...>)
Example 1

; request
(! MOVE INTO R101)

; its deep case structure
(((VERB MOVE) (DESTIN (PREP INTO) (NOUN R101))))

The master confirms that skill modification applies to
SKILL3. (The master regards 'Go into R101' and 'Move into
R101' semantically equivalent.) Its deep structure schema is
modified into:

SKILL3': (((VERB (GO MOVE)) (DESTIN (PREP INTO)
(NOUN R101)))

< ... R101 ... >)

No modification occurs in LTOC.

Example 2

; request
(! GO IN R101)

; its deep case structure
(((VERB GO) (DESTIN (PREP IN) (NOUN R101))))
Skill modification applies to SKILL3'. (The master regards 'Go into R101' and 'Go in R101' semantically equivalent.) Its deep structure schema is modified into:

SKILL3': (((VERB (GO MOVE)) (DESTIN (PREP (IN INTO)) (NOUN R101)))

< ... R101 ... >)

No modification occurs in LTOC.

Example 3

; request
(! TAKE BASKET)

; its deep case structure
((VERB TAKE) (OBJECT (NOUN BASKET)))

Skill modification applies to SKILL4. The new variable TAKE-ABLE is generated. SKILL4 is generalized into:

SKILL4': (((VERB TAKE) (OBJECT (NOUN TAKE-ABLE)))

< ... TAKE-ABLE ... >)

At least chairs and baskets are TAKE-ABLE. Since also tables can be taken (the master confirms that), TAKE-ABLE will occupy the same location in LTOC as PICK-UP-ABLE and LEAVE-ABLE do. The master further confirms that a basket
also is a PICK-UP-ABLE and LEAVE-ABLE (the respective skills apply to BASKET), hence PICK-UP-ABLE and LEAVE-ABLE remain in their locations in LTOC. Figure IV.2 shows LTOC after Example 3.

Example 4

; request
( ! GO INTO R102)

; its deep case structure
((VERB GO) (DESTIN (PREP INTO) (NOUN R102)))

Skill modification applies to SKILL3'. A new variable GO-INTO-ABLE is created and SKILL3' is generalized into:

SKILL3': (((VERB (GO MOVE)) (DESTIN (PREP (IN INTO)))
            (NOUN GO-INTO-ABLE)))
< ... GO-INTO-ABLE ... >)

Since that skill does not apply to the elevator (you cannot use an elevator in order to get into the elevator), GO-INTO-ABLE is less general concept than ENTER-IN-ABLE (see Figure IV.3).
Example 5

; request
(! PICK UP BOOK)

; its deep case structure
((VERB PICK) (OBJECT (PREP UP) (NOUN BOOK)))

Skill modification applies to SKILL2. It is not modified. BOOK is set to be a new son of PICK-UP-ABLE. The master further confirms that a book also is a LEAVE-ABLE and TAKE-ABLE (those skills apply to it), hence LEAVE-ABLE and TAKE-ABLE remain in their locations in LTOC (Figure IV.4).

Example 6

; request
(! ENTER IN R103)

; its deep case structure
((VERB ENTER) (DESTIN (PREP IN) (NOUN R103)))

Skill modification applies to SKILL1. It is not modified. Since SKILL3 also applies to R103, R103 becomes a son of GO-INTO-ABLE (Figure IV.5).
Example 7

; request
(! LEAVE CHAIR IN R101)

; its deep case structure
((VERB LEAVE) (OBJECT (NOUN CHAIR))
  (DESTIN (PREP IN) (NOUN R101)))

Skill modification applies to SKILL5. Since ELEVATOR (in SKILL5) and R101 have the common ancestor, ENTER-IN-ABLE, to which SKILL5 applies, the skill is generalized into:

SKILL5': (((VERB LEAVE) (OBJECT (NOUN PICK-UP-ABLE))
  (DESTIN (PREP IN)
   (NOUN ENTER-IN-ABLE)))
  < ... PICK-UP-ABLE ... ENTER-IN-ABLE ... >)

Since the skill does not apply to other sons of SOMETHING, than ENTER-IN-ABLE, no concept is created nor is LTOC otherwise modified.
Example 8

; request
(! LOOK FOR CHAIR)

; its deep case structure
(((VERB LOOK) (OBJECT (PREP FOR) (NOUN CHAIR))))

Skill modification applies to SKILL6. It is generalized. Since the skill applies to doors and TAKE-ABLE (and its aliases), but it does not apply to ENTER-IN-ABLE (the remaining son of SOMETHING), the new variable LOOK-FOR-ABLE is created.

SKILL6': (((VERB LOOK) (OBJECT (PREP FOR)
(NOUN LOOK-FOR-ABLE)))

< ... LOOK-FOR-ABLE ... >)

LOOK-FOR-ABLE gathers DOOR and TAKE-ABLE in the way shown in Figure IV.6.
Figure IV.1: LTOC Before Skill Modification Examples

Figure IV.2: LTOC After Example 3
Figure IV.3: LTOC After Example 4

Figure IV.4: LTOC After Example 5
Figure IV.5: LTOC After Example 6

Figure IV.6: LTOC After Example 3
IV.2.2 Skill Acquisition

Skill acquisition in this context refers to a nondevelopmental procedure SKILL-ACQUISITION which is activated if first RETRIEVER and then SKILL-MODIFICATION return FAILURE. This Computational Model does not propose any capability for the robot to create skills on its own — such features are left for future research — instead, SKILL-ACQUISITION simply asks advice from the master.

Also SKILL-ACQUISITION has another role to play than creating skills. It progressively differentiates concepts in LTOC. (Recall how SKILL-MODIFICATION performs in a way a complementary action by progressively associating concepts in LTOC.)

(27) SKILL-ACQUISITION is the following procedure that is activated after SKILL-MODIFICATION reports FAILURE.

Ask the master to define the new skill; if necessary, explain the syntax and display the existing skills; store the skill in SKILLS

IF the request contains a new concept (see Example 1 and Example 2)

THEN find the least general variable in LTOC such that the new concept would be a son of it by checking applicability of the skills that contain the variable;
if none is found, set the new concept to be a son of SOMETHING;
store the new skill;
return the skill definition

IF the request contains only existing concepts (see Example 3)

THEN return the skill definition
E.g., Some examples of skill acquisition are given below. We assume that LTOC has already developed to the stage shown in Figure IV.7 and that the skills below exist before the acquisition instances.

LTOC1: (SOMETHING (SONS PICK-UP-ABLE GO-INTO-ABLE))
LTOC2: (PICK-UP-ABLE (SONS TABLE CHAIR)).
LTOC3: (GO-INTO-ABLE (SONS R101 R102))

SKILL1: (((VERB PICK) (OBJECT (PREP UP) (NOUN PICK-UP-ABLE)))
< ... PICK-UP-ABLE ... >)

SKILL2: (((VERB GO) (DESTIN (PREP INTO)
(NOUN GO-INTO-ABLE))))
< ... GO-INTO-ABLE ... >)

Example 2

; request
(! ENTER IN ELEVATOR)

; its deep case structure
(VERB ENTER) (DESTIN (PREP IN) (NOUN ELEVATOR)))
SKILL3 is prompted from the master (below). Since neither (! PICK UP ELEVATOR) nor (! GO INTO ELEVATOR) apply in their defined forms, ELEVATOR is set to be a son of SOMETHING. Notice that the latter skill does not apply because in order to "Go into elevator", one does not have to use the elevator. If (! GO INTO ELEVATOR) is requested, a separate skill for that needs to be defined. Figure IV.8 shows LTOC after this example.

SKILL3: (((VERB ENTER) (DESTIN (PREP IN) (NOUN ELEVATOR)))
<br>
< ... ELEVATOR ... >)

Example 10

; request
(! TAKE BASKET)

; its deep case structure
((VERB TAKE) (OBJECT (NOUN BASKET)))

SKILL4 is prompted from the master (below). Since SKILL1 applies to baskets, BASKET represents a son of PICK-UP-ABLE (Figure IV.9).

SKILL4: (((VERB TAKE) (OBJECT (NOUN BASKET)))
<br>
< ... BASKET ... >)
Example 11

; request
(!_ PUT DOWN CHAIR)

; its deep case structure
((VERB PUT) (OBJECT (PREP DOWN) (NOUN CHAIR)))

SKILL5 is prompted from the master (below). Since CHAIR is an existing concept name, no modifications occur in LTOC.

SKILL5: (((VERB PUT) (OBJECT (PREP DOWN) (NOUN CHAIR))))
      < ... CHAIR ... >)
Figure IV.7: LTOC Before Skill Acquisition Examples

Figure IV.8: LTOC After Example 9
Figure IV.9: LTOC After Example 10
IV.2.3 Perceptual Development

The robot has no capabilities for developing its perceptual experts. All experts are initially provided by the master and they remain fixed. In this regard the robot is highly restricted in power compared to the utopian agent of the Conceptual Model.

IV.2.4 Orienting Development

Again, the robot possesses no capabilities to this effect. We assume that the master either gives the Orienting Map at the outset using claims, or that he defines a skill for orienting, and requests that skill be performed before any other tasks.
In this conclusion we discuss three topics. First, we return to the discussion initiated in the Introduction, namely, in what fundamental way is skilled behavior different from planned behavior. In the Introduction we took a few tentative steps in this direction; we are now ready to support our arguments with concrete examples from the simulation in Chapter II. Secondly, we give suggestions as to what kind of a task domain the proposed advice-taking skill system applies and how to apply it. Thirdly, we outline future research.

V. I DEMARCATION BETWEEN SKILLED AND PLANNED BEHAVIOR

The reader should keep in mind that this work presents two overlapping systems: a utopian developmental system which develops on its own as described in the Conceptual Model, and a concrete computational advice-taking system in which some utopian features of the former are replaced by advice-taking from a human master. We have simulated one possible implementation of the Computational Model. The distinction between the two systems is important, since in
generating his advice, a human master utilizes both his skills and conceptual world knowledge. If a system forms its skills on its own without the use of a conceptual world model, obviously such skills would be less powerful. Below we repeat our operational definitions contrasting the planned and skilled behavior we gave in the Introduction.

(1) A problem solver, manipulating a symbolic representation of world states and operators to transform the states (world model), produces an operator sequence (plan) as a behavioral solution to the task. A monitor then interprets and executes the plan and continuously checks its correctness.

(2) An earlier formed and stored task-specific skill is activated. In the skill, perception procedures, directly probing the states of the immediate surroundings, and recalling memory, probing recorded perceived experience, control movements. Skills do not employ inference on a symbolic world model.

The proposed dichotomy is not shared by researchers in general. For example, [Miller et al.] considers 'plan' as covering any form of internal readiness to act. To them even instincts and skills represent plans.
Based on the above dichotomy, we propose the following classification for the covert side of behaving:

1. Formation of skills
2. Execution of skills
3. Exception handling and error recovery
4. Formation of plans
5. Execution of plans

As we came to see, this work addresses categories (1) and (2). In the utopian Conceptual Model skill formation would be performed by the agent itself. In the Computational Model and in the simulated implementation, skills are formed using a master's advice. In the Introduction we contrasted our system with the paradigmatic robot system in Artificial Intelligence. Such a system treats the task of artificially creating behavior as a planning task. The planning approach is more powerful than a skilled approach in the sense that no theoretical limits exist in planning behavior, but there exist planned behavioral patterns that cannot be skilled without the use of a world model. For example, a behavioral pattern representing the depositing of money in a savings account requires planning to project the future event of withdrawing the money. Also, there are behavioral instances where both methods apply, but planning results in more optimal behavior.
in the sense that subtasks interact favorably as we will shortly see.

On the other hand, a skill system is attractive on practical grounds. A planning system embodies practical limitations which, although solvable in a restricted static world, may prevent a strictly model-based planning approach from succeeding in dynamic and large domains. These problems stem from the use of a symbolic world model, the very source of power in planning systems. We already analyzed some such problems in the Introduction: 1) explicit planning of even the most elementary movements leads into efficiency problems, 2) at execution time a monitor must continuously check consistency of a plan and, if it recognizes inconsistencies, call corrective actions, and 3) formation and maintenance of a world model for large and particularly for open-ended domains is a difficult problem. Since skills do not employ a symbolic representation but directly receive control information from the trustworthy world through the use of perceptions, they do not suffer from these problems. Particularly, skills are executed efficiently.

We hope that we have demonstrated in our simulation that the proposed skills are easily given by a master and that, indeed, such skills are efficient and far less vulnerable to the inconsistency problem than plans. Now we turn to the problems faced by the proposed skill system, but not by a planning system. Specifically, to what extent and
in what manner is the skilled system handicapped due to the lack of exception handling and error recovery capabilities, on one hand, and due to the lack of planning facility, on the other? We discuss these questions separately below.

V.1.1 Exception Handling and Error Recovery

Reflection on everyday human behavior reveals that behavior is frequently guided by skills without a need for conscious checking between believed states of the world and reality, and two important occasions trigger a problem solving apparatus: in forming plans and in correcting discrepancies in skilled behavior. This section discusses the latter issue; the next section is devoted to the former one.

'Exception' refers to a state of the world which occurs so infrequently that a formed skill does not anticipate it. For example, consider car driving as a skill. A noisy malfunction in the engine is an exception in the above sense. By 'error' we refer to a situation that has resulted from an erroneous execution of a skill. Both classes seem to be closely related and are studied together. Generally speaking, then, the issue is the recognition and correction of an inconsistency between the world as a skill anticipates it and the real world.
Exceptions and errors can be recognized only if a means of comparing reality with skills exists. Since skills themselves lack the ability for introspection, some sort of monitor is needed to recognize exceptions and errors. We discuss some specific exception handling problems in our simulated world and draw general conclusions regarding the proposed skills from them.

(A) Assume that Sampo is on its way into a room adjacent to the current one. It has gone to the separating door, pushed the button to open the door, and is waiting for the door to open. Assume further that the door does not open due to a malfunction. The subskill (ENTER IN ENTER-IN-ABLE) that is performed after the button has been pushed anticipates only two possible states of the world: the door is open and the doorway is clear from obstacles, and the door is open and the doorway is blocked. It follows that Sampo will attempt to remove the unopened door as an obstacle! This instance is a case of exception handling resulting from insufficient preconditions in a subskill. Assuming that Sampo recognizes the exception, the solution for this exception is readily at hand: add a precondition to test the (perception of) state of the door.

(B) The skill for entering the elevator assumes that whenever Sampo goes into the elevator, there is enough room for it. When the elevator door is open, Sampo moves forward until it is totally inside the elevator. What happens if the elevator is so full of people that Sampo does not fit? This instance is a case of exception handling resulting from either insufficient preconditions or goal-conditions in a subskill. If the fullness of the elevator can be perceived prior to entering it, the skill can be corrected by adding preconditions to it. If such a perceptual faculty does not exist (Sampo does not possess such), the subskill can be corrected by adding an alternative goal-condition for checking the state when Sampo neither can move forward nor is yet fully inside the elevator. Assuming again that Sampo recognizes the exception, the remedy for the exception would be to retreat and wait for the next elevator.

(C) Assume that Sampo is asked to look for an object by turning around, but the mechanism for revolving the vision is stuck. Unless this is recognized by Sampo, Sampo obviously will stay idle indefinitely.
To summarize exception handling and error recovery, we found three kinds of possible inconsistencies in Sampo's skills: i) those due to insufficiently discriminative preconditions in a skill, ii) those due to insufficiently discriminative goal-conditions in a skill, and iii) those due to malfunctions in Sampo's body. Obviously, malfunctions in the environment represent exceptions.

In considering extensions to the proposed system, we must make the distinction between a utopian system which is assumed to carry out exception handling, among others, on its own, and the advice-taking system. The former must both recognize an exception or an error and perform corrective actions. Corrective actions clearly belong to the realm of problem solving. In case of an exception, a proper corrective action would consist of first finding out whether a precondition or goal-condition is insufficient and then augmenting either. In discussing future research we briefly return to this topic. If the recognized event is an error, the obvious corrective action consists of calling the maintenance personnel. In the advice-taking form a master can, in case of an exception or an error, use his problem solving facilities for handling inconsistencies properly, provided that he knows when to do so. In both versions of the system, then, a monitor is deemed necessary to recognize exceptions and/or errors. The monitor of the implemented system lacks such capabilities; it only retrieves and activates skills.
We leave the design of an intelligent monitor for future research and only offer here a few inconclusive remarks. A theoretically trivial but practically cumbersome solution, which is implicitly applied in the simulated implementation, would be to let the master serve as a monitor. He would physically follow the system during an extensive teaching period, observe exceptions and errors, and define corrective actions. A time-out function alone would recognize many but not all of the exceptions and errors: if the duration of an execution of a motor organ reaches the chosen threshold, an inconsistency is reported. Time-out would recognize insufficient goal-conditions in a primitive skill (e.g., instance (B) above) and some errors (e.g., instance (C) above), but it would not recognize insufficient preconditions or insufficient goal-conditions in an iterative \texttt{REPEAT-UNTIL} skill.

V. 1. 2 Planning

Since skills directly recognize states of the world, it follows that only extremely local aspects of the world influence skilled behavior. Only aspects of the world that are in the perceiveble vicinity of a behaving agent (geographical locality) and hold at the present time (temporal locality) can effect skilled behavior. In contrast, since in a world model all aspects of the world
are represented in a uniform manner regardless of their geographic or temporal distances, no such restrictions exist in planning. We separately study below what consequences these two forms of confinement have in skilled behavior.

Geographic locality. Geographic locality of a skilled system represents both a weakness and a strength. Should Sampo in the simulation possess full knowledge of all objects in the house, for instance, then there would be no need for searching anything. But from geographic locality it also follows that an increase in the numerical complexity has no effect on Sampo's behavior. For example, a skilled robot designed as a household servant works equally smoothly regardless of the number of rooms or objects in the house.

Temporal locality. Without doubt, the lack of the ability to project future events is the most severe handicap in skilled behavior if compared with planned behavior. Since planning is based on a world model that can be manipulated at will, a problem solver can arrange optimal interactions between subtasks of a task. That is, at a moment t1 an action can be planned to be performed in a certain way rather than another to result in favorable consequences for another action performed at a later moment t2 for the same complex task. Obviously only practical considerations of the planning process itself prevent a plan from projecting overall the most favorable interactions.
To understand the ways temporal locality hampers skilled behavior, let us examine some examples where Sampo, although able to perform the tasks, does them awkwardly.

(D) Refer to Step 6 of Figure II.9. Sampo has removed an obstructing chair in order to grasp a table and carry it to room R204. The general strategy for removing an obstacle is simple: 'Grasp it and put it aside behind you.' In this particular example the obstructing chair will be located in the front of the door to room R201. Should Sampo have been requested to move the table into R201, it should remove the very same obstructing chair again! Although Sampo is capable of doing that, its overall behavior would be awkward. A human would have, of course, moved the chair so that it would not obstruct the doorway he is going to use next. That is, a human would have planned the performance of a subtask so that it would interact favorably with a future subtask. In Sampo's skilled behavior each subskill is autonomous in that each subskill has its preconditions and termination conditions which are always executed in the same manner regardless of subsequent subskills.

(E) In Task #2 in Figure II.10 Sampo was requested to find a basket. The strategy for finding was simple: 'If you do not find it in the current room, go to the closest one you have not checked yet and look for it there.' Had Sampo not found a basket on the floor it was initially, it would have gone to another floor to continue searching, but the distance between the last searched room and the elevator would have been a random value. A human who carefully plans his performance in the same task would search the rooms in such an order that the last room would be close to the elevator. The lesson seems clear: all 'going to the next room' subtasks would interact in planned behavior since they would have been organized in a predetermined sequence. And, again, in Sampo's skilled behavior subskills do not interact; choosing of the next room does not depend on future events.

Naturally, since a human master defines these skills, future events can be projected in his world model, so that favorable interactions occur at the expense of the generality of the skills. Such skills would resemble plans in being more or less ad hoc solutions. At stake is the
trade-off between generality of skills and their number. For example, in (D) a single subskill for generally removing an obstacle was sufficient, but occasionally its performance leads to unfavorable states of the world for subsequent subskills. The more we relax generality, the greater the number of skills is needed, and the more the skills begin to resemble plans in being ad hoc solutions.

As long as a task, no matter how complex otherwise, can be broken into subtasks that do not interact over consecutive subtasks, a general skill can be built for the task. Using autonomous subskills and imposing a hierarchy in the build-up, the task complexity barrier can be gradually broken. If in a task interactions inherently exist between events that are apart in the temporal dimension, the task requires planning and results in ad hoc skills. For the almost classical 'stack of blocks' problem (see Introduction), a hierarchical skill can be readily created because interactions only occur between consecutive subtasks. The skill would recursively place the lowermost block of the remaining stack first, and then the rest of the stack would follow. Subgoal interactions would be implicitly taken care of in the recursive definition of the skill, and the skill would implicitly contain 'knowledge' of gravity.

Obviously no skill formation system which does not have a planning mechanism at its disposal can play chess. Each move requires projections of many future board
configurations, hence planning becomes necessary. In this case, the use of a master's planning apparatus does not help in the proposed advice-taking system. The system would ask for a new skill from its master for almost every board configuration.

V.2 APPLYING THE PROPOSED SYSTEM

V.2.1 Where Does the Proposed System Apply?

The preceding chapter arrived at some conclusions on the limits of a skill formation system. How are the conclusions transformed into answers for practical questions such as: in what kind of task domains is the proposed skill formation system applicable, and what kind of domains resist application?

If tasks are decomposable into independent subtasks, so that the only interactions between subtasks occur in the boundaries of consecutive ones, then the proposed system readily applies. A task inherently requiring interactions between two different subactions far apart in the temporal dimension requires an ad hoc skill that rigidly binds the interacting subtasks with intervening subskills. If the number of such interacting tasks is large, the net result
yields a robot frequently asking new skills to be defined.

Complexity is not the only pertinent parameter for applicability. Skills use perceptions for control, and it is this interlacing of perceiving that provides skills with their power. A prerequisite for forming skills for a task domain is, then, the existence of appropriate perceptual expert procedures with enough discriminative power. The perceptual experts do not, however, need to simulate human perceptions. In particular, perceptual faculties need not be comprehensive as the term 'expert' already suggests. For example, if vision is one of the modes of perceiving, no need exists for implementing a general scene analysis procedure. A special recognition procedure for each pertinent individual in the environment will suffice. Also, special arrangements can be made in the surroundings to ease perceiving. The schema outlined in Section II, whereby perceiving rooms and doors can be easily implemented by using discriminative colored patches on walls and doors, represents one such example.

V.2.2 How to Apply the Proposed System?

How should one implement the proposed system provided that, after an analysis of a task domain, tasks seem well-fitted for skill formation? Another analysis of the task domain is called for, this one from the behavioral
viewpoint. One needs to find:

(1) A set of elementary bodily movements that can generate all required action sequences

(2) A set of perceptual experts that provide all required control impulses for those action sequences

Naturally, only technically feasible movements and perceptual experts should be considered. In a search of such sets, optimization over the cardinalities of the sets and ease of implementations are worthy considerations.

Then build the robot: implement the motor organs and sensory devices in the body, program the system (the Computational Model) and the perceptual expert procedures. The robot is then ready for development. Assign a human master to advise the robot. The master should first define the Orienting Map using claim sentences. Then the master would request the robot to perform tasks, define skills for the tasks, and participate in skill modification. The master should begin with elementary tasks and gradually increase their complexity imposing hierarchy on them.

V.3 FUTURE WORK
We discuss future work under two categories: replacing advice-taking by self-development (that is, implementing the Conceptual Model instead of the Computational Model) and augmenting the advice-taking system.

V.3.1 Self-developing System

Many of the remarks that follow are exceedingly brief and speculative. We have earlier argued that skills can be viewed as representing, in Piagetian terms, the robot's action-schemes, and a collection of skills represents the robot's sensori-motor level of intelligence. Then, following Piaget's line of thought, skills form a possible base for abstracting conceptual structures.

The lack of a means for modifying perceptual schemas cuts the proposed schemas short of the developing, perceptual faculties. Without doubt, the way we are mimicking natural perceptual faculties reflects an extremely naive enterprise. How to achieve developing perceptual schemas represents an ambitious future research problem.

In the advice-taking form, the robot acquires new skills simply by asking a human master to define them. In Piagetian terminology 'accommodation' is reserved to mean creation of new action-schemes. Accommodation by the robot on its own also represents an ambitious future research problem. Definition of skills by a human master is not,
however, to be viewed as just a simple trick to overcome a difficult design problem. In the development of the child, for instance, taking advice from parents and imitating them plays important role. The implemented robot can be viewed as imitating the behavior of a competent agent through verbal means. If we assume the existence of a parser, independently from the skill formation process, the cases in the deep case structure of a new kind of request provide some semantic clues to the structure of a new skill. We do not pursue this line of thought further here.

In Piagetian terminology 'assimilation' means modifying the existing action-schemes to respond new needs of the environment. The advice-taking system relies on the human master in this regard. A future research problem is, then, how the robot could modify skills (assimilate) on its own? Modification can at least exhibit either of the following forms:

(1) An attempt to generalize a skill to cover more task instances.

(2) An attempt to structurally modify a skill that has manifested a deficiency in a task instance.

For (1) the Computational Model provides a solution in the advice-taking form; (2), for which we offered no solution, represents debugging of skills. An underlying assumption in the implemented advice-taking system has been that defined skills are correct. Because of the developmental nature of
the proposed robot, the assumption is not, however, as strong as it may look. If a human master of the robot not only defines and participates in modification of skills, but also follows execution of defined skills and immediately corrects observed deficiencies, then correctness of skills follows naturally. A training period which fully covers the task domain results in correct skills.

If debugging by the robot is a design goal, what directions should one pursue? Skills have two structurally separable parts: preconditions and defining sub-skills. A somewhat easier instance of debugging is the case when a precondition is not discriminative enough. A corrective action consists of recognizing the situation and adding new conditions accordingly. A harder achievement is recognition and correction of deficient defining sub-skills by removing or adding them, or changing their ordering. Debugging of skills is closely related to exception handling and error recovery and, hence, all three instances should be handled in a uniform manner.

An orienting map is a set of propositions collectively depicting the geographic organization of the robot's surroundings. The Computational Model singled out one such map, the Orienting Map, that described the adjacency relation of the rooms of a house where the robot was located, and the floor locations of the rooms. ADJ and FLOOR properties attached to the room names indicate the Orienting Map. An Orienting development procedure for
creating orienting maps by abstracting traces of percepts contained in the Perceptual Memory represents a future research problem.

V.3.2 Augmenting the Advice-taking System

Even if we keep the basic structure of the system intact as an advice-taking facility, we can point to some future research to increase its power. First, as we have indicated time and again, providing the system with a problem solver and knowledge representations over that which skills implicitly provide would facilitate the system with the capability of planning. Such a system could either directly invoke an existing skill, or it could invoke the planning mechanism to manipulate the world model and representations of skills and come up with a proper sequence of skills (plan). The system would then let the sequence of the skills guide the behavior. The system should be able to decide when to resort to planning and, on the other hand, when to acquire new skills. Refer to V.1.2 for further discussion on planning.

The robot's question-answering capabilities are also left for future research. We conducted in the Computational Model an incomplete discussion on how skills implicitly represent knowledge of the world, and demonstrated how this knowledge can be expressed. Since the discussion on
'knowledge' was in the Computational Model more or less as a proposal for future research, it will not be repeated here.

The Propositional Memory contains abstracted properties from the traces of percepts. The implemented system defined a procedure for abstracting the property LOCATION for objects. Implementation of other abstractions from percepts is left for future research.

V.4 CONCLUSION

We have designed and demonstrated, using simulation, a robot system which holds its knowledge and practical competence in action form. In the implemented version of this system, if the robot does not possess competence for a given task, it prompts its human master to give advice, and then incorporates advice in the existing store of competence. Because complexity of advice is invariant, the system is open-ended. Advice is given in the form of a skill. We defined three basic forms of skills. Skills, using perception for control, directly recognize states of the immediately surrounding world and guide behavior without the use of an intermediary representation. We contrasted skilled behavior with planned behavior in which a problem solver manipulates a comprehensive world model producing a plan which a monitor then executes. We argued that although
a skilled system is less powerful than a planning system in terms of task complexity, a skilled system 1) performs those tasks it can perform more efficiently, and 2) is less vulnerable to inconsistencies than plans. A future system, which combines the power of planning and the efficiency of skills by using both a world model and representations of skills as operators would be an interesting advance.
APPENDIX A: SAMPO'S SKILLS

Skills are represented as ordered pairs: (<deep structure schema> <skill definition>). EXCLUSIVE and REPEAT-UNTIL type skills are indicated by their type names; a missing type indicator implies an ORDINARY skill.

((VERB GO-RIGHT) (TRAJECT (PREP AROUND) (NOUN OBSTACLE))
 (DESTIN (PREP TO) (NOUN GO-TO-ABLE))))

(EXCLUSIVE (((SEE GO-TO-ABLE)
  (NOT (BODY-SENSE FRONT)))
 (DONE))
 (((BODY-SENSE LEFT)
   (BODY-SENSE RIGHT))
   (TURN-RIGHT UNTIL (NOT (BODY-SENSE RIGHT))))
 (((BODY-SENSE FRONT)
   (BODY-SENSE RIGHT))
   (TURN-RIGHT UNTIL (NOT (BODY-SENSE RIGHT))))
 (T)
 (TURN-RIGHT UNTIL (NOT (BODY-SENSE FRONT)))
 (MOVE-FORWARD UNTIL (1 STEPS))
 (TURN-LEFT UNTIL (SEE GO-TO-ABLE))
 (GO-RIGHT AROUND OBSTACLE TO GO-TO-ABLE)))

((VERB GO-LEFT) (TRAJECT (PREP AROUND) (NOUN OBSTACLE))
 (DESTIN (PREP TO) (NOUN GO-TO-ABLE))))

(EXCLUSIVE (((SEE GO-TO-ABLE)
  (NOT (BODY-SENSE FRONT)))
 (DONE))
 (((BODY-SENSE RIGHT)
   (BODY-SENSE LEFT))
   (TURN-LEFT UNTIL (NOT (BODY-SENSE LEFT))))
 (((BODY-SENSE FRONT)
   (BODY-SENSE LEFT))
   (TURN-LEFT UNTIL (NOT (BODY-SENSE LEFT))))
 (T)
 (TURN-LEFT UNTIL (NOT (BODY-SENSE FRONT)))
 (MOVE-FORWARD UNTIL (1 STEPS))
 (TURN-RIGHT UNTIL (SEE GO-TO-ABLE))
 (GO-LEFT AROUND OBSTACLE TO GO-TO-ABLE)))
((VERB KEEP-PUTTING) (OBJECT (NOUN PUT-ABLE))
  (DESTIN (PREP ASIDE)))

(EXCLUSIVE (((SENSE-OPPOSITE-TURNING-POINT))
  (UNGRASP-FRONT PUT-ABLE))
  (((BODY-SENSE LEFT-TURN)
    (NOT (BODY-SENSE RIGHT)))
   (MOVE-BACKWARD UNTIL (1 STEPS))
   (MOVE-RIGHT UNTIL (1 STEPS))
   (KEEP-PUTTING PUT-ABLE ASIDE))
  (((BODY-SENSE LEFT-TURN)
    (NOT (BODY-SENSE BACK)))
   (MOVE-BACKWARD UNTIL (1 STEPS))
   (KEEP-PUTTING PUT-ABLE ASIDE))
  (((BODY-SENSE LEFT-TURN)
    (BODY-SENSE BACK))
   (TURN-RIGHT UNTIL (BODY-SENSE RIGHT-TURN))
   (MOVE-BACKWARD UNTIL (1 STEPS))
   (KEEP-PUTTING PUT-ABLE ASIDE))
  ((T)
   (TURN-LEFT UNTIL (OR (BODY-SENSE LEFT-TURN)
    (SENSE-OPPOSITE-TURNING-POINT)))
   (KEEP-PUTTING PUT-ABLE ASIDE)))

((VERB PUT) (OBJECT (NOUN PUT-ABLE))
  (DESTIN (PREP ASIDE)))

((T)
  (FIX-START-TURNING-POINT)
  (KEEP-PUTTING PUT-ABLE ASIDE))

((VERB TRY-REMOVE) (OBJECT (NOUN OBSTACLE))
  (DESTIN (PREP TO) (NOUN GO-TO-ABLE)))

(EXCLUSIVE (((BE-BY GO-TO-ABLE))
  (DONE))
  (((SEE GO-TO-ABLE)
    (NOT (BODY-SENSE FRONT)))
   (DONE))
  ((T)
   (GRASP-FRONT OBSTACLE)
   (PUT OBSTACLE ASIDE)
   (GO TO GO-TO-ABLE)))
((VERB ENTER) (DESTIN (PREP IN) (NOUN ENTER-IN-ABLE)))

(EXCLUSIVE (((BE-IN ENTER-IN-ABLE))
(DONE))
(((BODY-SENSE FRONT))
(GRASP-FRONT OBSTACLE)
(MOVE-FORWARD UNTIL (BE-IN ENTER-IN-ABLE))
(UNGRASP-FRONT OBSTACLE))
(T)
(MOVE-FORWARD UNTIL (OR (BE-IN ENTER-IN-ABLE)
(BODY-SENSE FRONT)))
(ENTER IN ENTER-IN-ABLE)))

((VERB GO) (DESTIN (PREP TO) (NOUN GO-TO-ABLE)))

(EXCLUSIVE (((SEE GO-TO-ABLE)
(NOT (SEE OBSTACLE IN-FRONT-OF GO-TO-ABLE)))
(MOVE-FORWARD UNTIL (BE-BY GO-TO-ABLE)))
(((SEE GO-TO-ABLE)
(SEE OBSTACLE IN-FRONT-OF GO-TO-ABLE))
(MOVE-FORWARD UNTIL (BODY-SENSE FRONT))
(CIRCUMVENT OBSTACLE TO GO-TO-ABLE)
(GO TO GO-TO-ABLE))
(((BE-BY GO-TO-ABLE))
(DONE)))

((VERB GO) (DESTIN (PREP INTO) (NOUN ELEVATOR)))

(((BE-IN-ADJ-TO ELEVATOR))
(GO TO ELEVATOR-DOOR)
(CALL-ELEVATOR)
(TURN-RIGHT UNTIL (SEE ELEVATOR-DOOR))
(WAIT UNTIL (SEE ELEVATOR-DOOR-OPEN))
(ENTER IN ELEVATOR))

((VERB EXIT) (SOURCE (PREP FROM) (NOUN ELEVATOR)))

(T)
(WAIT UNTIL (SEE ELEVATOR-DOOR-OPEN))
(MOVE-FORWARD UNTIL (NOT (BE-IN ELEVATOR))))

((VERB GO) (DESTIN (PREP TO) (NOUN FLOOR)))

(((NOT (BE-ON FLOOR)))
(GO INTO ELEVATOR)
(PUSH-BUTTON FLOOR)
(TURN-LEFT UNTIL (SEE ELEVATOR-DOOR))
(WAIT UNTIL (SEE ELEVATOR-DOOR-CLOSED))
(EXIT FROM ELEVATOR))
((VERB GO) (DESTIN (PREP INTO) (NOUN GO-INTO-ABLE)))

(EXCLUSIVE (((BE-IN GO-INTO-ABLE)))
  (DONE))
  (((NOT (BE-IN GO-INTO-ABLE))
    (BE-ON-FLOOR-OF GO-INTO-ABLE)
    (BE-IN-ADJ-TO GO-INTO-ABLE))
  (GO TO GO-INTO-ABLE-DOOR)
  (OPEN GO-INTO-ABLE-DOOR)
  (ENTER IN GO-INTO-ABLE)))

((VERB KEEP-LOOKING) (OBJECT (PREP FOR) (NOUN LOOK-FOR-ABLE)))

(EXCLUSIVE (((OR (SEE LOOK-FOR-ABLE)
    (SENSE-START-TURNING-POINT)))
  (DONE))
  (((BODY-SENSE RIGHT-TURN)
    (NOT (BODY-SENSE RIGHT)))
  (MOVE-RIGHT UNTIL (1 STEPS))
  (KEEP-LOOKING FOR LOOK-FOR-ABLE))
  (((BODY-SENSE RIGHT-TURN)
    (NOT (BODY-SENSE FRONT)))
  (MOVE-FORWARD UNTIL (1 STEPS))
  (KEEP-LOOKING FOR LOOK-FOR-ABLE))
  (((BODY-SENSE RIGHT-TURN)
    (BODY-SENSE FRONT))
  (TURN-LEFT UNTIL (NOT (BODY-SENSE FRONT)))
  (MOVE-FORWARD UNTIL (1 STEPS))
  (KEEP-LOOKING FOR LOOK-FOR-ABLE))
  (T)
  (TURN-RIGHT UNTIL (OR (SEE LOOK-FOR-ABLE)
    (BODY-SENSE RIGHT-TURN)
    (SENSE-START-TURNING-POINT)))
  (KEEP-LOOKING FOR LOOK-FOR-ABLE)))

((VERB LOOK) (OBJECT (PREP FOR) (NOUN LOOK-FOR-ABLE)))

(T)
  (FIX-START-TURNING-POINT)
  (KEEP-LOOKING FOR LOOK-FOR-ABLE))

((VERB PICK) (OBJECT (PREP UP) (NOUN PICK-UP-ABLE)))

(EXCLUSIVE (((NOT (SEE PICK-UP-ABLE)))
  (DONE))
  (((SEE PICK-UP-ABLE))
  (GO TO PICK-UP-ABLE)
  (GRASP-BACK PICK-UP-ABLE)))
((VERB TAKE) (OBJECT (NOUN TAKE-ABLE)))

((T)
 (LOOK FOR TAKE-ABLE)
 (PICK UP TAKE-ABLE))

((VERB LEAVE) (OBJECT (NOUN LEAVE-ABLE))
 (DESTIN (PREP BY) (NOUN GO-TO-ABLE)))

(EXCLUSIVE (((NOT (SENSE-CARRYING)))
 (DONE))
 ((T)
 (LOOK-FOR GO-TO-ABLE)
 (PUT DOWN LEAVE-ABLE BY GO-TO-ABLE)))

((VERB LEAVE) (OBJECT (NOUN LEAVE-ABLE))
 (DESTIN (PREP BY) (NOUN WALL)))

(EXCLUSIVE (((NOT (SENSE-CARRYING)))
 (DONE))
 (((SENSE-CARRYING)
 (NOT (SEE OBSTACLE)))
 (MOVE-FORWARD UNTIL (BODY-SENSE FRONT))
 (TURN-RIGHT UNTIL (BODY-SENSE RIGHT-TURN))
 (UNGRASP-BACK LEAVE-ABLE)))

((VERB PUT) (OBJECT (PREP DOWN) (NOUN PUT-DOWN-ABLE))
 (DESTIN (PREP BY) (NOUN GO-TO-ABLE)))

(EXCLUSIVE (((NOT (SENSE-CARRYING)))
 (DONE))
 (((NOT (SEE GO-TO-ABLE)))
 (LEAVE PUT-DOWN-ABLE BY WALL))
 ((T)
 (GO TO GO-TO-ABLE)
 (TURN-RIGHT UNTIL (BODY-SENSE RIGHT-TURN))
 (UNGRASP-BACK PUT-DOWN-ABLE)))

((VERB COME) (DESTIN (PREP BACK)))

(((RECALL ?ROOM BE-IN FIRST))
 (GO INTO $ROOM)
 (RELEASE $ROOM))

((VERB REMEMBER-ROOMS))

(((RECALL ?ROOM))
 (DONE))
((VERB REMEMBER-OTHER-ROOMS))

(((RECALL ?ROOM $ROOM EXCEPT BE-IN LAST))
  (DONE))

((VERB REMEMBER-ROOMS) (OBJECT (PREP EXCEPT)
  (NOUN GO-INTO-ABLE)))

(((RECALL ?ROOM EXCEPT GO-INTO-ABLE))
  (DONE))

((VERB FIND) (OBJECT (NOUN FIND-ABLE)))

(((BE-IN ROOM))
 ((REMEMBER-ROOMS)
  (LOOK FOR FIND-ABLE)
  (FIND FIND-ABLE IN ROOMS)
  (RELEASE $ROOM)))

((VERB FIND) (OBJECT (NOUN FIND-ABLE))
  (LOCAT (PREP IN) (NOUN ROOMS)))

(EXCLUSIVE (((SEE FIND-ABLE))
  (DONE))
  (((RECALL ?ROOM LOC-OF FIND-ABLE))
   (GO INTO $ROOM)
   (RELEASE $ROOM)
   (LOOK FOR FIND-ABLE)
   (SEARCH FOR FIND-ABLE))
  (T)
  (SEARCH FOR FIND-ABLE)))

((VERB SEARCH) (OBJECT (PREP FOR) (NOUN SEARCH-ABLE)))

(REPEAT-UNTIL (OR (SEE SEARCH-ABLE)
  (NOT (RECALL ?ROOM $ROOM)))
  (REMEMBER-OTHER-ROOMS)
  (GO INTO $ROOM)
  (LOOK FOR SEARCH-ABLE))
((VERB FIND) (OBJECT (NOUN FIND-ABLE))
 (LOCAT (PREP IN) (NOUN ROOMS))
 (LOCAT2 (PREP EXCEPT) (NOUN GO-INTO-ABLE)))

(EXCLUSIVE (((SEE FIND-ABLE))
 (DONE))
 (((RECALL ?ROOM LOC-OF FIND-ABLE
   EXCEPT GO-INTO-ABLE))
 (GO INTO $ROOM)
 (RELEASE $ROOM)
 (LOOK FOR FIND-ABLE)
 (SEARCH FOR FIND-ABLE))
 ((T)
  (REARCHE FOR FIND-ABLE)))

((VERB BRING) (OBJECT (NOUN BRING-ABLE))
 (LOCAT (PREP INTO) (NOUN GO-INTO-ABLE))
 (LOCAT2 (PREP BY) (NOUN GO-TO-ABLE)))

(EXCLUSIVE (((BE-IN GO-INTO-ABLE)
   (RECALL ?ROOM ADJ GO-INTO-ABLE))
 (GO INTO $ROOM)
 (RELEASE $ROOM)
 (BRING BRING-ABLE INTO GO-INTO-ABLE
   BY GO-TO-ABLE))
 ((T)
  (REMEMBER-ROOMS EXCEPT GO-INTO-ABLE)
  (LOOK FOR BRING-ABLE)
 (FIND BRING-ABLE IN ROOMS EXCEPT GO-INTO-ABLE)
 (RELEASE $ROOM)
 (PICK UP BRING-ABLE)
 (GO INTO GO-INTO-ABLE)
 (LEAVE BRING-ABLE BY GO-TO-ABLE))))
APPENDIX B: LOGICAL TAXONOMY OF CONCEPTS (LTOC)

(SOMETHING (SONS LOOK-FOR-ABLE ENTER-IN-ABLE GO-TO-ABLE-2
WALL ROOMS)))

(LOOK-FOR-ABLE (FATHER SOMETHING)
(SONS GO-TO-ABLE ELEVATOR-DOOR-OPEN))

(FIND-ABLE (ALIAS LOOK-FOR-ABLE))

(SEARCH-ABLE (ALIAS LOOK-FOR-ABLE))

(ENTER-IN-ABLE (FATHER SOMETHING)
(SONS GO-TO-ABLE ELEVATOR))

(GO-TO-ABLE-2 (FATHER SOMETHING)
(SONS $FLOOR FLOOR1 FLOOR2 FLOOR3))

(GO-TO-ABLE (FATHER LOOK-FOR-ABLE)
(SONS PICK-UP-ABLE ELEVATOR-DOOR
ELEVATOR-DOOR-CLOSED
R101-DOOR R102-DOOR R103-DOOR R104-DOOR
R105-DOOR R106-DOOR R201-DOOR R202-DOOR
R203-DOOR R204-DOOR R205-DOOR R206-DOOR
R301-DOOR R302-DOOR R303-DOOR R304-DOOR
R305-DOOR R306-DOOR))

(PICK-UP-ABLE (FATHER GO-TO-ABLE)
(SONS TABLE CHAIR BASKET OBSTACLE))

(LEAVE-ABLE (ALIAS PICK-UP-ABLE))

(TAKE-ABLE (ALIAS PICK-UP-ABLE))

(PUT-ABLE (ALIAS PICK-UP-ABLE))

(PUT-DOWN-ABLE (ALIAS PICK-UP-ABLE))

(BRING-ABLE (ALIAS PICK-UP-ABLE))

(GO-INTO-ABLE (FATHER ENTER-IN-ABLE)
(SONS $ROOM R101 R102 R103 R104 R105 R106
R201 R202 R203 R204 R205 R206
R301 R302 R303 R304 R305 R306))

(ROOMS (FATHER SOMETHING))

(WALL (FATHER SOMETHING))

(ELEVATOR-DOOR (FATHER GO-TO-ABLE))

(ELEVATOR-DOOR-CLOSED (FATHER GO-TO-ABLE))

(ELEVATOR-DOOR-OPEN (FATHER LOOK-FOR-ABLE))

(R101-DOOR (FATHER GO-TO-ABLE))
(R102-DOOR (FATHER GO-TO-ABLE))
(R103-DOOR (FATHER GO-TO-ABLE))
(R104-DOOR (FATHER GO-TO-ABLE))
(R105-DOOR (FATHER GO-TO-ABLE))
(R106-DOOR (FATHER GO-TO-ABLE))
(R201-DOOR (FATHER GO-TO-ABLE))
(R202-DOOR (FATHER GO-TO-ABLE))
(R203-DOOR (FATHER GO-TO-ABLE))
(R204-DOOR (FATHER GO-TO-ABLE))
(R205-DOOR (FATHER GO-TO-ABLE))
(R206-DOOR (FATHER GO-TO-ABLE))
(R301-DOOR (FATHER GO-TO-ABLE))
(R302-DOOR (FATHER GO-TO-ABLE))
(R303-DOOR (FATHER GO-TO-ABLE))
(R304-DOOR (FATHER GO-TO-ABLE))
(R305-DOOR (FATHER GO-TO-ABLE))
(R306-DOOR (FATHER GO-TO-ABLE))
(ELEVATOR (FATHER ENTER-IN-ABLE))
(R101 (FATHER GO-INTO-ABLE))
(R102 (FATHER GO-INTO-ABLE))
(R103 (FATHER GO-INTO-ABLE))
(R104 (FATHER GO-INTO-ABLE))
(R105 (FATHER GO-INTO-ABLE))
(R106 (FATHER GO-INTO-ABLE))
(R201 (FATHER GO-INTO-ABLE))
(R202 (FATHER GO-INTO-ABLE))
(R203 (FATHER GO-INTO-ABLE))
(R204 (FATHER GO-INTO-ABLE))
(R205 (FATHER GO-INTO-ABLE))
(R206 (FATHER GO-INTO-ABLE))
(R301 (FATHER GO-INTO-ABLE))
(R302 (FATHER GO-INTO-ABLE))
(R303 (FATHER GO-INTO-ABLE))
(R304 (FATHER GO-INTO-ABLE))
(R305 (FATHER GO-INTO-ABLE))
(R306 (FATHER GO-INTO-ABLE))
(ROOM (FATHER GO-INTO-ABLE))
(ROOM (FATHER GO-TO-ABLE-2))
(FLOOR1 (FATHER GO-TO-ABLE-2))
(FLOOR2 (FATHER GO-TO-ABLE-2))
(FLOOR3 (FATHER GO-TO-ABLE-2))
(BASKET (FATHER PICK-UP-ABLE))
(TABLE (FATHER PICK-UP-ABLE))
(CHAIR (FATHER PICK-UP-ABLE))
(OBSTACLE (FATHER PICK-UP-ABLE))
APPENDIX C: PERCEPTUAL EXPECTATIONS

A perceptual expectation is represented as an ordered pair: (<perception> <skill>). <skill> represents a behavioral solution to a subgoal of getting <perception> to evaluate TRUE.

(((NOT (BE-BY GO-TO-ABLE)) (T) (TURN-LEFT UNTIL (NOT (BE-BY GO-TO-ABLE))))))

(((BODY-SENSE RIGHT) (T) (TURN-LEFT UNTIL (BODY-SENSE RIGHT))))))

(((BODY-SENSE LEFT) (T) (TURN-RIGHT UNTIL (BODY-SENSE LEFT))))))

(((NOT (SEE LOOK-FOR-ABLE)) (T) (TURN-LEFT UNTIL (NOT (SEE LOOK-FOR-ABLE))))))

(((SEE LOOK-FOR-ABLE) (T) (LOOK FOR LOOK-FOR-ABLE)))

(((BE-BY GO-TO-ABLE) (T) (GO TO GO-TO-ABLE))))

(((BE-IN GO-INTO-ABLE) (T) (GO INTO GO-INTO-ABLE))))

(((BE-IN-ADJ-TO GO-INTO-ABLE) (((RECALL ?ROOM ADJ GO-INTO-ABLE)) (GO INTO $ROOM) (RELEASE $ROOM))))

(((BE-ON-FLOOR-OF GO-INTO-ABLE) (((RECALL ?FLOOR FLOOR-OF GO-INTO-ABLE)) (GO TO $FLOOR) (RELEASE $FLOOR))))
APPENDIX D: PARSING RESULTS

Parsing results are represented as ordered pairs:

\([\text{<surface sentence>} \text{<deep case structure>}]\)

\[
\begin{align*}
\text{(! LEAVE LEAVE-ABLE BY GO-TO-ABLE)} & \quad \text{(! TAKE TAKE-ABLE)} \\
\text{((VERB LEAVE) (OBJECT (NOUN LEAVE-ABLE))} & \quad \text{((VERB TAKE) (OBJECT (NOUN TAKE-ABLE)))} \\
& \quad \text{DESTIN (PREP BY) (NOUN GO-TO-ABLE))}) \\
\text{(! PICK UP PICK-UP-ABLE)} & \quad \text{(! PUT DOWN PUT-DOWN-ABLE BY GO-TO-ABLE)} \\
\text{((VERB PICK) (OBJECT (PREP UP) (NOUN PICK-UP-ABLE))} & \quad \text{((VERB PUT) (OBJECT (PREP DOWN) (NOUN PUT-DOWN-ABLE)))} \\
& \quad \text{DESTIN (PREP BY) (NOUN GO-TO-ABLE))}) \\
\text{(! LOOK FOR LOOK-FOR-ABLE)} & \quad \text{(! KEEP-LOOKING FOR LOOK-FOR-ABLE)} \\
\text{((VERB LOOK) (OBJECT (PREP FOR) (NOUN LOOK-FOR-ABLE)))} & \quad \text{((VERB KEEP-LOOKING) (OBJECT (PREP FOR)} \\
& \quad \text{(NOUN LOOK-FOR-ABLE)))} \\
\text{(! COME BACK)} & \quad \text{(! CIRCUMVENT OBSTACLE TO GO-TO-ABLE)} \\
\text{((VERB COME) (DESTIN (PREP BACK)))} & \quad \text{((VERB CIRCUMVENT) (OBJECT (NOUN OBSTACLE))} \\
& \quad \text{(DESTIN (PREP TO) (NOUN GO-TO-ABLE)))} \\
\text{(! TURN AROUND)} & \quad \text{(! KEEP-PUTTING PUT-DOWN-ABLE ASIDE)} \\
\text{((VERB TURN) (TRAJECT (PREP AROUND)))} & \quad \text{((VERB KEEP-PUTTING) (OBJECT (NOUN PUT-DOWN-ABLE))} \\
& \quad \text{(DESTIN (PREP ASIDE))}) \\
\text{(! GO-LEFT AROUND OBSTACLE TO GO-TO-ABLE)} & \quad \text{(! GO-RIGHT AROUND OBSTACLE TO GO-TO-ABLE)} \\
\text{((VERB GO-LEFT) (TRAJECT (PREP AROUND) (NOUN OBSTACLE))} & \quad \text{((VERB GO-RIGHT) (TRAJECT (PREP AROUND) (NOUN OBSTACLE))} \\
& \quad \text{(DESTIN (PREP TO) (NOUN GO-TO-ABLE)))} & \quad \text{(DESTIN (PREP TO) (NOUN GO-TO-ABLE))))} \\
\end{align*}
\]
(((PUT PUT-DOWN-ABLE ASIDE)
 ((VERB PUT) (OBJECT (NOUN PUT-DOWN-ABLE))
  (DESTIN (PREP ASIDE))))

(((GO TO GO-TO-ABLE)
 ((VERB GO) (DESTIN (PREP TO) (NOUN GO-TO-ABLE))))

(((TRY-LEFT-WAY TO GO-TO-ABLE)
 ((VERB TRY-LEFT-WAY) (DESTIN (PREP TO) (NOUN GO-TO-ABLE))))

(((TRY-RIGHT-WAY TO GO-TO-ABLE)
 ((VERB TRY-RIGHT-WAY) (DESTIN (PREP TO)
  (NOUN GO-TO-ABLE))))

(((TRY-REMOVE OBSTACLE TO GO-TO-ABLE)
 ((VERB TRY-REMOVE) (OBJECT (NOUN OBSTACLE))
  (DESTIN (PREP TO) (NOUN GO-TO-ABLE))))

(((ENTER IN ENTER-IN-ABLE)
 ((VERB ENTER) (DESTIN (PREP IN) (NOUN ENTER-IN-ABLE))))

(((GO INTO GO-INTO-ABLE)
 ((VERB GO) (DESTIN (PREP IN) (NOUN GO-INTO-ABLE))))

(((REMEMBER-ROOMS)
 ((VERB REMEMBER-ROOMS))

(((REMEMBER-ROOMS EXCEPT GO-INTO-ABLE)
 ((VERB REMEMBER-ROOMS) (OBJECT (PREP EXCEPT)
  (NOUN GO-INTO-ABLE))))

(((REMEMBER-OTHER-ROOMS))
 ((VERB REMEMBER-OTHER-ROOMS))

(((SEARCH FOR SEARCH-ABLE)
 ((VERB SEARCH) (OBJECT (PREP FOR) (NOUN SEARCH-ABLE))))

(((FIND FIND-ABLE IN ROOMS)
 ((VERB FIND) (OBJECT (NOUN FIND-ABLE))
  (LOCAT (PREP IN) (NOUN ROOMS))))

(((FIND FIND-ABLE)
 ((VERB FIND) (OBJECT (NOUN FIND-ABLE))))

(((FIND FIND-ABLE IN ROOMS EXCEPT GO-INTO-ABLE)
 ((VERB FIND) (OBJECT (FIND-ABLE))
  (LOCAT (PREP IN) (NOUN ROOMS))
  (LOCAT2 (PREP EXCEPT) (NOUN GO-INTO-ABLE))))
((! BRING BRING-ABLE INTO GO-INTO-ABLE BY GO-TO-ABLE)
  ((VERB BRING) (OBJECT (NOUN BRING-ABLE))
   (DESTIN (PREP INTO) (NOUN GO-INTO-ABLE))
   (LOCAT (PREP BY) (NOUN GO-TO-ABLE))))

((! EXIT FROM ELEVATOR)
  ((VERB EXIT) (SOURCE (PREP FROM) (NOUN ELEVATOR))))

((! FIND FIND-ABLE)
  ((VERB FIND) (OBJECT (NOUN FIND-ABLE))))

((! FIND FIND-ABLE IN ROOMS)
  ((VERB FIND) (OBJECT (NOUN FIND-ABLE))
   (LOCAT (PREP IN) (NOUN ROOMS))))
APPENDIX E: THE ORIENTING MAP

(R101 (ADJ R102 R105 ELEVATOR)
(FLOOR FLOOR1))

(R102 (ADJ R101 R103 R104 R105)
(FLOOR FLOOR1))

(R103 (ADJ R102)
(FLOOR FLOOR1))

(R104 (ADJ R102 R105)
(FLOOR FLOOR1))

(R105 (ADJ R101 R102 R104 R106)
(FLOOR FLOOR1))

(R106 (ADJ R105)
(FLOOR FLOOR1))

(R201 (ADJ R201 R205 ELEVATOR)
(FLOOR FLOOR2))

(R202 (ADJ R201 R203 R204 R205)
(FLOOR FLOOR2))

(R203 (ADJ R202)
(FLOOR FLOOR2))

(R204 (ADJ R202 R205)
(FLOOR FLOOR2))

(R205 (ADJ R201 R202 R204 R206)
(FLOOR FLOOR2))

(R206 (ADJ R205)
(FLOOR FLOOR2))

(R301 (ADJ R302 R305 ELEVATOR)
(FLOOR FLOOR3))

(R302 (ADJ R301 R303 R304 R305)
(FLOOR FLOOR3))

(R303 (ADJ R302)
(FLOOR FLOOR3))

(R304 (ADJ R302 R305)
(FLOOR FLOOR3))

(R305 (ADJ R301 R302 R304 R306)
(FLOOR FLOOR3))
(R306 (ADJ R305)
(FLOOR FLOOR3))
BIBLIOGRAPHY


[Tate] A. Tate, "Interacting Goals and Their Use", *IJCAI*, 1975.


