*Advanced Robotics*, Vol. 7, No. 4, pp. 309–328 (1993)


*Paper*

# Integration of problem-solving and learning in intelligent robots

JUN MIURA,*‡ ISAO SHIMOYAMA† and HIROFUMI MIURA†

**Mechanical Engineering for Computer-Controlled Machinery, Osaka University, Suita, Osaka 565, Japan*

†*Mechano-Informatics, University of Tokyo, Hongo, Tokyo 113, Japan*

Received for *AR* 11 March 1990

**Abstract**—In the real world, there are many kinds of uncertainties in the motions of robots. Moreover, robots cannot always get sufficient knowledge about tasks in advance. Intelligent robots therefore have to possess both problem-solving ability, to decide on the proper motions with insufficient knowledge, and learning ability, to acquire knowledge from experiences. Effective integration of these two kinds of ability is also important.

In this paper, we describe an experimental system named ARPEX-L (Automatic Robot Planning and Execution System with Learning Ability). ARPEX-L consists of the integration of reactive planning and learning. We applied ARPEX-L to two kinds of robot task: a pick-and-place task and a pushing block task. The former is a simple example of automatic robot programming. The latter is an attempt to make an actual robot learn by a symbolic framework. These applications show the validity of our approach in constructing intelligent robot systems. Current defects of the system and future work are also described.

## 1. INTRODUCTION

Intelligent robots which perform autonomously in the real world must possess both problem-solving ability, which involves making a plan and executing it, and learning ability, which is to acquire useful knowledge from experiences. In the AI (Artificial Intelligence) field, there have been many studies on problem-solving and learning. Some of these studies have treated learning in the problem-solving process [1–5]. However, in refs 1–5 it was assumed that knowledge about the operators used for problem-solving is complete. For example, in STRIPS [1]-like systems, preconditions, add-lists, and delete-lists of operators are given beforehand. Therefore, changes of the environment caused by execution of the operators are completely predictable and we can argue every issue in the closed world inside computers.

In reality, as information about operators is incomplete, mainly because of the uncertainty of motions, we must consider the following. First, if changes of the environment caused by operators (we call these changes 'effects' of operators) are unknown, robots need the ability to acquire the effects of operators. Second, since the effects of operators depend on the situations in which they are executed, it is difficult to give a robot every possible effect of the operators in advance, and

‡To whom correspondence should be addressed.

predicted effects cannot be free from uncertainty. Robots must therefore possess the ability to manage uncertainty.

In this paper, we describe an experimental system named ARPEX-L (Automatic Robot Planning and Execution System with Learning Ability). In ARPEX-L, we integrate problem-solving and learning to cope with the issues mentioned above. ARPEX-L treats real world problems in a symbolic framework. Note that ARPEX-L is not merely a problem-solving system or a learning system, but an integrated system of problem-solving and learning. Once ARPEX-L is given initial knowledge and a goal, it works without any help from users and learns from experiences. To construct such an integrated system is basic research for automatic robot programming in the future.

In Section 2, we describe the design policy of our system. We explain each part of ARPEX-L in Sections 3–6. In these sections, we use the three pick-and-place

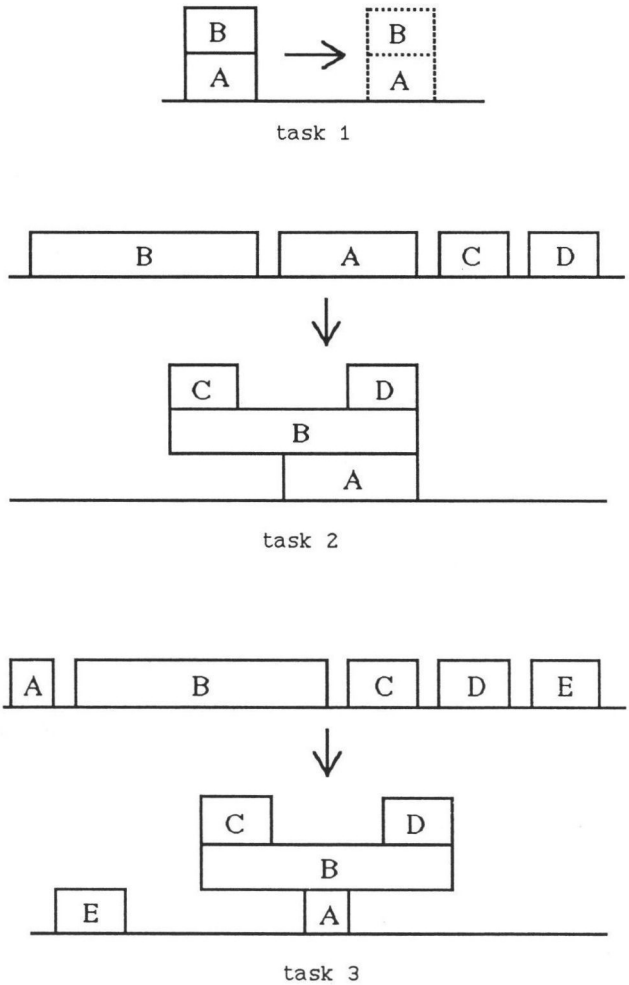



**Figure 1.** Three pick-and-place tasks.

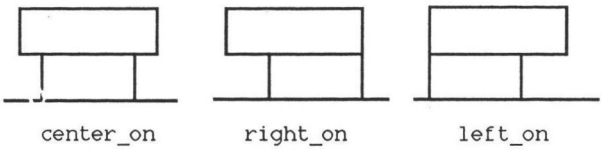



**Figure 2.** Three ways to stack blocks.

tasks shown in Fig. 1 for explanation. In these tasks, we limit the method of stacking blocks to three as shown in Fig. 2, and assume that a manipulator can carry only one block at a time. In Sections 7 and 8, we apply ARPEX-L to two tasks, a pick-and-place task and a pushing block task. The former is a simple example of automatic robot programming. The latter is the first attempt to make a real robot learn in a symbolic framework.

## 2. BASIC DESIGN OF THE SYSTEM

### *2.1. Representation, utilization, and learning of knowledge*

Some kind of knowledge is applicable in limited situations. For example, the effect of an operator in putting one block on top of another block varies according to the situation, as shown in Fig. 3. Knowledge which is applicable in a specific situation

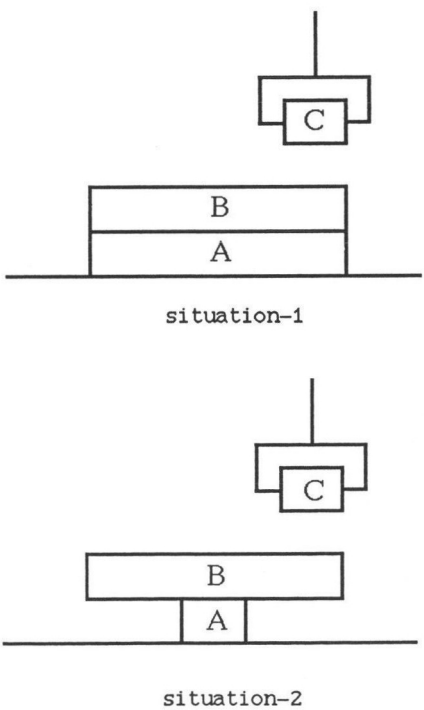



**Figure 3.** Effects of an operator and situations.

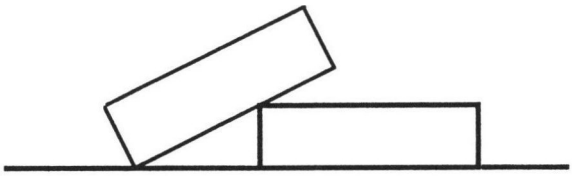


**Figure 4.** An erroneous situation.

(e.g. Fig. 4) may not be applicable in other situations. Therefore it is necessary to represent knowledge with applicable situations and to utilize it by referring to the current situation. Some kind of knowledge, for example that blocks must be stacked from bottom to top, is applicable regardless of the situation and can be given as common sense in advance. However, robots still need learning ability, since knowledge which is applicable in limited situations may be acquired only from experiences.

### *2.2. Basic strategy of problem-solving*

If knowledge is complete, we can simulate every change caused by operators and can make a plan in advance. In reality, however, changes of the environment are not completely predictable and robots cannot make a decision about the next motion without referring to the current situation. Therefore, ARPEX-L decides on the next movements one by one according to the current state of the environment. This manner is based on the notion that when knowledge is insufficient, selecting locally optimal motions is efficient as a result. However, for more efficient problem-solving, we must use knowledge which is applicable regardless of the situation. Therefore in ARPEX-L, a robot makes a rough plan in advance and makes a more precise decision at the execution time. Let us take task 1 as an example. Before execution, the robot decides to carry block $A$ first and then stack block $B$. At the execution time, the robot carries out planned movements, if possible. When an error occurs or no planned motion is available, the robot selects the most appropriate motion according to the situation. This two-stage motion selection is similar to that of reactive planning [6, 7].

## 3. REPRESENTATION OF KNOWLEDGE

### *3.1. Description of the situation*

We describe a situation (a state of the environment) with a predicate *relation* (*Obj1*, *Obj2*, *Relation*). The value of the argument *Relation* depends on the current problem domain. In the pick-and-place task, we use the following six relations: the three relations (*center_on*, *right_on*, *left_on*) shown in Fig. 2, *touching*(*Trans*) (two objects are touching each other in a relation rather than the above three; *Trans* is a transformation matrix), *on_table*(*Pos*) (an object is at *Pos* on the table), and *grasped* (an object is grasped by a manipulator).

### *3.2. Operator and ACT*

Each operator has information about its predicate description, precondition, and effect. Besides, primitive operators have a description of the motions of the robot, and macro operators have a description of their lower level operators to which these operators are reduced to.

The effect of an operator is called ACT. Each ACT describes what states are established by execution of the operator. An ACT consists of a situation-independent part and a situation-dependent part. They are called basic-ACT (b-ACT) and extra-ACT (e-ACT) respectively. A b-ACT describes states which are considered to be established at least by the operator, and is given in advance. An e-ACT is represented by a triple of an operator, a situation, and the state to be established. It is either given beforehand or acquired by the robot. Figure 5 shows an example of e-ACT acquired in task 1. This e-ACT means that we can establish the goal state from its initial state by executing macro operator *mo-1*. From this point, we use the 'ACT' to indicate b-ACT and e-ACT together.

We initially gave the system the nine operators listed in Table 1 in the pick-and-place task. In Table 1 the left side indicates the names of operators and the right side indicates their ACTs. Both *pick_up* and *put_on_xxx* are primitive operators. *carry_on_xxx* are macro operators which consist of these primitive operators. In this case, all ACTs are b-ACTs.

| ACT-1 | |
|---|---|
| operator | *mo-1* |
| situation | *relation(#A, #Table, on_table(#CurrPosA))*<br>*relation(#B, #A, center_on)* |
| effects | *relation(#A, #Table, on_table(#PosA))*<br>*relation(#B, #A, center_on)* |

**Figure 5.** An example of extra-ACT.

**Table 1.**
Initial operators in the pick and place task

| Primitive operators | |
|---|---|
| *pick_up(Obj, Pos)* | *relation(Obj, # Robot, grasped)* |
| *put_on_table(Obj, Pos)* | *relation(Obj, # Table, on_table(Pos))* |
| *put_on_centre(Upper, Lower)* | *relation(Upper, Lower, centre_on)* |
| *put_on_right(Upper, Lower)* | *relation(Upper, Lower, right_on)* |
| *put_on_left(Upper, Lower)* | *relation(Upper, Lower, left_on)* |
| **Macro operators** | |
| *carry_on_table(Obj, Pos)* | *relation(Obj, # Table, on_table(Pos))* |
| *carry_on_centre(Upper, Lower)* | *relation(Upper, Lower, centre_on)* |
| *carry_on_right(Upper, Lower)* | *relation(Upper, Lower, right_on)* |
| *carry_on_left(Upper, Lower)* | *relation(Upper, left_on)* |

### 3.3. Operator selection rule

Knowledge used for selecting an appropriate operator in a specific situation is called the *operator selection rule* (*SR*). There are two types of SR: positive-SR and negative-SR. A positive-SR represents knowledge such that in a situation, selection of an operator is preferable. A negative-SR represents knowledge such that in a situation, selection of an operator is not preferable. Robots can easily acquire knowledge of this form from experiences. An SR consists of a situation description, an operator designation, constraints for variables, and a certainty factor.

Figure 6 shows an example of SR. This SR means that it is preferable to put the upper block on the table in the situation shown in Fig. 4. Constraints of variables are set by inductive generalization of past examples. An attribute of objects is represented in the form *Obj^attr* and we decide beforehand which attributes can be used as constraints.

| SR-1 | |
|---|---|
| type | *positive* |
| situation | *relation(Block1, #Table, on_table(Pos))*<br>*relation(Block2, Block1, touching(Trans))* |
| operator | *carry_on_table(Block2, #PosOnTable)* |
| constraints | *30 <= Block1^width <= 40*<br>*30 <= Block2^width <= 40* |
| certainty factor | *0.795* |

**Figure 6.** An example of an operator selection rule.

### 3.4. Planning rule

Knowledge for setting orders between subgoals is called the *planning rule* (*PR*). Setting orders by PRs is the resolution of conflicts between subgoals. If we consider order information in operator selection, we can avoid falling into infinite loops caused by operator selection which is performed only from the local point of view. There are also two types of PR: positive-PR and negative-PR. Let Op1 and Op2 be operators. A positive-PR represents knowledge such that in a situation, executing Op1 before Op2 is preferable. A negative-PR represents knowledge such that in a situation, executing Op1 before Op2 is not preferable. However, note that this negative-PR does not mean that executing Op2 before Op1 is preferable. A PR consists of a situation description, two operator designations, constraints for variables, and a certainty factor.

Let us take task 2. To achieve this task successfully, two PRs are necessary. One is that blocks are stacked from bottom to top. The other is that block *D* is stacked before block *C*. These PRs can be described as shown in Figs 7 and 8. That the situation description of PR-1 (Fig. 7) is 'nil' indicates that this PR is applicable in any situation (situation-independent). Underscores ('_') in Op1 and Op2 are anonymous variables which can match with any term. PR-1 means that it is preferable to execute an operator to stack *Block* on other blocks before another operator to stack other blocks on *Block*. By preparing this kind of PR for every combination of proper operators, knowledge that blocks are stacked from bottom

| PR-1 | |
|---|---|
| type | *positive* |
| Op1 | *carry_on_center(Object, _)* |
| Op2 | *carry_on_center(_, Object)* |
| situation | *nil* |
| certainty factor | *1.000* |

**Figure 7.** An example of a planning rule.

| PR-2 | |
|---|---|
| type | *positive* |
| Op1 | *carry_on_right(#D, Block1)* |
| Op2 | *carry_on_left(Block2, Block1)* |
| situation | *relation(#A, #Table, on_table(#PosA))*<br>*relation(Block1, #A, right_on)*<br>*relation(Block2, #Table, on_table(Pos))*<br>*relation(#D, #Table, on_table(#Pos1))* |
| constraints | *140 <= Block1 · width <= 150*<br>*50 <= Block2 · width <= 60* |
| certainty factor | *1.000* |

**Figure 8.** Another example of a planning rule.

to top can be represented. PR-2 (Fig. 8) states that it is preferable to stack the right block first among two upper blocks. In this PR, variables and constraints are used as in Fig. 6.

## 4. UTILIZATION OF KNOWLEDGE

### *4.1. Description of a goal and its reduction to subgoals*

A goal is represented by a description of the state of environment to be established. For example, the goal state of task 2 is shown in Fig. 9. When the ACT of an operator is equivalent to part of the goal state description, that part is replaced by a subgoal which indicates execution of the operator. This operation continues until no further replacement is available and a part of the description which has never been replaced still remains. Each replaced subgoal is called a *goal node* (*GN*). This replacement is performed only once before execution and can never be retried. When there are multiple candidates for replacement, operators which can replace larger parts of the descriptions or operators whose preconditions are satisfied at the

```
relation(#A, #Table, on_table(#PosA))
relation(#B, #A, right_on)
relation(#C, #B, left_on)
relation(#D, #B, right_on)
```

**Figure 9.** Description of the goal state of task 2.

initial state are preferred. This replacement is also not affected by the result of replacement of other parts. In the case of task 2, all parts of the goal state description are replaced by four GNs to execute the following operators:

- *carry_on_table*(#*A*, #*PosA*),
- *carry_on_right*(#*B*, #*A*),
- *carry_on_left*(#*C*, #*B*),
- *carry_on_right*(#*D*, #*B*).

### 4.2. Ordering subgoals

When multiple GNs are generated, ARPEX-L sets orders between them using situation-independent PRs if possible. This operation is a rough planning before execution. Order information is stored in the *order link*. In the case of task 2, the orders shown in Fig. 10 are set. In this figure, a subgoal to carry block *X* is indicated as GN-X and order links are indicated by arrows. Of course, ARPEX-L cannot set the complete order if knowledge is insufficient.

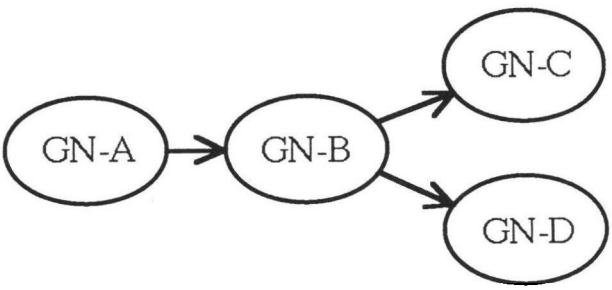



**Figure 10.** Goal nodes and orders in task 2.

### 4.3. Operator selection according to situations

ARPEX-L selects the next operator at each time when execution of the current operator finishes. The detailed procedure is as follows.

First, order links which start from already achieved GNs are deleted, if there are any. An unachieved GN to which no order link is set is called an *achievable GN (AGN)*. An AGN is a GN which is considered to be achieved if it is tried, since this GN has no need to wait for the achievement of other GNs. GN-A in Fig. 10 is an example of an AGN. Achieving AGNs one after another is the process of establishing the goal along the plan which is made before execution.

Then, by using PRs, ARPEX-L deletes AGNs, which is currently inappropriate to be tried by using PRs, from the set of AGNs. For example, let us consider the situation shown in Fig. 11 in task 2. In this situation, GN-A and GN-B have already been achieved and there is no order link. Therefore, both GN-C and GN-D are AGNs. If the PR shown in Fig. 8 is applied, ARPEX-L decides that achieving GN-D must come before achieving GN-C and deletes GN-C from the set of AGNs. In general, AGNs which are judged to be inappropriate in the current situation are

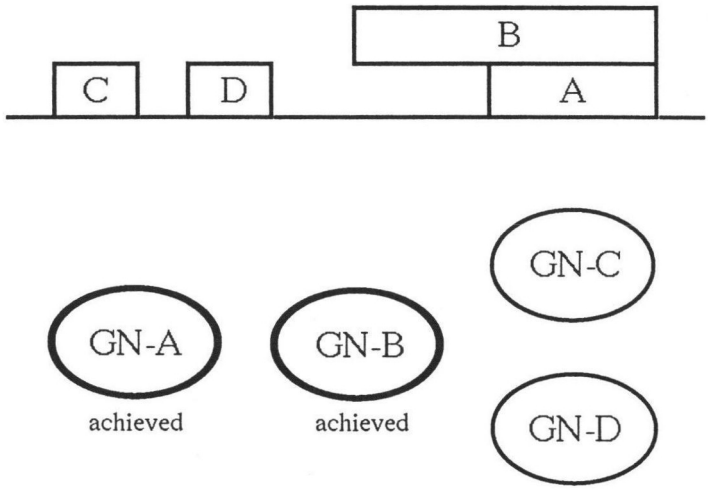



**Figure 11.** An intermediate situation in task 2.

deleted. One problem arises when multiple PRs compete with each other and both directions of order between two GNs are possible. In the case of competition between positive and negative PRs, the former is preferred because it is more reliable. When two negative PRs compete with each other, both GNs are deleted from the set of AGNs because both are inappropriate in the current situation. On the other hand, when two positive PRs compete with each other, ARPEX-L does no operation because both GNs are appropriate in the current situation.

Next, ARPEX-L collects operators which can be executed in the current situation by examining their preconditions. Collected operators are called executable operators. The following algorithm is used to decide on the next operator among selected ones:

*Step A:* If several executable operators match with some AGNs, select one of the operators at random. Let us take Fig. 11 as an example. In this situation, both GN-C and GN-D are AGNs unless there is some knowledge for setting an order between them. Moreover, *pick_up* and *carry_on_xxx* are executable for blocks *C* and *D*. Therefore, *carry_on_left*(#*C*, #*B*) and *carry_on_right*(#*D*, #*B*) become candidates and one of them is selected. If there is no matching pair of executable operators and AGNs, select the next operator in the following steps.

*Step B:* Select the next operator using SRs. First, ARPEX-L sets the evaluation value (*EV*) of each operator to zero (initial value). Then it collects SRs whose situation description matches with the current situation and whose *CF* (certainty factor) is larger than some threshold, and tries to apply them to every executable operator in turn. If an SR is applied to an operator, the *EV* of the operator is changed according to the following expression:

$$\text{In the case of positive SRs:} \quad EV = EV + CF,$$
$$\text{In the case of negative SRs:} \quad EV = EV - CF.$$

After every possible SR has been applied, the operator with the highest *EV* is

selected. If any SRs have never been applied, select the next operator in the following steps.

*Step C:* Select the next operator with ACTs. ARPEX-L selects the operator which can achieve as much of the remaining goal state description as possible. Some given function may be used to evaluate the degree of achievement. In pick-and-place tasks, this method is never used because the goal state description is completely replaced by GNs. Such preliminary replacement, however, can be regarded as preprocessing using this step. If no proper operator is found, select the next operator in the next step.

*Step D:* Select the next operator among executable operators at random. No knowledge is employed in this step of selection.

### *4.4. Execution of operators*

When a primitive operator is executed, the robot moves. When a macro operator is executed, GNs corresponding to lower level operators are generated and initial order links are set between GNs.

A primitive operator succeeds only when the states described in its ACT are established. A macro operator succeeds only when all lower level GNs are achieved. If failure occurs during the achievement of lower level GNs, the macro operator fails and all the generated GNs are deleted.

Execution of an operator may destroy states which other GNs have already established. In this case, ARPEX-L makes GNs which establishes destroyed states so that they can be retried, and establishes order links which had started from these GNs. For example, in Fig. 11, execution of the operator to carry block *C left_on B* destroys GN-B. In this case, GN-B becomes an AGN again and two order links, GN-B → GN-C and GN-B → GN-D, are established again.

## 5. LEARNING

ARPEX-L learns every time that execution of the current operator finishes, if possible.

### *5.1. Learning of operator selection rules*

ARPEX-L examines the result of execution and classifies the previous operator selection into three cases as follows:

Case 1: The selected operator failed.
Case 2: The selected operator succeeded, but it is judged to be an undesirable operator by heuristics.
Case 3: The selected operator succeeded and is judged to be a desirable operator by heuristics.

Using this classification, learning of SRs which were used in the selection is carried out as follows. In cases 1 and 2, operator selection is judged to be undesirable, and the *CF*s of positive SRs are decreased and those of negative SRs are increased. Oppositely, in case 3, operator selection is judged to be desirable, and the *CF*s of

positive SRs are increased and those of negative SRs are decreased. By considering the contribution of each SR to this selection, the amount of change of its CF can be calculated. Suppose that $n$ SRs are applied in the selection and the $CF$ of the $i$th SR is $CF_i$. The contribution of the $i$th SR, $value_i$ is calculated by the following expression:

$$value_i = K \times \frac{CF_i}{\sum_{i=1}^{n} CF_i},$$

where $K$ is a coefficient whose value is 0–1. When increasing a $CF$, ARPEX-L uses the following expression:

$$NewCF_i = OldCF_i + (1 - OldCF_i) \times value_i.$$

When decreasing a $CF$, the following expression is used:

$$NewCF_i = OldCF_i - OldCF_i \times value_i.$$

ARPEX-L uses the following two heuristics:

(1) An operator which makes a GN achievable (AGN) is desirable if that GN was neither achieved nor achievable.
(2) An operator which makes another operator executable is desirable if the latter operator can match with some AGN.

ARPEX-L achieves the given goal by achieving GNs which are generated before execution one after another. There are three states of GNs. The first one is the state where a GN is waiting for other goals to be achieved. The second one is the state where an AGN is waiting for the operator described in the AGN to be executed. The third one is the state where the corresponding operator is executed. So, there are two transition steps of GNs: from the first state to the second state, and from the second state to the third state. The heuristics mentioned above states that operators which make GNs move these steps are preferable.

When no SR is applied in the previous operator selection and there are appropriate SRs for generalization, these SRs are generalized (see Section 5.5). Otherwise, a new SR is created. ARPEX-L creates a negative SR in cases 1 and 2, and creates a positive SR in case 3. Its initial $CF$ is high in case 1 and low in cases 2 and 3, according to the reliability of the evaluation of operator selection. Figure 12 is an example of an acquired positive SR. This SR is knowledge that it is

| SR-2 | |
|---|---|
| type | *positive* |
| situation | *relation(#A, #Table, on_table(#PosA))*<br>*relation(#B, #A, center_on)*<br>*relation(#C, #Table, on_table(#Pos1))*<br>*relation(#D, #Table, on_table(#Pos2))*<br>*relation(#E, #Table, on_table(#Pos3))* |
| operator | *carry_on_center(#E, #A)* |
| constraints | *nil* |
| certainty factor | *0.300* |

**Figure 12.** An operator selection rule acquired in task 3.

preferable to carry block *E center_on* block *B* as a counter-weight for stacking blocks *C* and *D*. After ARPEX-L had actually executed the designated operator in the designated situation and this operator had been judged to be preferable in that situation, this SR was acquired.

ARPEX-L can represent uncertain knowledge with *CF*s and can control its reliability by modifying *CF*s. When an operator which had been selected by an SR fails, ARPEX-L does not deny that SR but reduces its reliability by decreasing its *CF*. If that failure is rare, its *CF* increases again after successful execution. If that operator is apt to fail in that situation, its *CF* decreases further and the applied SR will never be used in that situation. Competition of several SRs does not matter because their *CF*s become empirically appropriate values. The time of applying SRs is almost proportional to the number of SRs. This time, however, can be reduced by parallel processing.

### 5.2. *Learning of planning rules*

The manner of learning of PRs is similar to that of learning of SRs. *CF*s of PRs, however, are always 1 and are never modified. When there are multiple executable operators which match with AGNs at Step A of operator selection (see Section 4.3), ARPEX-L records candidate executable operators. After execution of the operator, the previous operator selection is classified into three cases as described in Section 5.1. Then ARPEX-L creates a PR corresponding to each combination of the selected operator and other unselected operators. If the selected operator is judged to be desirable, positive PRs are created. Otherwise, negative PRs are created. For example, if *carry_on_left*(#*C*, #*B*) is selected and fails in Fig. 11, a negative PR such that it is not preferable to execute *carry_on_left*(#*C*, #*B*) before *carry_on_left*(#*D*, #*B*) in that situation is created.

### 5.3. *Learning of ACTs*

Among ACTs, ARPEX-L can only learn e-ACTs. ARPEX-L learns e-ACTs by examining changes of the situation by execution of operators. Since there is uncertainty in the motions, a change of the situation which was observed as many times as some threshold is accepted as an e-ACT.

### 5.4. *Learning of macro operators*

ARPEX-L acquires macro operators by analysing sequences of executed operators. We uses ACTs of operators to limit the generation of useless macro operators.

If a sequence of successful operators is executed as many times as some threshold, ARPEX-L regards the sequence as a candidate for a macro operator and calculates the ACT of the whole sequence. Since a macro operator represents a piece of work and ACTs represent the effects of operators, ARPEX-L decides whether the candidate is accepted as a macro operator or not by heuristics which consider ACTs. Currently, the following heuristic is used:

- A candidate whose ACT does not include a description such that the robot holds some block is accepted as a macro operator.

By using this heuristic, a macro operator which ends under carrying blocks is never generated. In general, we can use heuristics stating that ACTs of macro operators never include the description of some intermediate states, e.g. a bolt is in place but the nut is not on, the cover of a box is opened, and so on. It is one of the promising ways to use ACTs (i.e. semantics of operators) in order to limit the generation of useless macro operators.

The precondition of a newly acquired macro operator is set to the state just before execution of the corresponding operator sequence. An adequate name and arguments are also given to represent the operator in predicate form. Arguments are what appear in its ACT among all arguments of operators which form the macro operator. In other words, arguments which do not appear in its ACT are not important.

We give an example of the learning of a macro operator in task 1. The shortest sequence of achieving this task consists of the following three operators:

- *carry_on_table*(*#B, TempPosB*),
- *carry_on_table*(*#A, #PosA*),
- *carry_on_centre*(*#B, #A*),

where *#PosA* is the destination of block *A* and *#TempPosB* is a temporary position on the table for block *B*. Assume that this sequence of three operators becomes a candidate. Its ACT is shown in Fig. 5 and a macro operator is generated from the sequence because it is judged to be proper by the above-mentioned heuristics. There are four candidates of arguments: *#A*, *#B*, *#PosA*, and *#TempPosB*. ARPEX-L accepts the first three by referring to ACT (*#TempPosB* is not essential in this task) and the macro operator shown in Fig. 13 is acquired.

| MO-1 | |
|---|---|
| operator | *mo1(#A, #B, #PosA)* |
| precondition | *relation(#A, #Table, on_table(#Pos1))*<br>*relation(#B, #A, center_on)* |
| suboperators | *carry_on_table(#B, #TempPosB)*<br>*carry_on_table(#A, #PosA)*<br>*carry_on_center(#B, #A)* |

**Figure 13.** A macro operator acquired in task 1.

### *5.5. Learning by generalization*

ARPEX-L generalizes SRs, PRs, and macro operators to extend their applicable situation. ARPEX-L uses two types of generalization. One is variablization of constant terms and modification of their constraints. The other is deletion of terms in the conditions. In both generalizations, only syntactic information is used. We do not describe further details of the generalization algorithm here. SR-1 shown in Fig. 6 and PR-2 shown in Fig. 8 are examples of the result of generalization.

## 6. BASIC STRUCTURE OF ARPEX-L

We constructed ARPEX-L with ARP, which is an intelligent robot language developed by us [8]. ARPEX-L consists of several subsystems, as shown in Fig. 14. Their functions are as follows.

When ARPEX-L is given a goal, *Goal Manager* interprets it and generates GNs. Then *Planner* sets orders between GNs with PRs. After this initialization, ARPEX-L enters the ordinary execution loop. In this loop, *Goal Collector* collects AGNs and *Operator Collector* collects executable operators and they send this information to *Operator Selector*. *Operator Selector* decides on the next motion according to the algorithm described in Section 4.3. If the selected operator is primitive, *Operator Selector* sends it to *Robot Controller* to make the robot move. If the operator is macro, *Operator Selector* sends it to *Goal Manager* to generate lower level GNs. *Rule Manager* and *Operator Manager* acquire and manage knowledge. *Enviroment Model Manager* manages an environment model for the robot.

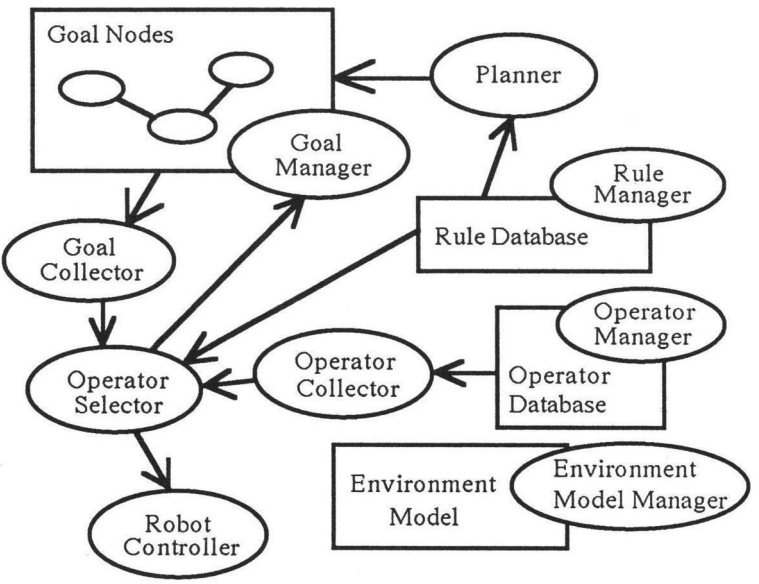



**Figure 14.** Constitution of ARPEX-L.

## 7. SIMULATION RESULTS OF THE PICK-AND-PLACE TASK

We applied ARPEX-L to three pick-and-place tasks as shown in Fig. 1. In each task, the system was given nine operators (listed in Table 1) and PRs such that the blocks were stacked from bottom to top as initial knowledge. Such knowledge is commonly utilized in all tasks. On the other hand, task-specific knowledge was acquired automatically by ARPEX-L. All tasks were carried out in simulation.

ARPEX-L accomplished task 1 by six steps (minimum number of steps) and

acquired one SR at the first trial. The number of steps is counted by the number of executed primitive operators.

In task 2, ARPEX-L needed 14 steps at the first trial and eight steps (minimum number) at the second. In these processes, two PRs and one SR were acquired.

In task 3, ARPEX-L had to use block *E* as a counter-weight to stack block *C* and block *D*. When the destination of block *E* was restricted to some place on the table, ARPEX-L could not achieve the goal because the problem space spread too widely. When we did not restrict the destination of block *E*, ARPEX-L accomplished task 3 by 26 steps and the final position of block *E* was *centre_on block B*. Four SRs including the one shown in Fig. 12 were acquired in this process. Using acquired knowledge, ARPEX-L achieved task 3 with restriction of the final position of block *E* by 26 steps at the first trial and by 12 steps (minimal steps) at the second trial. This example shows that the order of problems given to the system is important. That is, the system can acquire knowledge smoothly if easier problems are given before more difficult ones.

In each task, ARPEX-L acquired a macro operator to achieve the goal directly from the initial state by repeating the same task.

## 8. RESULTS OF THE PUSHING BLOCK TASK

We applied ARPEX-L to another task: a pushing block task. In this task, we used an actual manipulator. An overview of the task is shown in Fig. 15. There are a block and a hole on a flat table. The goal is to push the block into the hole. The state of the environment (i.e. the position and orientation of the block) is detected by the vision system with a single TV camera. The manipulator pushes the block and an ACT is acquired by detecting the effect of the motion by the vision system. ARPEX-L uses acquired ACTs to achieve the goal. Problems like this task, in

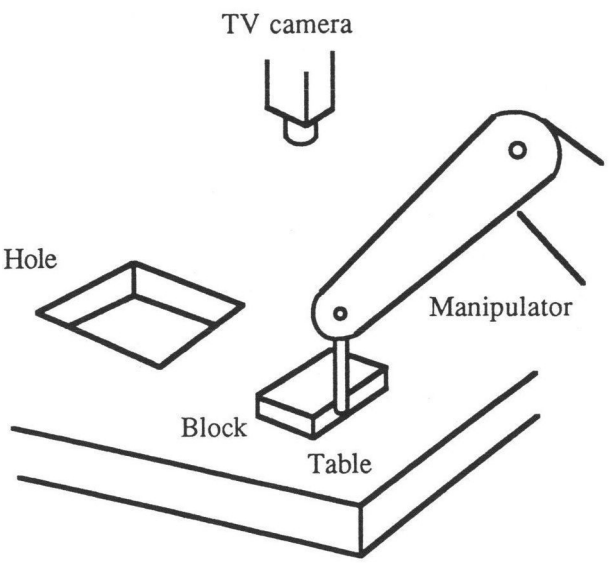



**Figure 15.** Overview of the pushing block task.

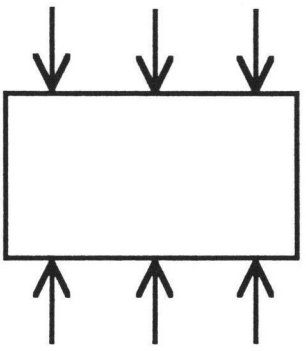


**Figure 16.** Six pushing positions.

which information about operators is incomplete, have been difficult for ordinary problem-solvers without learning ability.

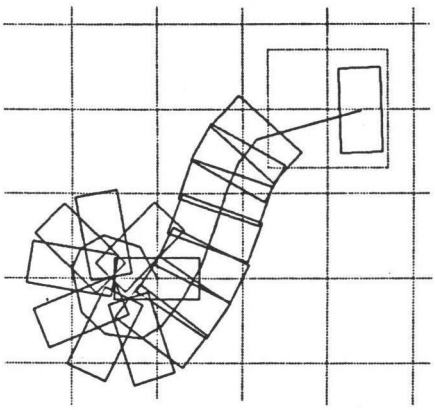
The shape of the block is rectangular and the pushing positions are limited to six, as shown in Fig. 16. Therefore six operators are initially given to the system. However, the ACTs of the operators are unknown, since it is very difficult to predict them because of the bias of the centre of mass, friction, slip, and so on.

Figure 17 shows the result of experiments with a block whose centre of mass was left-biased. In the figure, traces of the outline and the centre of the block are indicated. When selecting the next operator, the operator whose ACT is known and which is regarded to be able to reduce the distance to the hole most is chosen (see Section 4.3, Step C). If there is no appropriate operator, an operator whose ACT is unknown is selected at random (see Section 4.3, Step D) and its ACT is acquired. In this task, the situation is represented by the position and orientation of the block, and ACTs are represented by their changes. A function to evaluate the distance between the block and the hole is also given. When the centre of the block is pushed, the block does not move straight. To move the block in a straight line, the manipulator must push close to its centre of mass. Figure 17 shows that the more times the robot tries, the more appropriate operators are selected.

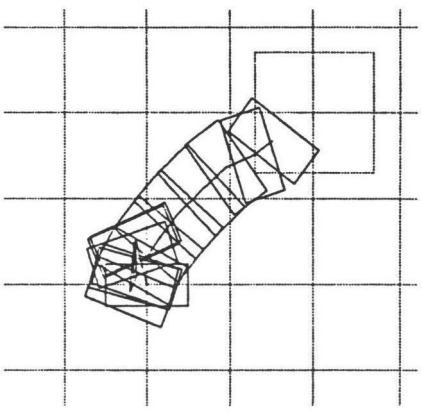
## 9. DISCUSSION

### *9.1. Pick-and-place tasks*

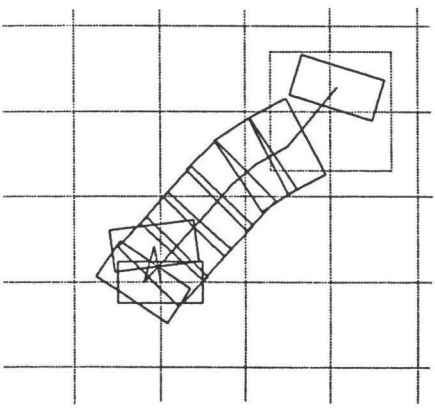
In pick-and-place tasks, ARPEX-L worked based upon initial knowledge and acquired SRs, PRs, and macro operators from experiences. Acquired knowledge was used for more efficient problem-solving. Only common knowledge for all tasks was initially given and knowledge specific to each task was not. Nevertheless, ARPEX-L accomplished each task without programming. This experiment is a simple example of automatic robot programming, since automatically acquired macro operators are programs for each task. As this experiment was performed by simulation and there is no uncertainty of the real world, we have not yet verified the validity of the management of uncertain knowledge by certainty factors. Certainty factors, however, were useful for managing competing SRs. Experiments with real tasks are to be carried out in the future.



first trial



second trial



third trial

**Figure 17.** Experimental results of the pushing block task.

### 9.2. Pushing block task

This task is very simple and can be carried out more certainly by employing non-symbolic (numerical) methods such as adaptive control. It is, however, important that this experiment is the first attempt to make a real robot learn by a symbolic framework, ARPEX-L. Currently, for simplicity, pushing positions are limited to six and the function to evaluate the distance between the block and the hole is given beforehand. In this task, ARPEX-L acquires and utilizes only ACTs and uses only a part of the whole functions. Application of ARPEX-L to more complicated problems will be investigated in the future.

Since symbolic approaches are indispensable for extension of intelligence to higher levels, more sophisticated methods to treat non-symbolic (numerical) data in symbolic frameworks are required.

### 9.3. Related work

To our knowledge, there has never been research on the integration of problem-solving and learning in the robotics field. We describe here the relation between our work and work in artificial intelligence and cognitive science.

Anzai [9, 10] proposed the concept of 'learning, understanding by doing'. He considered the human process of learning skills for operating ships from the cognitive point of view and simulated the process with a production system. In this research, he showed the following process: If the effects of operators are not given, they are acquired first to construct the problem space (understanding of the problem). Then, within that problem space, learning proceeds by doing and the problem-solving process becomes more efficient. ARPEX-L is an example of this concept as a problem-independent framework in the robotics field.

PROGIDY [2] by Minton and Carbonell acquires search control knowledge which is similar to positive and negative SRs and positive PR. One of the features of PROGIDY is the application of explanation-based specialization. Carbonell [3] has proposed a model of an integrated system of problem-solving and learning based on analogical reasoning. Both systems, however, are insufficient for robotics applications because they cannot treat operators whose effects are unknown and they cannot represent uncertain information.

SAGE [4] by Langley uses certainty factors for operator control rules and learns by modifying their values, as in the case of the SRs of ARPEX-L. SAGE augments rules only from successful experiences, while ARPEX-L learns also from experiences of failure. SAGE uses only one rule which has the highest utility at operator selection, while ARPEX-L considers all possible rules and modifies their certainty factors according to their contribution to selection. Since ARPEX-L considers multiple rules together even if they are competing with each other, a more appropriate operator will be selected. There are many ways to represent uncertain knowledge by using certainty factors [11]. Comparison of our method with other methods is also one of the future works.

Research by Nagata and Teramoto [12] uses orders between subgoals to plan. Their system automatically retrieves orders from information about operators. This approach, however, is not useful when information about operators is insufficient. ARPEX-L represents orders directly by PRs and can learn them from experiences. Our approach is more suitable for applications in the real world.

Learning of macro operators was first studied in MACROP of STRIPS [1]. STRIPS considers the dependences between variables by using triangle tables and acquires generalized macro operators. It is, however, necessary to limit the generation of useless macro operators. Minton [13] introduced two concepts, S-macro (a macro operator which is frequently used) and T-macro (a macro operator which is used for complicated problems), for efficient management of macro operators. Yamada and Tsuji [14] used the concept of perfect causality as a criterion for macro operator extraction. It is a feature point of learning of macro operators in ARPEX-L that semantic information (ACTs) is used for deciding whether a macro operator is useful or not. However, ARPEX-L needs domain-specific knowledge for such a decision.

### *9.4. Other future work*

- Replacement of the goal description by GNs before execution is not always preferable. It is desirable to delay the replacement until sufficient knowledge is available. For efficiency, we should be able to decide on the next several steps (operators) at a time using reliable knowledge.
- We should introduce other useful problem-solving methods. In particular, problem-solving based on analogical inference seems to be promising. However, we can consider that ARPEX-L partially employs analogical inference by SRs and PRs because these rules represent past experiences.
- We should introduce other useful learning methods. In particular, control of learning by knowledge like explanation-based learning seems to be promising. The generalization methods which ARPEX-L currently uses are based only on syntactic information. Acquired knowledge can be adapted to only limited problems. It is desirable to acquire more general knowledge by generalization using deep knowledge like physics rules or hierarchy of knowledge.

## 10. CONCLUSION

In this paper, we have described an intelligent robot system, ARPEX-L, in which problem-solving and learning are integrated, aiming at applications in the real world. We applied the system to two kinds of robot tasks: a pick-and-place task and a pushing block task. We studied its usefulness and its current defects. In the former task, we showed the possibility of automatic robot programming in the future. In the latter task, we were able to make a real robot learn by a symbolic framework.

*Acknowledgement*

We thank Professor Shirai at Osaka University for his useful comments.

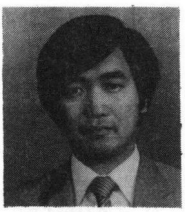
## REFERENCES

1. R. E. Fikes, P. Hart and N. J. Nilsson, "Learning and executing generalized robot plans," *Artif. Intell.*, vol. 3, pp. 251–288, 1972.

2. S. Minton and J. G. Carbonell, "Strategies for learning search control rules: an explanation-based approach", *Proc. 10th IJCAI*, pp. 228–235, 1987.
3. J. G. Carbonell, "Derivational analogy: a theory of reconstructive problem solving and expertise acquisition," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell and T. M. Mitchell, eds., Los Altos, CA: Morgan Kaufmann, 1986, vol. II, pp. 371–392.
4. P. Langley, "Learning effective search heuristics," *Proc. 8th IJCAI*, pp. 419–421, 1983.
5. T. M. Mitchell, "Learning and problem solving," *Proc. 8th IJCAI*, pp. 1139–1151, 1983.
6. R. J. Firby, "An investigation into reactive planning in complex domains," *Proc. AAAI-87*, Seattle, pp. 202–206, 1987.
7. M. P. Georgeff and A. L. Lansky, "Reactive reasoning and planning," *Proc. AAAI-87*, Seattle, pp. 677–682, 1987.
8. J. Miura, I. Shimoyama and H. Miura, "Development of programming language ARP for intelligent robots," *Trans. Jpn. Soc. Syst. Eng.*, vol. 11, no. 2, pp. 33–46, 1987 (in Japanese).
9. Y. Anzai, "Cognitive control of real-time event-driven systems," *Cognitive Sci.*, vol. 8, pp. 221–254, 1984.
10. Y. Anzai, "Doing, understanding, and learning in problem solving," in *Production System Models of Learning and Development*, D. Klahr, P. Langley and R. Neches, eds., MA: MIT Press, Cambridge, 1987, pp. 55–97.
11. K. Tanaka (ed.), *Knowledge Engineering*, pp. 187-199, Tokyo: Asakura, 1984 (in Japanese).
12. T. Nagata and Y. Teramoto, "A planning system utilizing meta knowledge," *J. Jpn. Soc. Artif. Intell.*, vol. 3, no. 2, pp. 186–195, 1988 (in Japanese).
13. S. Minton, "Selectively generalizing plans for problem-solving," *Proc. 9th IJCAI*, pp. 596–599, 1985.
14. S. Yamada and S. Tsuji, "Selective learning of macro-operators with perfect causality," *J. Jpn. Soc. Artif. Intell.*, vol. 4, no. 3, pp. 321–329, 1989 (in Japanese).

## ABOUT THE AUTHORS

**Jun Miura** was born in Kyoto, Japan, on 29 March 1962. He received a B.E. degree in mechanical engineering in 1984, and M.E. and Ph.D. degrees in information engineering in 1986 and 1989 respectively, all from the University of Tokyo. He is currently a research associate of the Department of Mechanical Engineering for Computer-Controlled Machinery at Osaka University. His research interests include robotics and artificial intelligence. He is a member of the Robotics Society of Japan, the Information Processing Society of Japan, and the Japanese Society for Artificial Intelligence.



**Isao Shimoyama** was born in Hyogo, Japan, on 6 January 1955. He received a B.E. degree in 1977, an M.E. degree in 1979, and a D.E. in 1982, all in mechanical engineering from the University of Tokyo. From 1982 to 1983 he was a lecturer, currently he is associate professor of Mechano-Informatics at the University of Tokyo. His research interests include microrobotics. He is a member of the Robotics Society of Japan, the Japan Society of Mechanical Engineers, and the Society of Instrument and Control Engineers.

**Hirofumi Miura** was born in Tokushima, Japan, on 17 March 1938. He received a B.E. degree in 1960, an M.E. degree in 1962, and a D.E. in 1965, all in mechanical engineering from the University of Tokyo. From 1965 to 1966 he was a lecturer and from 1966 to 1978 he was an associate professor. He is currently professor of Mechano-Informatics at the University of Tokyo. His research interests include robots in space and insect-model based microrobotics. He is a member of the Robotics Society of Japan, the Japan Society of Mechanical Engineers, the Operations Research Society of Japan, and the Japan Society of System Engineering.