Programming the Deliberation Cycle of Cognitive Robots

Mehdi Dastani[*] and **Frank de Boer** and **Frank Dignum** and
Wiebe van der Hoek and **Meindert Kroese** and **John-Jules Meyer**

Institute of Information and Computing Sciences
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

Abstract

This paper presents an overview of ongoing research that aims to develop a programming language for high level control of cognitive robots and software agents. The language is called 3APL and its formal specification is already presented in (Hindriks *et al.* 1999). We explain 3APL programming constructs and its existing interpreter. We argue that a part of the deliberation cycle, which is fixed in the interpreter, should be programmable. We discuss a set of programming constructs that can be used to program some aspects of the deliberation cycle.

Introduction

For many realistic applications, a cognitive robot should have both reactive and deliberative behavior (Fischer, Müller, & Pischel 1994; Konolige 1997; Shanahan 1999; Giacomo, Lespérance, & Levesque 2000). The first type of behavior concerns the recognition of emergency situations in time and provides rapid responses whereas the second type of behavior concerns the planning of actions to achieve its long term goals. In order to implement a robot's deliberative behavior, its mental attitudes such as goals and beliefs as well as the interaction between these attitudes should be implemented. Issues related to the implementation of robots' mental attitudes can be considered as object-level concerns while issues related to the implementation of the interaction between mental attitudes form meta-level concerns.

To illustrate these levels, consider a cognitive mobile robot at rest. The goal of this robot is to transport boxes from a $source$ position to a different position, whenever there are boxes at the $source$ position. Another goal of the robot is to clean a room whenever it is not clean. The robot can achieve these goals in several ways. For example, the robot can transport boxes before cleaning the room, or vice versa, or transport one box and clean the room for a certain amount of time and repeat it again. The beliefs and goals of the robot form the object-level concerns while the strategy of how to achieve these goals is a meta-level concern. Of course, a robot strategy can be considered as an issue that should be incorporated in the robot's goals. But, we believe that doing so violates the basic programming principle called *separation of concerns*. Therefore, we aim to distinguish these two levels explicitly and develop programming constructs to implement these levels separately.

In this paper, we present our ongoing research on developing a programming language for cognitive robots. The programming language is called 3APL (Hindriks *et al.* 1999) and it is designed to implement the deliberative behavior of cognitive robots. At this stage, the 3APL programming language provides a set of programming constructs to implement robots' mental attitudes such as its beliefs, goals, basic actions, and revision rules. The interaction between these entities are not programmable yet and is fixed by the 3APL interpreter. This interpreter determines the flow of control through a so-called deliberation cycle. In each cycle, a subset of goals is revised, selected and executed. As the deliberation cycle is fixed by the interpreter, the meta-level concerns can only be implemented to the extent that they are implementable indirectly by the set of object-level programming constructs and given that the programmers know how the flow of control is implemented in the 3APL interpreter. The aim of our research is to extend the 3APL language with a second set of programming constructs by means of which the deliberation cycle becomes programmable. This extension should make 3APL a programming language for cognitive robots that respects the separation of concerns principle.

This paper is organized as follows. First, we present the architecture of cognitive robots that can be programmed by 3APL. The syntax and semantics of the 3APL programming language is explained. An example of a 3APL program is proposed to implement the above mobile robot scenario. Subsequently, we explain the deliberation cycle of the 3APL interpreter and discuss the needs for programming constructs to implement the deliberation cycle. Finally, we indicate some future research directions and conclude the paper.

An Architecture for Cognitive Robots

The cognitive robot or software agent that we consider has a mental state consisting of beliefs, goals, basic actions, a set of practical reasoning rules for revising goals, and an interpreter. The beliefs represent the robot's general world knowledge as well as its knowledge about the surrounding

[*]Contact author is Mehdi Dastani: email: mehdi@cs.uu.nl. More information on 3APL can be found at: http://www.cs.uu.nl/3apl/

environment. In contrast to the BDI tradition (Cohen & Levesque 1990; Rao & Georgeff 1991), where goals are considered as states to be reached by the robot, and like ConGolog (Giacomo, Lespérance, & Levesque 2000), the goals are considered here to be the robot's control program that specifies its activities. The basic actions are the actions that the robot can perform. These actions may be cognitive actions such as belief updates or external actions such as moving in a certain direction. In general, a basic action can be considered as a wrapper that provides a parameterized interface to other processes. The set of practical reasoning rules contains rules that can be applied to revise actions that are blocked, goals that are not achievable, to optimize goals, or to generate some sort of reactive behavior. Finally, the interpreter determines the flow of control among the abovementioned ingredients. The interpreter is usually specified by a deliberation cycle (Georgeff & Lansky 1987; Giacomo, Lespérance, & Levesque 2000). This cognitive architecture, which we call the 3APL architecture, is illustrated in Figure 1.

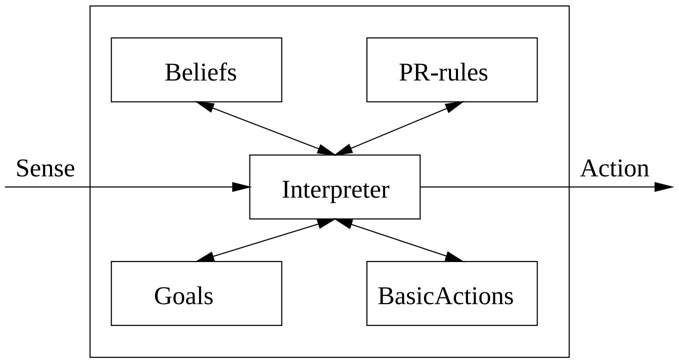

Figure 1: *The 3APL architecture.*

Although the 3APL architecture has many similarities with other cognitive architectures such as PRS, proposed for the BDI agents (Rao & Georgeff 1995; Georgeff & Lansky 1987), they differ from each other in many ways. For example, the PRS architecture is designed to plan agents' goals (desires) while the 3APL architecture is designed to control and revise robots' control program. In fact, a goal in the 3APL architecture is a control program which can be compared to plans in the PRS architecture. Moreover, there is no ingredient in the PRS architecture that corresponds to the practical reasoning rules which is a powerful mechanism to revise mental attitudes. Finally, the deliberation cycle in 3APL will become a programmable component while the deliberation cycle in PRS is integrated in the interpreter.

3APL Programming Language

3APL (Hindriks *et al.* 1999) consists of languages for beliefs, basic actions, goals, and practical reasoning rules. A set of programming constructs has been introduced for each of these languages. A 3APL program consists of four modules identified by the following key words: `BELIEF BASE, BASIC ACTIONS, GOAL BASE, PRACTICAL REASONING RULES`. Each of these modules consists of its corresponding programming constructs. Below is an overview of the programming constructs for each module.

Definition 1 (Programming Constructs for Beliefs) *Given a set of domain variables and functions, the set of domain terms is defined as usual. Let $t_1, \ldots, t_n$ be terms referring to domain elements and $Pred$ be a set of domain predicates, then the set of programming constructs for belief formula, BF, is defined as follows:*

- $p(t_1, \ldots, t_n) \in BF$.
- *if* $\phi, \psi \in BF$, *then* $\neg\phi, \phi \wedge \psi \in BF$.

All variables in the BF formula are universally quantified with maximum scope. The belief base module of a cognitive robot is the following programming construct:
`BELIEF BASE=` $\{\phi \mid \phi \in BF\}$.

The set of basic actions is a set of (parameterized) actions that can be executed if certain preconditions hold. After execution of an action certain postconditions must hold. These actions can be, for example, motor actions or belief update operations.

Definition 2 (Programming Constructs for Basic Actions) *Let $\alpha(\overline{X})$ be an action name with parameters $\overline{X}$ and $\phi, \psi \in BF$. Then, programming constructs for basic actions have the form:* $\{\phi\}\ \alpha(\overline{X})\ \{\psi\}$
The basic action module of a cognitive robot is the following programming construct:
`BASIC ACTIONS=` $\{\ C \mid C$ *is a basic action* $\}$.

The set of goal programming constructs that we use here differs from its specification (Hindriks *et al.* 1999) in some aspects. At this stage, we do not propose programming constructs for the choice and parallel operators. The absence of the choice operator implies that 3APL goals are deterministic programs and the absence of the parallel operator implies that no two activities can take place simultaneously. However, two new programming constructs are specified, i.e. `IF-THEN-ELSE` and `WHILE-DO`. Note that these programming constructs are specific uses of the choice operator and recursion, respectively.

Definition 3 (Goal Programming constructs) *Let BA be a set of basic actions, BF be a set of belief sentences, $\pi, \pi_1, \ldots, \pi_m \in GOAL$ and $\phi \in BF$. Then, the set of programming constructs for 3APL goals ($GOAL$) can be defined as follows:*

- *BactionGoal:* $BA \subseteq GOAL$
- *PredGoal:* $BF \subseteq GOAL$
- *TestGoal: if* $\phi \in BF$, *then* $\phi? \in GOAL$
- *SkipGoal:* `skip` $\in GOAL$
- *SequenceGoal: if* $\pi_1, \ldots, \pi_n \in GOAL$, *then* π_1 `;` ... `;` $\pi_n \in GOAL$
- *IfGoal:* `IF` ϕ `THEN` π_1 `ELSE` $\pi_2 \in GOAL$
- *WhileGoal:* `WHILE` ϕ `DO` $\pi \in GOAL$.

The goal base module of a cognitive robot is then the following programming construct:
`GOAL BASE =` $\{\ \pi_i \mid \pi_i \in GOAL\ \}$.

Practical reasoning rules are defined for various reasons. They can be applied to revise robots' goals that are not achievable, to actions that are blocked, to optimize robots' goals, or to generate some sort of reactive behavior. Before we define programming constructs for practical reasoning rules, a set of goal variables, $GVAR$, is introduced. First we extend the set $GOAL$ to includes goals which contain goal variables. The resulting set $VGOAL$ is defined by extending the definition of $GOAL$ with the following clause: if $X \in GVAR$, then $X \in VGOAL$.

Definition 4 (Practical Reasoning Programming constructs) *Let $\pi_h, \pi_b \in VGOAL$ and $\phi \in BF$, then a practical reasoning rule is defined as follows: $\pi_h \leftarrow \phi \mid \pi_b$.*
This practical reasoning rule can be read as follows: if the robot's goal is π_h and the robot believes ϕ, then π_h can be replaced by π_b.
The practical reasoning module of a cognitive robot is then the following programming construct:
`PRACTICAL REASONING RULES=` $\{$ *P | P is a practical reasoning rule* $\}$.

For example, the following practical reasoning rule generates reactive behavior since it generates the $repair()$ goal whenever it believes that an error has occurred.
$\leftarrow event(error) \mid repair()$

Example of a 3APL Robot Program

The example of the mobile robot, mentioned in the introduction, can be implemented in 3APL as follows. The beliefs of the robot are determined by the following statements:

$pos(x)$: current position is $x \in \{source, room\}$.
$box(source)$: there are boxes at the $source$ position.
$clean(room)$: the $room$ is clean.
$busy(self)$: the robot is performing a task.

The goal of the robot consists of two goals statements (i.e. $transport()$ and $cleanroom()$) and the following basic action statements:

$Goto(x)$: go to the position $x \in \{source, room\}$.
$DoTransport()$: transport box.
$DoClean()$: clean room.

Given these statements of the belief and goal languages, the following program implements the mobile robot.

```
BELIEF BASE= { ¬busy(self) }

BASIC ACTIONS= {
  {¬busy(self)}
      Goto(source)
  {busy(self) ∧ pos(source)},

  {¬busy(self)}
      Goto(room)
  {busy(self) ∧ pos(room)},

  {pos(source)}
      DoTransport()
  {¬busy(self)},

  {pos(room)}
      DoClean()
      {¬busy(self)}
}

GOAL BASE= {
  transport(), cleanroom()
}

PRACTICAL REASONING RULES= {
  transport() ← box(source) |
      Goto(source); DoTransport(); transport(),

  cleanroom() ← ¬clean(room) |
      Goto(room); DoClean(); cleanroom(),
}
```

The statement in the belief base module states that the robot is initially not busy with a task. The first statement in the basic action module states that the robot can only go to the source position if it is not busy with a task and that after going to the source position, it will be busy at the source position. The other statements in the basic actions module should be read in the same way. The statement in the goal base module states that the robot has two goals: transport boxes and clean the room. Finally, the first statement in the practical reasoning module states that if the robot's goal is to transport boxes and if it believes that there are boxes at the source position, then its goal should be revised into going to the source position, transporting a box and maintain the transport goal. The other statements in the practical reasoning rule module should be read in the same way.

3APL Semantics

In (Hindriks *et al.* 1999) an operational semantics for the 3APL language is proposed which is defined by means of a transition system. This semantics specifies transitions between the agent's states by means of transition rules. The state of an agent is defined as follows:

Definition 5 *The state of a 3APL agent is defined as a 4-tuple $< \Pi, \sigma, \Gamma, \theta >$, where Π is the set of the agent's goals, σ is the agent's beliefs, Γ is the set of practical reasoning rules, and θ is a substitution consisting of binding of variables that occur in the agent's beliefs and goals. (In the sequel we leave out Γ from the agent's states since this part does not change by transitions.)*

The binding of variables θ is generated and updated by transitions such as those responsible for the execution of test goals and the application of practical reasoning rules. The variable binding θ is passed through to other transitions. For the application of practical reasoning rules the guard of the rules that contain domain variables should be unified with a belief formula. Also, the heads of the rules containing both domain variables and goal variables should be unified with the goal-base. Both of these unifications result in substitutions which need to be passed to the next states. The set Π is a set of goals from which one goal can be executed. The transition rule on the agent level specifies the execution of a goal from Π.

Definition 6 *Let* $\Pi = \{\pi_0, \ldots, \pi_i, \ldots \pi_n\} \subseteq G$*, and* θ *and* θ' *be ground substitutions. Then,*

$$\frac{< \{\pi_i\}, \sigma, \theta > \rightarrow < \{\pi_i'\}, \sigma', \theta' >}{< \{\pi_0, \ldots, \pi_i, \ldots \pi_n\}, \sigma, \theta > \rightarrow < \{\pi_0, \ldots, \pi_i', \ldots \pi_n\}, \sigma', \theta' >}$$

The above transition rule states that transitions of sets of goals can be accomplished by transitions on each goal separately. Thus, the agent develops by transitions and revision of each goal in its goal-base. The application of practical reasoning rules to the goal-base may introduce new occurrences of existing variables in the goal-base and therefore may cause implicit unwanted bindings between variables. These bindings can be prevented by assuming fresh variable names which replace all variables that occur in a practical reasoning rules before they are applied to the goal-base. For more details on variable bindings see (Hindriks *et al.* 1999). Additional transition rules are given for the application of practical reasoning rules on a goal or the execution of a goal when it consists of a basic action. The transition rule for basic actions is formulated as follows.

Definition 7 *Let* $A(\overrightarrow{t})$ *be a basic action with a list of parameters* $\overrightarrow{t}$ *and* τ *be an update function that specifies the effect of the basic action on beliefs. The semantics of basic actions is captured by the following transition rule.*

$$\frac{\tau(A(\overrightarrow{t})\theta, \sigma) = \sigma'}{< \{A(\overrightarrow{t})\}, \sigma, \theta > \rightarrow < \emptyset, \sigma', \theta >}$$

The complete set of transition rules can be found in (Hindriks *et al.* 1999).

The 3APL Interpreter

In order to execute a 3APL program an interpreter is provided that specifies the flow of control among different modules of the program. The interpreter implements a deliberation cycle through which practical reasoning rule statements and goal statements are selected and subsequently applied and executed. This deliberation cycle is illustrated in Figure 2.

This deliberation cycle consists of two phases: a goal revision and a goal execution phase. In the goal revision phase (the upper part of the deliberation cycle in Figure 2), a subset of practical reasoning rules is selected on the basis of their applicability to goals and beliefs. Then, from this subset a practical reasoning rule is chosen and applied to the goal to which it is applicable. This application revises one of the robot's goals. In the goal execution phase (the lower part of the deliberation cycle in Figure 2), a subset of goals that can be executed is selected and from this subset a goal is chosen and executed.

The fact that this deliberation cycle is implemented in the interpreter implies that many deliberation choices are made

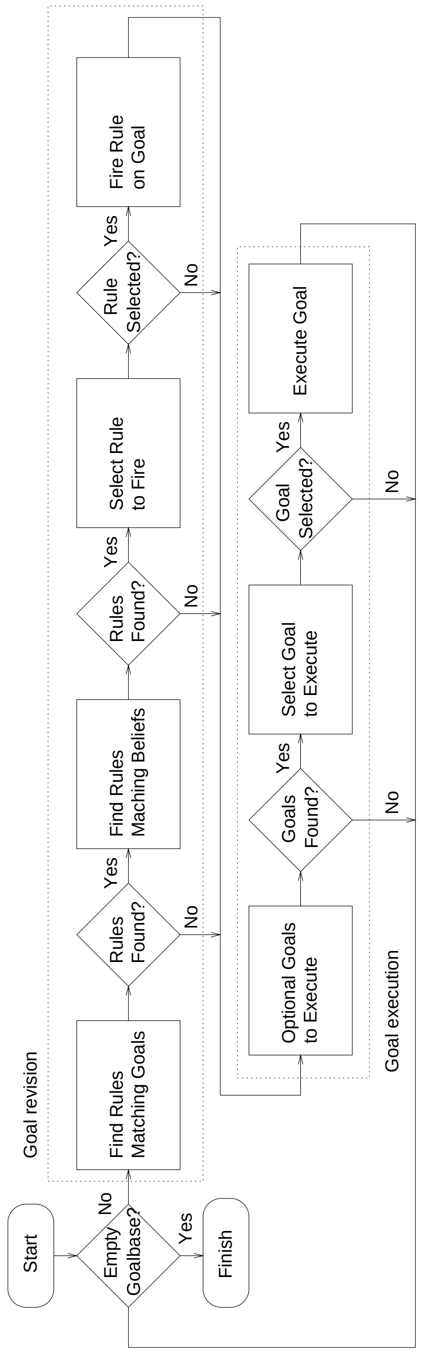

Figure 2: *The deliberation cycle of the 3APL interpreter.*

statically. For example, practical reasoning rules are chosen on the basis of their applicability to goals and beliefs. This implies that a programmer should design the practical reasoning rules in such a way as to ensure the selection of intended practical reasoning rules. In the example presented in the previous section and in situations where there are boxes at the source position and the room is not clean, the first two rules can be chosen. However, in a situation where the transportation of boxes is thought to be more important than the cleaning of the room, one may want to select the first practical reasoning rule repeatedly.

Of course, these types of strategies can implicitly be implemented by, for example, adding extra conditions to practical reasoning rules. For our robot, if we add $\neg box(source)$ to the conditions of the second practical reasoning rule, then the interpreter is forced to choose the first practical reasoning rule whenever there are boxes at the source position regardless of whether the room is clean or not. The drawback of such a solution is that additional conditions should be introduced and added to the practical reasoning rules. In addition to complexities, which are introduced by extra conditions to practical reasoning rules, the programmer is also forced to implement such strategies in an implicit way instead of programming them by explicit statements. This type of solution does not respect the programming principle called separation of concerns. The reader should note that similar problems exist for the selection of goals as well. In order to enforce the choice of a certain goal, the programmer should know the specific selection mechanism implemented in the interpreter.

Programming Module for the Deliberation Cycle

The specification of 3APL includes a (meta) language for programming some parts of the deliberation cycle. However, there are no programming constructs yet to implement those parts of the deliberation cycle. The aim of our research is to extend 3APL and introduce programming constructs to make the deliberation cycle a programmable module. The existing meta language for the deliberation cycle includes terms that refer to the set of goals and the set of practical reasoning rules that occur in their corresponding modules. These terms will then be used as arguments of programming constructs that implement the deliberation cycle.

Definition 8 *Let Π be a term that refers to the set of goals, and Γ be a term that refers to the set of practical reasoning rules. The set of terms that refer to goals T_g and practical reasoning rules T_r are defined as follows.*

- $\emptyset, \Pi \in T_g$ *and* $\emptyset, \Gamma \in T_r$.
- *if* $g_0, g_1 \in T_g$ *and* $r_0, r_1 \in \Gamma$, *then* $intersection(g_0, g_1), union(g_0, g_1), diff(g_0, g_1) \in T_g$ *and* $intersection(r_0, r_1), union(r_0, r_1), diff(r_0, r_1) \in T_r$.
- *if* $g_i \in T_g$ *and* $r_i \in T_r$, *then* $max_g(g_i) \in T_g$ *and* $max_r(r_i) \in T_r$.

This definition assumes an ordering among goals and practical reasoning rules. The max_g and max_r functions select respectively the goals and the practical reasoning rules that are highest in the ordering. In the sequel, we extend the set of functions that can select particular types of goals and practical reasoning rules.

A possible extension is based on the assumption that there are dependencies among goals in the sense that the selection of one goal gives rise to the selection of other goals. For example, suppose our robot has a third goal to clean another room close to the first room and far from the box positions. The selection of a cleaning goal gives rise to the selection of the second cleaning goal. For this type of extension, various relations between goals need to be represented and additional operations need to be defined. For example, for our robot one may define the dependency relation between goals (a set of goal pairs) and define an operator, e.g. $depend(g)$, which selects the set of possible goals that depend on g. Another useful function is one which identifies non-conflicting goals for a given goal. This function can be used to implement a strategy that selects and executes non-conflicting and coherent goals consecutively.

Similar extensions can be introduced for the practical reasoning rules. In general, practical reasoning rules can be classified according to the purpose of their use. They can, for example, be used to revise goals that are not achievable, to actions that are blocked, to plan goals, or to optimize goals. The types of practical reasoning rules can be used to select rules based on a certain strategy. For example, a strategy is to select rules that generate reactive behavior preferably, i.e. if more than one rule can be applied, select first the reactive rule. This makes a robot behave more reactively. Similarly, a preference on selecting rules that generate plans makes a robot more deliberative. Of course, a more refined classification of practical reasoning rules results in a variety of agent types. A different classification of rules can be defined based on their structural properties such as having a head, a body, or a condition. For example, a preference to apply rules without body makes the robot drop its goals preferably. To select certain types of practical reasoning rules we need to introduce additional function names to refer to various types of rules.

Definition 9 *Let F_g and F_r be sets of function names defined on goal terms and rule terms, respectively. The last clause of definition 8 can be generalized with the following two clauses:*

- *if* $g_i \in T_g$ *and* $f_g \in F_g$, *then* $f_g(g_i) \in T_g$.
- *if* $r_i \in T_r$ *and* $f_r \in F_r$, *then* $f_r(r_i) \in T_r$.

Given the terms denoting sets of goals and practical reasoning rules, the programming constructs for implementing the deliberation cycle are defined as follows:

Definition 10 *The set of meta statements S is defined as follows:*

1. *if* $G \in Var_g$ *and* $g \in T_g$, *then* $G := g \in S$
2. *if* $R \in Var_r$ *and* $r \in T_r$, *then* $R := r \in S$
3. *if* $g \in T_g, G \in Var_g$, *then* $selex(g, G) \in S$
4. *if* $G, G' \in Var_g$, *then* $ex(G, G') \in S$
5. *if* $r \in T_r, g \in T_g, R \in Var_r, G \in Var_g$, *then* $selap(r, g, R, G) \in S$

6. *if* $R \in Var_r$ *and* $G, G' \in Var_g$, *then* $apply(R, G, G') \in S$
7. *if* $g, g' \in T_g, r, r' \in T_r, \alpha, \beta \in S$ *and* $\phi \in BF \cup \{g = g', r = r', g \neq g', r \neq r'\}$, *then* α;β , IF ϕ THEN α ELSE β , WHILE ϕ DO $\alpha \in S$

The first two meta statements are designed to assign goals and rules to goal and rule variables, respectively. The statement $selex(g, G)$ (third clause) selects an executable goal from the set of goals g and assign it to G. The statement $selap(r, g, R, G)$ (fifth clause) selects a goal from g and a rule from r and assigns them to G and R, respectively. The choice to select a goal and a rule is based on a given ordering on goals and rules. Moreover, the selected rule should be applicable to the selected goal. The fourth statement executes goals G and the result is then assigned to G'. The sixth statements applies rules R to goals G and the result is assigned to G'. The seventh statement indicates the formation rules for composite statements.

The meta statements are in fact macros that can be used to implement some parts of the 3APL interpreter. For example, the first three boxes of the goal revision part of the 3APL deliberation cycle, illustrated in Figure 2, can be programmed by the action $selap(\Gamma, \Pi, R, G)$ and the fourth box can be programmed by the $apply(R, G, G')$ action. Moreover, the first two boxes of the the goal execution part of the 3APL deliberation cycle can be programmed by the action $selex(\Pi, G)$ and the last box can be programmed by the $ex(G, G')$ action. The reason to introduce these macros is a design issue and the result of a tradeoff between simplicity and expressiveness. On the one hand the macros should make deliberation cycles easy to program and, on the other hand, they should enable the programmer to implement interesting choices. Note that these macros can be implemented using other programming constructs such as assignment statement to variables. For example, $selex(g, G)$ can be implemented as $G := f_{select}(g)$, for a given selection function f_{select}.

Let Π be the set of goals of a 3APL agent and Γ its set of practical reasoning rules. The deliberation cycle of the 3APL interpreter, illustrated in Figure 2, can thus be programmed by the following meta-program.

```
WHILE $\Pi \neq \emptyset$ DO
 BEGIN
    $selap(\Gamma, \Pi)$ ;
    IF $R \neq \emptyset$ THEN
        $apply(R, G, G')$ ;
    $selex(\Pi, G)$ ;
    IF $G \neq \emptyset$ THEN
     BEGIN
        $ex(G, G')$ ;
        $\Pi := union(diff(\Pi, \{G\}), \{G'\})$
     END
 END
```

The selection of goals and rules can be conditionalized on robots' beliefs or on comparison between terms that refer to sets of goals or sets of rules. For example, it is possible to program a strategy for our mobile robot which makes sure that the robot selects the transport goal as long as there are boxes at the source position. Let $\Pi = \{transport(); cleanroom()\}$ and f_{trans} be a function name that selects transport goals. Then, the following meta program implements the above strategy.

```
WHILE $box(source)$ DO
 BEGIN
    $selex(f\_trans(\Pi), G)$;
    $ex(G, G')$
    $\Pi := union(diff(\Pi, \{G\}), \{G'\})$
 END
```

Note that the same type of strategies can be used to enable our robot to select certain types of goals or rules whenever it believes it is in an emergency situation. For example, whenever there is an overload of boxes at the source position, the robot should not select the cleaning goal but keep transporting the boxes.

Conclusion

In this paper, we presented the ongoing research on extending 3APL with programming constructs by which the deliberation cycle of the interpreter can be implemented. 3APL is a programming language for implementing high level control of cognitive robots and software agents. This language is a combination of imperative and logic programming. The logic programming part is designed to implement the beliefs while the imperative programming part is designed to implement the control program of the robot. Moreover, additional programming constructs are provided to design practical reasoning rules and basic actions.

In contrast to other programming languages designed to implement high level control of cognitive robots, such as ConGolog, the 3APL goals are deterministic programs and does not allow the choice and parallel operator. Instead, deterministic goals can be revised by applying practical reasoning rules to those goals. The application of practical reasoning rules to deterministic goals is equivalent to allowing the choice operator in goals and using backtracking mechanism to meet the choices. Note that elsewhere (Hindriks, Lespérance, & Levesque 2000) it is proven that ConGolog can be embedded in 3APL.

At this moment there exists a partial implementation of the 3APL interpreter. We are now implementing 3APL as a JAVA class. This class has private attributes for each of the 3APL modules such as belief base, goal base, etc. This class has at least two methods, one method for creation of the 3APL object through which the private attributes are set, and one method for running the object. The latter method corresponds to the deliberation cycle of 3APL.

References

Cohen, P., and Levesque, H. 1990. Intention is choice with commitment. *Artificial Intelligence* 42:213–261.

Fischer, K.; Müller, J. P.; and Pischel, M. 1994. Unifying control in a layered agent architecture. Technical Report TM-94-05, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH Erwin-Schrödinger Strasse Postfach 2080, 67608 Kaiserslautern Germany.

Georgeff, M., and Lansky, A. 1987. Reactive reasoning and planning. In *In Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, 677–682.

Giacomo, G. D.; Lespérance, Y.; and Levesque, H. 2000. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence* 121(1–2):109–169.

Hindriks, K. V.; Boer, F. S. D.; der Hoek, W. V.; and Meyer, J.-J. C. 1999. Agent programming in 3apl. *Autonomous Agents and Multi-Agent Systems* 2(4):357–401.

Hindriks, K.; Lespérance, Y.; and Levesque, H. 2000. An embedding of congolog in 3APL. Technical Report UU-CS-2000-13, Department of Computer Science, University Utrecht.

Konolige, K. 1997. COLBERT: A language for reactive control in sapphira. In *KI - Kunstliche Intelligenz*, 31–52.

Rao, A., and Georgeff, M. 1991. Modeling rational agents within a BDI architecture. In *Proceedings of the KR91*.

Rao, A., and Georgeff, M. 1995. BDI agents: From theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*, 312–319.

Shanahan, M. 1999. Reinventing shakey. In Minker, J., ed., *Workshop on Logic-Based Artificial Intelligence*. College Park, Maryland: Computer Science Department, University of Maryland.