Automated Systems & Technologies
25-26 May 2015 • St. Petersburg, Russia

LOGIC PROGRAMMING APPROACH IN SITUATION-SPECIFIC MANUFACTURING CONTROL

Vladislav E. Kovalevsky, Vyacheslav P. Shkodyrev

Peter the Great St. Petersburg Polytechnic University,
Control systems and technologies department
Grazhdansky Pr. 28,
195220 St. Petersburg, Russia
e-mail address of lead author: danik@kovalevsky.ru

Abstract

In this work we describe development of an intelligent agent for situation control of single station in a chemical manufacturing model. We give a definition of a situation and describe how it can be implemented in terms of logic programming. The Prolog language is used for implementation of the agent. We show design and implementation of the agent and demonstrate its work on semi-natural model of a chemical manufacturing.

INTRODUCTION

The task of a situation control is important for many control and safety applications. In many cases this task includes need for identification of emergency events sequences and taking appropriate actions to avoid or stop progress of undesirable situations as well as elimination of bad consequences that occurred due to hitting in such situation. Situation control is tightly bound with control of complex or ill-formalized objects and situations.
Logic programming seems to be a convenient tool for development of intelligent agent that will handle the concept of state space and work in situation control paradigm. Program written in logic programming language contains facts, rules and queries. Using these components it is possible to describe a domain in which the agent should work, state space and rules for

transition between states. An agent that will be appropriately programmed in such a way should be able to plan sequence of actions that will lead to the successful accomplishment of control task.

INTELLIGENT AGENT

First of all let's give definition. In this work we follow the definition given by Wooldridge and Jennings[1], and assume intelligent agent to be a system that is situated in some environment and which able to work autonomously and perform actions that lead to achievement of goals that are given to this system. Figure 1 illustrates this interaction between the agent and his environment.

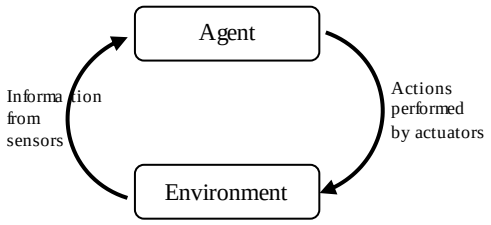

Figure 1. Interaction between agent and his environment

In vast majority of real domains the agent lacks full control on his environment. In best case the agent has partial control because it can perform actions that influence the environment. However the results of its actions cannot be fully predicted. Even worse, result of the same action can be different. This means that the agent should take in account possibility of failure.

Usually the agent has set of available actions. This set contains actions that the agent can perform to change the environment. However not all available actions are possible to execute in all states. In some states some particular actions may be prohibited. For example for a station that deals with pumping liquids from one tank to another action "drain the tank A" is only possible when tank A actually has liquid in it, and action "fill the tank B" is only possible when tank B is not full. It means that every action should have precondition that defines in which situation the action can be performed.

DEFINITION OF A SITUATION

Let’s give the following definition of a situation that the intelligent agent will use to perform actions that will lead to achievement of his goals. The situation S_i in some point time ‘i’ is a conjunction of states of all observable by the agent’s sensors parameters in the moment ‘i’ with conjunction with internal state of the agent itself which is defined by conjunction of states of its actuators.

Let's give the following definitions:

$S = \{S_1 \dots S_n\}$ – n possible states (situations).
$A = \{A_1 \dots A_m\}$ – m possible actions.
$S_i = d_i^1 \cap d_i^2 \cap \dots \cap d_i^v \cap f_i^1 \cap f_i^2 \cap \dots \cap f_i^w$ – situation in a moment 'i' for a system that contains v sensors and w actuators.
$S_0 \in S$ – start states.
$S_f \in S$ – goal states.

Using this definitions it can be said that the task of situation control for the agent is to change the initial state S_0 by performing possible actions A over the environment so that through the sequence of states $S_0 \overset{a_1}{\rightarrow} S_1 \overset{a_2}{\rightarrow} \dots \overset{a_f}{\rightarrow} S_f$ reach the goal state S_f. It should be noted that the state of the environment is non-deterministic because it depends not only on the actions of the agent but also on another factors. Thus the agent should continually adjust its course of actions and take in account changing situations. The goal of the control is this case can be either achievement of some desirable state or avoidance of some undesirable state.
It has been already mentioned that we regard the situation to be conjunction of states of all sensors and actuators of the agent. In the simplest case when the system has v sensors and w actuators and they are Boolean (i.e. gets only two states, for example: pump is on/off, tank is full/not full) the total number of possible situations will be 2^{v+w}. Thus introduction of new sensors or actuators to the system leads to exponential growth of the state space. In relatively small system which contains only 5 sensors and 5 actuators it will be already more than thousand possible states. It makes it difficult to supply beforehand the system with all transition rules that will describe transition from one state to another. Therefore it would be better to develop such an intelligent agent that given information about current situation and goal to be achieved will generate plan of actions without knowing all possible state transitions beforehand.

PROLOG LANGUAGE

Prolog (PROgramming in LOGic) is a logic programming language which is based on the language of mathematical logic that uses Horn clause, which is a subset of first-order logic. Its syntax is the syntax of formulas of first-order predicate calculus, recorded in conjunctive normal form in which the quantifiers are not written explicitly.
The language is focused around small set of basic mechanisms which include pattern matching, treelike representation of data structures and automatic backtracking. It is well suited for tasks that deal with structured objects that

have relations between them. Thanks to its features, Prolog is widely used in the field of artificial intelligence[2], computational linguistics, and non-numeric programming. In some cases, implementation of symbolic computations using standard programming languages calls into being large amount of hardly understandable code, whereas implementation of these algorithms in Prolog gives small comprehensible programs. One of the promising areas of the language usage is implementation of the concept of intelligent agents.

SITUATION CONTROL IN TERMS OF PROLOG

Now we have to formulate the problem of the situation control in terms of the chosen programming language, i.e. Prolog. For purpose of situations description in Prolog we can use special language construction named functor. Functor defines compound object and consists of its name after which in round brackets go its arguments. For example if we have system with 3 sensors and 3 actuators then set of situations can be defined by the following functor:

```
state(Sens1, Sens2, Sens3, Act1, Act2, Act3)        (1)
```

where concrete values of variables Sens1, Sens2, Sens3, Act1, Act2, Act3 indicate some concrete situation. It should be noted that Prolog variables are written with a capital letter and concrete values are written in small letters.
Set of actions that transfer the system from one state to another can be defined by triple functor ‘action(State1, Action, State2)’, where State1 stands for the state before the action is executed, State2 stands for the state in which the system will be after the action is executed and Action stands for the action to be executed. The states State1 and State2 themselves are written using above-stated functor ‘state’.
Now the previous example of the system which has 3 sensors and 3 actuators can be extended by introduction of some concrete actions. For example we can describe action ‘open_act3’ which opens valve3 (transfers it from state ‘act3_closed’ to the state ‘act3_open’) by the following functor:

```
action(state(Sens1, Sens2, Sens3, Act1, Act2, act3_closed),
                    open_act3,                                  (2)
       state(Sens1, Sens2, Sens3, Act1, Act2, act3_open)).
```

Other actions that the agent can perform can be describes in the same way. It should be noted that not all variables in the functor above have got concrete values. It means that states of these variables don’t important for the action execution. They can be in any possible state. Concrete values of variables in functor ‘State1’ which describes the state before action execution define

precondition that must hold for the action to be executed. Partial concretization of the variables eliminates need of beforehand description of all possible states and transactions between them.

Now let's define algorithm that given the transaction rules in the form stated above will find sequence of actions that will transfer the system from initial state to the desired goal state. In other words we want algorithm that will generate a plan. The following recursive definition can be given:

1. If current state of the system and the goal state are identical then no action should be performed and the work is done.
2. If the current state of the system differs from the goal state then exists an action that transfers system from current state to some another state and from this another state exists path to the goal state.

Let's define now recursive function 'goToGoal' that implements this algorithm. First part of this recursive definition can be defined by the next fact in Prolog:

```
goToGoal(GoalState, GoalState, []).                    (3)
```

Square brackets in Prolog, like in many other programming languages, indicate array or list. In our case this is list of actions in the plan. Empty brackets in this definition mean that there is no action need to be performed if we already in the goal state.

Second, recursive part of our algorithm can be defined by the following Prolog rule:

```
goToGoal(StartState, GoalState, [Action|Plan]) :-
        action(StartState, Action, State2),            (4)
    goToGoal(State2, GoalState, Plan).
```

This rule says that if our goal is to transfer the system from some initial state 'StartState' to some goal state 'GoalState' then first action in sequence of actions that lead to this goal will be some action 'Action' that will transfer system from the current state 'StartState' to some intermediate state 'State2' from which exists path to the goal state, and this path is contained in list 'Plan'.

Now the only thing that remains to do is to send to the Prolog request to generate the plan by supplying it with information about the current state and the desired goal state and it should generate sequence of actions that should be done to perform this transition. This information we will describe using the stated above functor 'state'. For the previous example of system with three sensors and three actuators this request can looks as follow:

```
goToGoal(
state(Sens1, Sens2, Sens3, act1_off, act2_on, Act3),   (5)
state(Sens1, Sens2, Sens3, act1_on, act3_on, Act3),
Plan).
```

After execution of this request variable ‘Plan’ will contain list of actions that make the plan.

EXPERIMENTS AND RESULTS

A mixing station from FESTO MPS® PA system (fig. 2) was used for the purpose of testing. This station consists of 4 tanks: 3 tanks contain source liquids and 1 tank contains mix of them in some definite proportions which are defined by the technological process. For the current situation evaluation the station has set of sensors, and for actions execution it has set of actuators. List of these is given in table 1.

Table 1. Sensors and actuators of mixing station of the FESTO MPS PA system

Type	Description	Amount
Sensor	Float switch	4
Sensor	Capacitive proximity sensor	6
Sensor	Flow sensor – float (mechanical)	1
Sensor	Flow sensor – impeller (electrical)	1
Actuator	Proportioning pump	1
Actuator	Pump of downstream	1
Actuator	2-way ball valve with pneumatic drive	3

The programmable logic controller (PLC) SIMATIC S7-300 (CPU 313C) of the Siemens company controls work of the station.

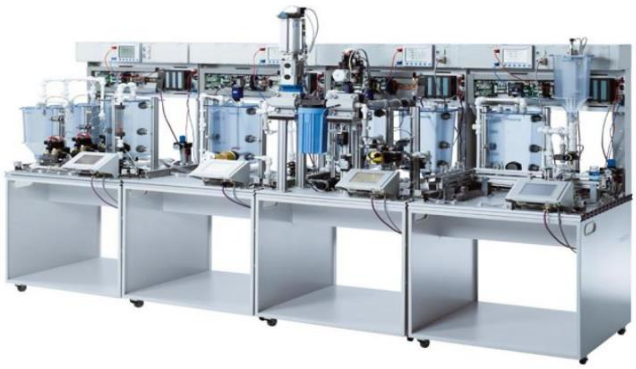

Figure 2. FESTO MPS® PA system

The PLC is used for control of the mixing station, but Prolog program that responsible for plan generation runs on personal computer (PC). So the information about situation at the station must be transferred from PLC to PC and generated plan must be transferred backwards. This data flow and interaction of the system with its environment are shown at figure 3.

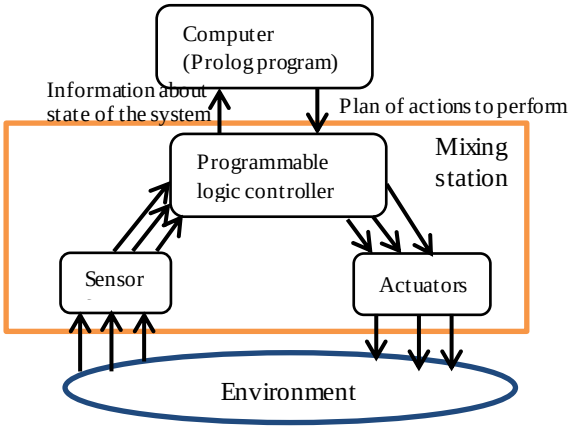

Figure 3. Control system of mixing station and its interaction with environment

Nowadays a lot of different implementations of Prolog language are exist. The comparison of the most popular implementations is given in table 2.

Table 2. Comparison of Prolog implementations

Name	Ciao	GNU Prolog	SICStus Prolog	SWI-Prolog	Visual Prolog
Operating system	Unix, Windows, Mac OS X	Unix,Windows, Mac OS X	Unix, Linux, Windows, Mac OS X	Unix, Linux, Windows, Mac OS X	Windows
License	GPL, LGPL	GPL, LGPL	Commercial	LGPL	Freeware, commercial
Compiled code	Yes	Yes	Yes	Yes	Yes
Unicode			Yes	Yes	Yes
Object oriented	Yes		Yes		Yes
Standalone Executable	Yes	Yes	Yes	Yes	Yes
C interface	Yes	Yes	Yes	Yes	Yes
Java interface	Yes		Yes	Yes	Yes
Interactive interpreter	Yes	Yes	Yes	Yes	Yes
Debugger	Yes	Yes	Yes	Yes	Yes
Code Profiler	Yes		Yes	Yes	Yes
Syntax	ISO-Prolog, plus extensions	ISO-Prolog	ISO-Prolog	ISO-Prolog, Edinburgh Prolog	Own

It can be seen from the comparison that SICStus Prolog, SWI-Prolog and Visual Prolog have most capabilities. But SICStus Prolog is a commercial product and Visual Prolog uses nonstandard syntax and also offers limited functionality in his free edition. Therefore it was decided to use SWI-Prolog[3], an open implementation of Prolog that includes a lot of different useful functions and libraries. For data transfer between Prolog and program in C# library SwiPlCs was used which implements an interface between Prolog

and C#. It was shown in [4] that libNoDave, a free communication library for Simatic S7, can be used to communicate between PLC and PC. For purpose of testing the latest version of the library was downloaded from the developer web page[5] (version 0.8.5) and based on it a user program was written in C# using IDE Visual Studio 2010.
The developed system works as follow: Data of sensors and actuator about the environment arrives to PLC. From the PLC this data arrives to a programming module in C# on PC. This module transfers the data to the Prolog module written in SWI-Prolog. From a user the module receives the goal to achieve. Using the received goal and information about the system state Prolog module generates plan, this plan is transferred to the C# module and from there to PLC for execution.

CONCLUSIONS

In this work was described development of intelligent agent using methods of logic programming, namely Prolog, for control of single station in a chemical manufacturing model. The principles of functioning of such an agent were shown and description of the situation control task in terms of Prolog was given. Testing of the suggested methods was performed using developed hardware and software stand.

REFERENCES

[1] Wooldridge, M. and Jennings, N. R. *Intelligent agents: theory and practice* // The Knowledge Engineering Review. 10(2). 1995. Pp. 115-152.

[2] Ivan Bratko. *Prolog Programming for Artificial Intelligence (4th Edition)* (International Computer Science Series), Pearson Education Canada 2011

[3] SWI-Prolog - a free implementation of the programming language Prolog. URL: http://www.swi-prolog.org (date of access: 11.05.2015)

[4] Kovalevsky V.E. *Usage of free software for purpose of data exchange between industrial controllers Simatic S7 and a personal computer* // Proceedings of interuniversity scientific and technical conference "Microsoft technologies in theory and practice of programming". – 25 March 2014, St.Petersburg. P.26-28.

[5] LibNoDave – a free communication library for Simatic S7. URL: http://libnodave.sourceforge.net/ (date of access: 11.05.2015)