

PSI Team: Planning Method and Main Principles of the Agents Architecture

PSI

Alexander Kozhushkin

AK: Program Systems Institute

Abstract. *The paper describes the PSI team developed at Program Systems Institute of Russian Academy of Science. We propose a dynamical refinement method of plans with defaults for the construction of software agents playing soccer. The architecture of our agents is composed of two main parts: the set of elementary plans and the planning system. The planning system uses on-line plan refinement procedure for constructing plans with elementary ones. There are some default conditions of normal continuation associated implicitly with each plan. The purpose of the refining process is to add new actions to the initial plan in order to maintain or recover normal continuation of the plan.*

1 Introduction

This paper is a description of the RoboCup team PSI, and the planning method that we use to construct our software agents (players). We propose a dynamical refinement method of plans with defaults for the agents construction. The architecture of an agent is composed of two main parts: the finite set of basic elementary plans and the planning system (planner).

Each elementary plan is designed to determine a skill or a role of the player or to solve a particular task. For example, there are plans designed to solve the task of the ball interception, overtaking another players and so on. More precisely, the elementary plan is just a triple $p = \langle B_p, C_p, \phi_p \rangle$ where B_p , C_p are some logical conditions on the information currently available for the player and ϕ_p is a controller, i.e. essentially the map from this information to the set of actions which the soccer server can perform. We call B_p and C_p , respectively, beginning and continuation conditions of the elementary plan p .

The player has a finite number of elementary plans called his *basic elementary plans*. The planning system uses not only basic elementary plans of the player. It may construct other elementary plans from basic ones by some simple rules of manipulation with beginning and continuation conditions.

The main purpose of the planner is to generate finite sequences of elementary plans, which are called *extended plans*. This is done by means of some simple rules (see below).

We assume, by default, that if C_p holds there is no obstacle which can prevent the normal continuation of the ϕ_p usage for every elementary plan p . For example, the plan for the ball interception is designed without any account of possible obstacles (another players) on the way of the agent. If C_p holds but from the "view point of the planner" some obstacle appears the planning system must temporarily postpone, or in our notation, interrupt the plan p in favor of some other elementary plan p' which can recover the normal continuation of the plan p and for which the condition $B_{p'}$ is satisfied. The plan p will be interrupted while $C_{p'}$ holds. At this time, p' may be also interrupted in favor of some other elementary plan p'' , etc. In other words, the planning system must refine the plan p by means of other elementary plans in order to solve the task of the plan p in spite of changes of the agent environment. The planner uses a mechanism of extended plans construction in order to determine which elementary plan can be interrupted by another one. The planning system is described more precisely in Section 2.

Let us present one simple illustrative example. Consider a player that has only three basic skills and, respectively, only three plans in the set of basic elementary plans. These are (i) the plan $p_1 = \langle C_1, C_1, \phi_1 \rangle$ of moving to the ball along a straight line and kicking the ball to the goal of the another team, (ii) the plan $p_2 = \langle \neg C_1, \neg C_1, \phi_2 \rangle$ of moving to some point on the field along a straight line and (iii) the plan $p_3 = \langle C_2 \wedge C_3, C_2, \phi_3 \rangle$ of another player overtaking.

Condition C_1 holds if the agent derives from available information that it can achieve the ball before the other team mates can do this. C_2 holds if there are other players in front of the agent. C_3 is satisfied if the speed of the agent is greater than 0. If the player uses at the current time the plan p_1 or p_2 and there is another player on its way the initial plan will be interrupted by the elementary plan $\langle C_1 \wedge C_2 \wedge C_3, C_1 \wedge C_2, \phi_3 \rangle$ or $\langle \neg C_1 \wedge C_2 \wedge C_3, \neg C_1 \wedge C_2, \phi_3 \rangle$ respectively. Both of these elementary plans are constructed by the planner on the base of the plan p_3 .

The rest of the paper is organized as follows. In Section 2, we make short description of the mathematical model of the planning method that we have used for the construction of our software agents. In Section 3, we briefly survey the agents architecture. In Section 4, we make some concluding remarks.

2 Model

Let us consider a model of planning system built in each player. It is defined by the discrete set of time moments $T = \{0, 1, 2, 3, \dots\}$, a set O of elementary actions which soccer server can execute, a set I of input (or external) states defined by all possible values of data available from soccer server and the space Ω of internal states of the player. $I \times T$ and O represent input information (*input*) and actions (*output*) of the player, respectively. Play history, or just a play, from the point of view of a player may be defined as a map $H : T \rightarrow I \times \Omega$, where $H(t)$ consists of the information from soccer

server and of the internal state of the player at the moment t . Let $B(i, \omega)$, $C(i, \omega)$ be some logical conditions on the set of full states $I \times \Omega$ and ϕ be a map $I \times \Omega \times T \rightarrow \Omega \times O$. Define *elementary plan* of the player as a triple $p = \langle B_p, C_p, \phi_p \rangle$, where B_p and C_p are, respectively, its beginning and continuation conditions.

For our work we use some fixed finite set of *basic elementary plans* π and its extension π^* . We build the set π^* by the following rules:

- (i) $\pi \subset \pi^*$;
- (ii) if $p = \langle B_p, C_p, \phi_p \rangle \in \pi^*$ then plan $\langle C_p, C_p, \phi_p \rangle$ is in π^* ;
- (iii) if $p = \langle B_p, C_p, \phi_p \rangle \in \pi^*$ and $p' = \langle B_{p'}, C_{p'}, \phi_{p'} \rangle \in \pi^*$ then plan $\langle B_p \wedge C_{p'}, C_p \wedge C_{p'}, \phi_p \rangle$ is in π^* .

We define the set of all *applicable* plans at the time moment t of the play H , i.e. elementary plans whose beginning or continuation conditions are satisfied at the moment t , as $S_t = S_t^H = \{p \in \pi^* \mid \models B_p(H(t)) \vee C_p(H(t))\}$.

We define *extended plan* P as a word of a plan language L in the alphabet $\pi^* \cup \{*\}$. L is defined by the next rules:

- (1) $*$ is plan;
- (2) if $Q*$ is a plan and $p \in \pi$, then $Q * p$ is plan;
- (3) if $Q_1 * p Q_2$ is a plan and $p \in \pi^*$ then $Q_1 p * Q_2$ is plan;
- (4) if $Q_1 * \langle B, C, \phi \rangle Q_2$ is a plan, where $\langle B, C, \phi \rangle \in \pi^*$, $\langle B', C', \phi' \rangle \in \pi$,

then $Q_1 \langle B, C, \phi \rangle * \langle B' \wedge C, C' \wedge C, \phi' \rangle \langle C, C, \phi \rangle Q_2$ is plan.

In (1) – (4) Q, Q_1, Q_2 are letters sequences without $*$, possibly empty.

Every extended plan represents some play history and possible evolution of the play history from the view point of the planning system. All letters before " $*$ " are the past of the planning system. These letters represent elementary plans, which the agent has used. The first elementary plan after " $*$ " is a current one, i.e. behaviour of the agent is determined by the ϕ map of this plan at present. All other elementary plans are those which the planning system is going to execute in the future.

The planning system, or the planner, constructs dynamically an extended plan from the initial one $[*]$ depending on the situation on the field. It uses the two following rules. The planner can concatenate an elementary plan from $S_t \cap \pi$ to the extended plan, if the extended plan has been exhausted (see rule 2 above). The planning system can temporarily interrupt some elementary plan in favour of another one from $S_t \cap \pi$ (see rule 4). In order to construct only correct extended planes, the planner uses *Int* and *Prior* relations, defined on the set of basic elementary plans π , and the family of sets $D_i \subset \pi$. The *Int* relation represents possible plan refinements, and *Prior* relations play the role of evaluation functions [1].

Divide the set π into plan levels, the family of sets $D_i, i \in \{0, 1, 2, \dots, n\}$, such that $\cup_i D_i = \pi$ and $D_i \cap D_j \neq D_i$ for $i < j$. D_i are not *necessarily disjoint* sets. By means of plan levels we define on the set π binary interruptability relation $Int(p, p')$. This relation can be defined arbitrarily, with the only constraint on it: $Int(p, p') \wedge p \in D_i \Rightarrow p' \in D_{i+1}$.

Let us consider some partial order relations $Prior_i$ of priority on respective D_i . Constraints on this are:

- (i) $Prior_0$ restricted on the set $D_0 \cap S_t$ is a linear order for each t ;
- (ii) For every plan p and every $i > 0$ the restriction of $Prior_i$ on $\{p' \in D_i | Int(p, p')\} \cap S_t$ is a linear order for each t .

Let us explain the meaning of these relations. Consider the D_i family. Every D_i represents some set of the basic elementary plans. Let an elementary plan $p \in D_i$ be under execution. Some situation can occur, that can disturb or prohibit the normal plan p continuation. In this case the planner interrupts the plan p in favour of some other plan $p' \in D_{i+1}$ to recover the normal plan p continuation. Roughly speaking, the planner refines the plan p by using the plan p' . The D_0 level plays a special role. Only plans in D_0 planner can concatenate to the exhausted extended plan.

Consider the Int relation. The meaning of $Int(p, p')$ is that the plan p can be interrupted by the plan p' for some time, if there exists the condition for interruption. This condition is the beginning condition of the plan p' , implicitly connected with the plan p by Int relation. All such conditions are defaults for p , i.e. defaults for some plan $p \in D_i$ are $\bigwedge_{p' \in \{p' | Int(p, p')\}} (\neg B_{p'})$. If some plan p is interrupted by another plan p' , then it will be interrupted while $\models_t C'$, where C' is continuation condition of the plan p' . After this plan p will continue its work. The meaning of $Prior_i$ is that if $Prior_i(p', p)$ for some plans $p, p' \in D_i$ then plan p' is preferable to p . $Prior_0$ is a linear order on $D_0 \cap S_t$ and the planner uses $Prior_0$ to determine what elementary plan $p \in D_0 \cap S_t$ it must be concatenated to the exhausted extended plan. Similarly, $Prior_i, i > 0$ is a linear order on $\{p' \in D_i | Int(p, p')\} \cap S_t$, and the planner uses $Prior_i$ to determine what elementary plan p' must interrupt the plan p .

3 Agents architecture

Let us consider the agent architecture. There are four basic modules. (1)**Interface module** receives information from the soccer server, (transforms it into a internal representation) and sends commands to the soccer server. (2)**Information processing module** obtains information that the agent can not to receive directly (for example, absolute coordinates, velocity of the player). It uses data from the interface module and information derived from a internal world model, i.e. set of facts concerning the size of field, maximal speed of palyers, etc. (3)**Set of basic elementary plans** is, exactly, the set π in our model. (4)**Planner** makes complete plans from elementary ones in correspondence with the basic relations of our model.

For a presentation of a complete plan state as a function of time, we use the stack of plans. Its elements contain information about elementary plans under execution, namely, plan name, number of a current plan step and information for other elementary plans. We enumerate stack elements from the bottom to the top of the stack, thus if stack contains n elements, then element e_{n-1} is on top of the stack. There are two constraints on the stack. First, if element e_i contains the name of some plan p , then $p \in D_i$. Secondly, there is no recursion, i.e. if two elements e_i and e_j contain the name of the same plan (at the same time), then $i = j$. At every time moment, each

elementary plan has one of three statuses. A plan is *active* iff the element on the top of the stack contains the name of this plan. A plan is *interrupted* iff there exists a not-top stack element that contains the name of this plan. Otherwise the plan is *passive*. From now, on brevity we will not distinguish between an elementary plan and the stack element, containing its name.

Every elementary plan is an independent module. It reads information from the interface, information processing module and from the stack. At every simulation step it uses these data to test beginning and continuation conditions and to tune plan variable parameters. In case the testing of its continuation conditions fails, the active plan removes itself from the stack, and interrupted plan removes all including itself up to the top of the stack. Otherwise plan (regardless of status) sends information about the possibility of its execution to the planner. All plans that can possibly be executed form the set S . In case the planner allowed the plan to execute, active or interrupted plan continues its work, but passive plan adds itself on top of the stack. Active plan can send command to the server by means of the interface module.

The planner uses the set S for its work. For each plan level D_i the planner searches S for a maximal plan p_{max} (in sense of $Prior_i$ relation), such that $Int(p', p_{max})$, where p' is the $(i - 1)$ -th stack element. If plan p_{max} is not contained in the stack as the i -th element, and $e_{(i-1)}$ is not the top of the stack, then planner removes all plans from the i -th level to the top from the stack. Otherwise planner does nothing. Then the planner activates the plan p_{max} .

Planner and all elementary plans are implemented as functions of C language. All modules are the same for all agents, the player identifies its role by the player number given from server. Behaviours according to the roles are due to special plans (Goal-Keeper, Defender, Attacker, etc.). Players can change their roles depending on the situation on the field.

4 Conclusion

Our approach is somewhat analogous to that presented in [2, 3, 4], with one essential difference: our planning system works *on-line*, and plans refinements are being made dynamically in case of need. More detailed comparisons deserve the further investigations and are to be presented elsewhere. The further development of the method itself and the more precise formulation of its essence is a goal of further work. At least the experiments show that our method is a powerful and flexible tool for construction of autonomous agents working in unpredictable environment.

References

- [1] T. Dean, S. Kambhampati. Planning and Scheduling. *CRC Handbook of Computer Science and Engineering* (1996).
- [2] M. Ginsberg. Modality and Interrupts. *Journal of Automated Reasoning* (14)1 (1995), pages 43–91.
- [3] M. Ginsberg 1995. Approximate Planning *Artificial Intelligence Journal* (76)1–2(1995), pages 89–123.

- [4] M. Ginsberg, H. Holbrook. What defaults can do that hierarchies can't. *Fundamental Informaticae* (21) (1994), pages 149–159.