A Modified Rete Match Algorithm for Predicate Management in Real Time Planning and Execution Systems

Marcelo Nicoletti Franchin

Electrical Engineering Department, Faculty of Engineering
UNESP – Sao Paulo State University
Av. Luis Edmundo Carrijo Coube 14-01
Bauru – Sao Paulo – CEP 17033-360 – Brazil
franchin@feb.unesp.br

Abstract. Automatic action planning for task accomplishment is one of the AI areas called *AI Planning*, where intense research occurred in the last decade. Among the many categories of planners and diverse problem domains, there are real time planners that consider external events and generate the plan interleaved with the execution system, processing real time collected data (i.e. sensory data) that have influences on the planning process. This paper presents a modified version of RETE match algorithm that is used to instantiate the parameters of prototype predicates and to verify if a predicate is true in a given world state. This modified version was implemented in a real time planner using $ALWAYS_{TRX}$ method. As a result, an efficient system for real time intelligent control was obtained and it is currently used in our research robots.

Keywords: RETE algorithm, AI/Planning, Artificial intelligence, Real time planning, Intelligent systems.

1 Introduction

Automatic action planning for task accomplishment is one of the Artificial Intelligence areas called *AI Planning*, where intense research occurred in the last decades. A great problem in automated reasoning is to design systems that can find automatically a sequence of actions (or a plan) that allow an agent to change the environment (called world) from an initial state to a desired state (goal state). This plan could be delivered, for example, to a manufacturing system, a robot, or any kind of actuator that, following the plan, execute the actions and make the world change to the desired goal state. In artificial intelligence terminology they can be called agents [1][2].

Early AI planning systems appeared in the sixties and at that time some assumptions were created to simplify the planning problem [3][4]. One of the

Industrial Informatics
Research in Computing Science 31, 2007, pp. 153-162

assumptions was considering the agent (a robot, a manufacturing system or some another form of actuator) the only entity responsible for changing of the world state. Another supposition of the planning system was the belief that the changes made in the world by the agent were totally deterministic.

At that time the planning problem was to create techniques and heuristics to search in a tree of possible plans to achieve a desired goal. The complexity of this problem was analyzed years later and considered NP-Complete and NP-Hard [5][6]. Nowadays the complexity was demonstrated to be much harder with variations [2][7].

Considering applications in complex environments with uncertainty and dynamic changes, easily found in real world, like manufacturing systems, mobile robots, software agents, inspection, maintenance, surveillance, etc., the changes in world state are made by multiple agents. If the planner considers that only its agent can change the world, the resulting action plan may not represent the reality and the execution of this plan would be incompatible with the world where the agent is inserted. Therefore, the planning system must use its perception resources to adequate the plan to the eventual contingencies during the plan execution.

Beyond the dynamics and unpredictability of the environment, the agent owns sensors and actuators that do not work in an ideal way and the time available for decisions is very limited. The knowledge base of the agent therefore is incomplete and the planning system must consider this aspect to make possible agent interaction in unstructured environments [8][9][10].

Nowadays, the planning systems evolved to deal with real world applications. A planning system with treatment of external events integrated to the execution is a problem of enormous complexity and demands techniques that optimizes performance of execution to avoid system overhead, otherwise its use will be impracticable in real world environments because response time constraints [11].

Amongst the inherent problems of the planning methods there is the parameter instantiation of predicate prototypes that consists in a combinatorial task that can take a considerable processing time. Moreover, the process of verifying if a predicate is true in the world state consists of comparing the predicate with each one of the predicates that describe the state of the world. The computational cost of both comparisons increases linearly with the amount of facts and exponentially with the amount of changeable parameters of a predicate.

This paper presents an adaptation of RETE match algorithm that was implemented in a real time planner using the method ALWAYS $_{TRX}$.

Section 2 presents the complexity of the problem and related work. Section 3 presents a summary of ALWAYS$_{TRX}$ method where the adaptation of RETE algorithm was used successfully. Section 4 presents the adaptations made in RETE algorithm with an example case study in the blocks world. Finally, in section 5, the conclusions and future works are presented, followed by the bibliographical references.

2 Problem Complexities and Related Work

Early work integrating planning and execution like STRIPS/PLANEX [12][13], Universal Plans [14], IPEM [15], BUMP [16], XII [9], SIPE-2/PRS-CL [17][3] e SAGE [11], as well early works in planning with incomplete information like UCPOP [18], BURIDAN and C-BURIDAN [19], among others, had been collaborated to the development of planning systems but partially solve the problem of plan in dynamic and unstructured environments because they do not consider, altogether, the aspects early presented. More detailed discussions can be seen in [20].

Research in planning systems that handle external events have been developed by our planning group since the eighties. Among the systems developed are: the PETRUS system [21], EXTEPS [22], PBE [23] and finally, the ALWAYS$_{TRX}$, in its early research stage being presented shortly in this paper. The ALWAYS$_{TRX}$ method [24][25] is an effort to create systems that can effectively plan in complex, dynamic and unstructured real world domains.

As cited previously, the problem of parameter instantiation of predicates and the process of logical verification of a predicate in a given state, both have a very high computational cost. Solutions have been found to make possible real time planning systems [26].

A similar problem to the described one in the previous paragraph is the work of find the match in the comparison of a set of patterns with a set of objects, made by the interpreters of rule production systems. They must determine which rules of production must be triggered through the comparison of its logical expressions (conditions to fire a rule) with the facts declared in the working memory.

A fast and efficient algorithm to execute this task is RETE algorithm [27]. RETE algorithm is, at least, many times more efficient than any another algorithm already developed for the task to compare objects with patterns [28][29][2].

Among the best algorithms that work with this kind of problem, there are TREAT [30] and GATOR [31]. The LANA algorithm [32] is a parallel version of RETE algorithm. An extension of RETE algorithm for the management of events with time information is presented in [33]. RETE algorithm still is the champion of efficiency being used in commercial and open systems as JESS (rule engine of the Java platform).

Therefore, RETE algorithm was chosen as a reference algorithm to be adapted to be used with parameter instantiation of predicates and in the verification and storage of predicates in a given world state, resulting in the performance enhancement of the ALWAYS real time planner.

3 ALWAYS$_{TRX}$ Method

The ALWAYS$_{TRX}$ method (ALWAYS Thinking, Reacting and eXecuting) [24][25] was designed to integrate the agent's planning task and execution task, including the external events treatment inside the planning phase, to give to the agent reactivity capabilities to the dynamic world changes in which it is inserted.

The integration approach requires that planning decisions followed by execution must be based in a common knowledge representation.

With ALWAYS$_{TRX}$ the planning system integration with the plan execution system considers that planning must be done concurrently with plan execution (the plan is made "on the fly"). The plan being created by the planner is represented by a tree where the nodes mean activities and events. Each path in the tree represents a sequence of activities with events that may occur. The execution module must follow one path in the tree. While the planner creates new paths, the execution module chooses one of these paths to execute, based on the actual occurrence of the external events associated with the chosen activities.

The planning system uses the information about which external events actually happened, obtained from the environment through the execution module, to adequate the paths to the real world state. As the execution module executes one step of the plan, the paths of the search tree not associated to the path that contains this step are excluded from the tree. One plan step means to observe the occurrence of an external event and initiate the execution of the correspondent activity. When a event occurs, the planner keep expanding paths below this event in the tree and exclude the paths that do not go through this event.

The ALWAYS$_{TRX}$ is domain independent and can be used in many engineering and automation applications such as intelligent manufacturing and robots control.

Fig. 1 presents how the method is inserted in the context "see, think, act" and its internal architecture that consists of two modules running concurrently: the planning module and the execution module. Both modules constantly communicate through a common knowledge base.

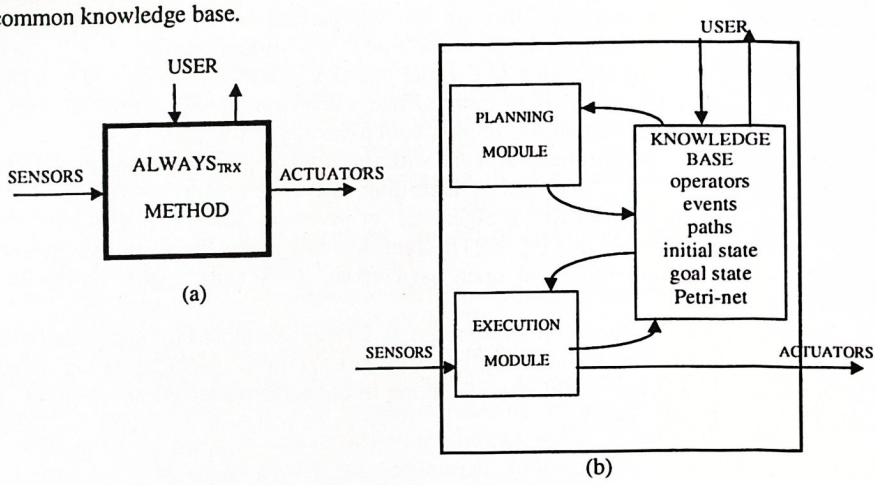

Fig. 1. *(a)* ALWAYS$_{TRX}$ method in the context "see, think, act"; *(b)* ALWAYS$_{TRX}$ method's internal architecture.

As the planning module generates the plan, the paths are stored in the knowledge base. The execution module reads the knowledge base and follows the plan to be executed. As the execution module works, the activity being executed and all the information about which external events are happening are stored in the knowledge base.

The planning module uses the information stored by execution module in the knowledge base in order to keep synthesizing the plan paths with updated world information.

4 RETE Algorithm Adaptation for the ALWAYS Method and Results

The RETE match algorithm is widely used in production rule systems. It works in two distinct phases: one of compilation (creation of the RETE net) and another one of matching check (operational). In the first phase, the algorithm compiles the set of rules inside the knowledge base (KB) in a net of linked nodes (an acyclic graph). Each node contains part of the conditions to fire a rule. In matching phase, the net processes each fact added or removed from the working memory. When a fact is added to the KB, it is presented to all nodes sons of root. Each node receives the fact, applies its test and, if the result of the test is true, the fact is stored and sent to the subsequent node (son nodes). If the test fails, the node simply ignores the presented fact and discards it. The last nodes (the leaves) in the net that receive a fact represent the rules to be fired.

In the implementation of the $ALWAYS_{TRX}$, RETE algorithm was adapted to make the analysis of the preconditions of operators and events, considering its performance and efficiency in finding equalities in multiple comparisons of facts with a high computational performance. The adapted RETE algorithm will be described in this section. The new adapted algorithm and new adapted RETE net will be referenced as RETEADAPT and RETEADAPT net, respectively.

Each change in world state (facts added or removed) calculated by the planning module is sent for the RETEADAPT and it indicates, as it was a combinational circuit, the operators and the events whose preconditions had been satisfied. With the operators and events enabled, the planner module expands each node of the planning tree.

RETEADAPT uses an acyclic graph where the nodes, except root node, represent the patterns that will be compared. The paths from root node until leaves represent the preconditions of the operators and of the events, which are the patterns to be analyzed. This graph is called RETEADAPT net. Inside RETEADAPT net, each node stores a pattern that will represent which elements from the world state satisfy the precondition of an operator or an event.

4.1 RETEADAPT Net Compiler

After the user insertion of operators and events with the corresponding preconditions, a procedure of creation of the net must be executed, that consists of compiling all the preconditions of operators and events, creating the nodes of the RETEADAPT net. Each node represents the test of a constant or variable declared as a parameter of a precondition. When there are preconditions with many parameters (constants or variables) and many predicates prototypes, a node will be created for each one of them.

RETEADAPT net has four types of nodes: the root node, one-input node, two-input node and enable-operator or enable-event node. Each one-input node can represent the name of the predicate prototype and the constants or variables that were declared as parameters. When a precondition has two terms linked by AND conjunction (e.g. clear(x)^ontable(y)), one-input nodes will be used to test the individual results and a two-input node will be used to join them.

Below there is an example of blocks world to illustrate how RETEADAPT net is created and used. It assumes the blocks world given by fig. 2. Three blocks are on a table and a manipulator robot must stack the blocks to reach a goal state. The definition of the $ALWAYS_{TRX}$ operators and events follows STRIPS definitions and can be seen in [24][25].

$ALWAYS_{TRX}$ operators:

```
operator     : pickup(x)
precondition: ontable(x)^handempty^clear(x)
del-list     : ontable(x),handempty,clear(x)
add-list     : holding(x),running(PICKUP)

operator     : putdown(x)
precondition: holding(x)
del-list     : holding(x)
add-list   :ontable(x),handempty,clear(x),running(PUTDOWN)

operator     : stack(x,y)
precondition: holding(x)^clear(y)
del-list     : holding(x),clear(y)
add-list     : handempty,on(x,y),clear(x),running(STACK)

operator     : unstack(x,y)
precondition: handempty^on(x,y)^clear(x)
del-list     : handempty,on(x,y),clear(x)
add-list     : holding(x),clear(y),running(UNSTACK)
```

$ALWAYS_{TRX}$ events:

```
event        : endactivity(x)
precondition: running(x)
del-list     : running(x)
add-list     : nil
```

In this example, the robot can execute four actions. The pickup(x) operator represents the robot action "to catch block x that is on the table and keep this block in its grip". The putdown(x) operator represents the robot action "to place block x, that is in its grip, on the table and release the block, freeing the grip". The stack(x, y) operator represents the robot action "to place block x, that is in its grip, on block y,

leaving the grip free". The unstack(x, y) operator represents the robot action "to catch a block x that is on block y and keep this block in its grip".

Initially RETEADAPT net must be created compiling the preconditions of the four operators. Analyzing the precondition of the pickup(x) operator, there are three terms: the predicate prototype *ontable* and *clear*, with x variable as parameter, and the predicate without parameters *handempty*. RETEADAPT net is assembled with three nodes below root node.

Initial State
clear(C)
clear(B)
on(C,A)
handempty
ontable(A)
ontable(B)

Goal State
on(B,C)
on(A,B)
ontable(C)
clear(A)

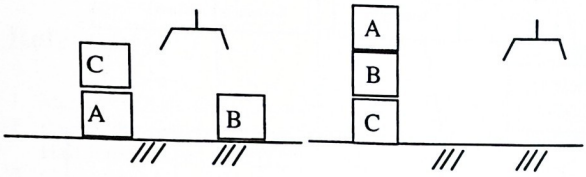

Fig. 2. Blocks world example. *Left* Initial State. *Right* Goal State.

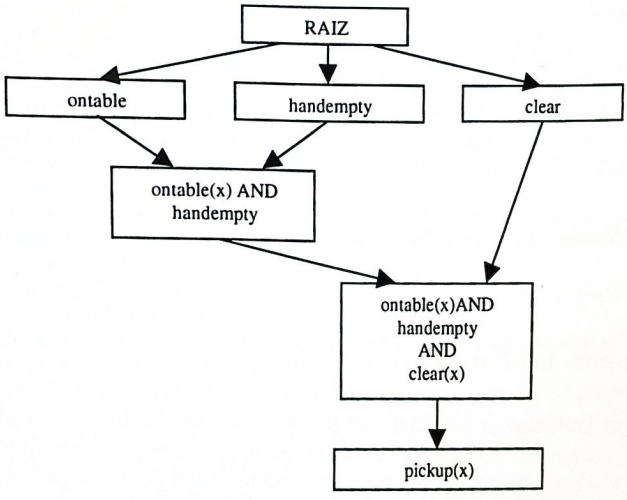

Fig. 3. RETEADAPT net for pickup(x) operator.

As the pickup(x) operator has a three term precondition with conjunctions AND, the algorithm creates two-input nodes to represent ontable(x)^handempty and after that ontable(x)^handempty^clear (x). Fig. 3 shows these two nodes with the enable operator node pickup(x). After the analysis of all preconditions of the others three operators, RETEADAPT net will be link fig. 4.

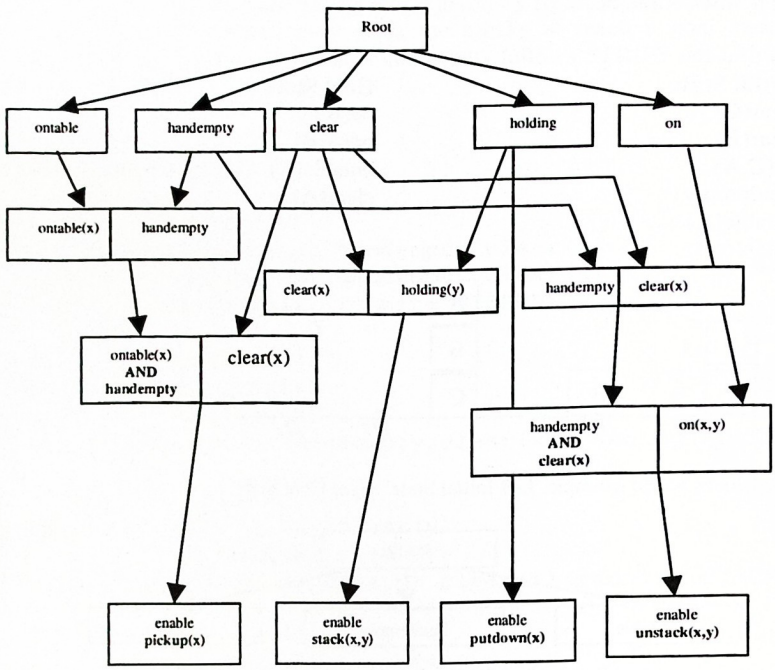

Fig. 4. RETEADAPT net complete after finished operator compilation.

5 Conclusions and Future Work

This paper presented an adaptation of RETE match algorithm, common used by production rule systems in expert systems, applied to real time planning systems. The adaptation was made for processing the instantiations of the predicate prototypes and testing of facts inside the Knowledge Base that describes the world state. This algorithm was implemented in a real time planner that uses $ALWAYS_{TRX}$ method. The planner works interleaved with execution and considers external events in the

planning phase. With this new algorithm the planning system increased the computational performance in analysis and synthesis of each state of the plan tree. Results show until 6 times less processing time of facts and overall planning time. The implementation of this suitable algorithm became possible the use of the real time planner to plan actions in many dynamic and unstructured environments. A detailed description of using this real time planning system on a manufacturing cell's manipulator robot, on a service mobile robot and in soccer robots, is being developed and will be published in the future.

Also as future work it is important to point out the performance optimization of the algorithm to better improve the response time of the system. The planner uses a RETEADAPT net in each node of the decision tree. Studies must be made to verify if it is possible to decrease the amount of information in each node and to use only one net. Another important aspect is in backtracking of the search tree where the corresponding predicates to the facts must be removed from the net in order to find new plans. Further studies must be made in compacting the data bytes used by the net.

References

1. Maes, P.: Designing Autonomous Agents. Mit/Elsevier (1991)
2. Russel, S., Norvig, P.: Artificial Intelligence: a modern approach. Second edition. Prentice Hall International (2003)
3. Georgeff, M.P. et al.: Reasoning about plans and actions. Exploring Artificial Intelligence. Morgan Kaufmann, AAAI, chap.5, pp.173--196 (1988)
4. Fikes, R.E., Nilsson, N.J.: Strips: a retrospective. Artificial Intelligence. 59, 227--232 (1993)
5. Noreils, F.R., Chatila, R.G.: Plan Execution Monitoring and Control Architecture for Mobile Robots. IEEE Transactions on Robotics and Automation. 11(2), pp.255--266 (1995)
6. Erol, K., Nau, D., Subrahmanian, V.S.: Complexity, decidability and undecidability results for domain independent planning. Artificial Intelligence. 76, 75--88 (1995)
7. Hoffmann, J. The FF planning system: fast plan generation through heuristic search. **The Journal of Artificial Intelligence Research.** 14, pp.253--302 (2001)
8. Abramson, B., NG, K.: Uncertainty Management in Expert Systems. IEEE Expert. 5(2), 29--48 (1990)
9. Golden, K., Etzioni, W., Weld, D.: Omnipotence without omniscience: Efficient sensor management for Planning. In: National Conference on Artificial Intelligence. 12, AAAI Press. pp.1048--1054 (1994)
10. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. San Mateo, California, Morgan Kauffmann (1988)
11. Knoblock, C.: Planning, executing, sensing and replanning for information gathering. In: IJCAI, 14, Montreal. pp.1686--1693 (1995)
12. Fikes, R.E., Nilsson, N.J.: Strips: A new approach to the application of theorem proving to problem solving. Artificial Intelligence. 2, 189--208 (1971)
13. Fikes, R.E.: Monitored Execution of robot plans produced by Strips. In: IFIP Congress, Ljubljana, Yugoslavia. 1--5 (1971)
14. Schoppers, M.J.: Universal Plans for reactive robots in unpredictable environments. In: IJCAI, 10, Milan. pp.1039--1046 (1987)
15. Allen, J., Hendler, J., Tate, A.: Readings in PLANNING. Morgan Kaufmann, San Mateo, California (1990)

16.Olawsky, D., Gini, M.: Deferred planning and sensor use. In: Workshop on Innovative Approaches to Planning, Scheduling and Control. San Diego, CA, DARPA. pp.166--174 (1990)
17.Wilkins, D.E.: Practical Planning: extending the classical AI planning paradigm. Morgan Kauffmann (1988)
18.Pemberthy, J., Weld, D.: UCPOP: A sound, complete, partial order planner for ADL. In: International Conference on Principles of Knowledge Representation and Reasoning. 3, pp.103--114 (1992)
19.Kushmeric, N., Hanks, S., Wold, D.: An algorithm for probabilistic planning. Artificial Intelligence, 1-2(76), pp.239--286 (1995)
20.Ash, D.J., Dabija, V.G.: Planning for Real Time Event Response Management. Prentice Hall PTR (2000)
21.Rillo, M.: Aplicações de Redes de Petri em Sistemas de Manufatura. São Paulo, PhD Thesis, Departamento de Engenharia de Eletricidade, EPUSP – Polytechnic School of University of Sao Paulo (1988)
22.Rillo, M.: Expectation-Based Temporal Projection System. In: Annual Conference on AI, Simulation and Planning in High Autonomy Systems, 3, Perth, Australia. pp.276--281 (1992)
23.Lopes, C.: Planejamento Baseado em Expectativas. São Paulo, PhD Thesis, Departamento de Engenharia de Eletricidade, EPUSP - Polytechnic School of University of Sao Paulo (1998)
24.Franchin, M.N. et. al.: Planejamento de ações com tratamento de eventos externos integrado ao controle de execução para agentes inteligentes. In: IV SBAI – Simpósio Brasileiro de Automação Inteligente. São Paulo, SP. pp.233--238 (1999)
25.Franchin, M.N.: Um método de planejamento usando eventos externos e sua integração com o sistema de controle para agentes móveis. São Paulo, PhD Thesis, Departamento de Engenharia de Eletricidade, EPUSP – Polytechnic School of University of Sao Paulo (1999)
26.Ulusar, U.D.,Akin, H.L. Design and implementation of a Real Time Planning System for Autonomous Robots. In: IEEE ISIE International Symposium on Industrial Electronics, july 9-12, Montréal, Québec, Canada. pp.74--79 (2006)
27.Forgy, C.L.: RETE: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence. 19(1), 17--37 (1982)
28.Winston, P.H.: Artificial Intelligence. 3rd ed., Addison Wesley, Reading, MA (1992)
29.Giarratano, J.,Riley, G.: Expert System Principles and Programming. 2nd ed., PWS Publishing Company (1993)
30.Miranker, D.P.: TREAT a better match algorithm for AI production systems. In: AAAI87 Conference on Artificial Intelligence, pp.42--47 (1987)
31.Hanson, E.N.: Gator: A Generalized Discrimination Network for Production Rule Matching. In: IJCAI workshop on Production Systems and Their Innovative Applications. (1993)
32.Mostafa M. Aref, M.M., Mohammed A. Tayyib, M.A.: Lana-Match Algorithm: A Parallel Version of the Rete-Match Algorithm. Parallel Computing. 24(5-6), pp.763--775 (1998)
33.Berstel, B.: Extending the RETE algorithm for event management. In: Proceedings of the Ninth International Symposium on Temporal Representation and Reasoning (TIME'02), pp.49--51 (2002)