

Modeling and Analyzing Multi-Agent Behaviors Using Predicate/Transition Nets

Dianxiang Xu, Richard A. Volz
Thomas R. Ioerger

Department of Computer Science
Texas A&M University
College Station, TX77843-3112, USA
 {xudian, volz, ioerger}@cs.tamu.edu

John Yen

School of Information Sciences and Technology
004 D Thomas Building
The Pennsylvania State University
University Park, PA 16802, USA
jyen@ist.psu.edu

Abstract

How agents accomplish a goal task in a multi-agent system is usually specified by multi-agent plans built from basic actions (e.g. operators) of which the agents are capable. The plan specification provides the agents with a shared mental model for how they are supposed to collaborate with each other to achieve the common goal. Making sure that the plans are reliable and fit for the purpose for which they are designed is a critical problem with this approach. To address this problem, this paper presents a formal approach to modeling and analyzing multi-agent behaviors using Predicate/Transition (PrT) nets, a high-level formalism of Petri nets. We model a multi-agent problem by representing agent capabilities as transitions in PrT nets. To analyze a multi-agent PrT model, we adapt the planning graphs as a compact structure for reachability analysis, which is coherent to the concurrent semantics. We also demonstrate that one can analyze whether parallel actions specified in multi-agent plans can be executed in parallel and whether the plans can achieve the goal by analyzing the dependency relations among the transitions in the PrT model.

Keywords: Formal methods, predicate/transition nets, multi-agent systems, plan, verification.

1. Introduction

One of the common approaches for the design and development of multi-agent systems is to specify multi-agent plans for achieving the joint goal in terms of agent capabilities [1-4]. The plan specification provides the agents with a shared mental model for how they are supposed to collaborate to achieve the common goal [2, 3]. Psychological research has shown that effective teamwork often relies on such a shared mental model [2] (this paper is not concerned about the interpretations of mental models of agents, such as intentions, desires, etc, because of the focus on the formal engineering perspective). Another major motivation of this approach is due to the difficulty a general-purpose planning algorithm has in generating a plan that is optimal in certain sense. The measurement of optimum may vary from

application to application. For example, the performance of a specific system may not simply depend on the number of actions, which is often treated as an important quality factor in planning systems [5]. A plan with minimum steps does not necessarily have minimum execution time because different actions usually have different durations of execution time. In contrast, pre-specified multi-agent plans may achieve better performance by taking into consideration specific application requirements. However, making sure the specified plans are fit for their purpose is a critical problem for plan designers. Specifically, are the multi-agent plans feasible for accomplishing the goal? Can specified parallel actions in multi-agent plans be executed in parallel? Is the goal achievable or reachable in terms of agent capabilities at all? From the formal engineering perspective, the answers to the above questions are of importance for detecting design defects.

To address these issues, this paper presents a formal approach to modeling and analyzing multi-agent behaviors using Predicate/Transition (PrT) nets [6, 7]. As a high-level formalism of Petri nets, PrT nets are capable of capturing logical relations among objects and actions due to the similarity to first order logic and thus well suited for modeling distributed intelligent systems. We think of the totality of agent capabilities specified by preconditions and post-conditions as a behavior model of what the agents as a group can do. We also think of multi-agent plans as a description of the process for moving from some start state, through this model, to a goal state. We demonstrate that PrT nets are suited for modeling multi-agent problems by representing agent capabilities as transitions in PrT nets. The reachability problem of a PrT net model can be analyzed efficiently using a compact structure called planning graphs [5], which is coherent to the concurrent semantics. We also demonstrate that one can analyze whether specified parallel actions can be executed in parallel and whether the parallel plans can achieve the expected goal by analyzing the dependency relations among the transitions in the PrT model. In particular, our approach can offer the system designer the following three types of feedback: 1) A specific action cannot be performed because its precondition is not satisfied; 2) A finite number of plan invocations or actions cannot be executed in parallel as specified because of the interdependencies among them; 3) A given goal is not reachable through the execution of a certain plan. Such feedback may indicate design defects in the plan specifications or in the definitions of agent capabilities.

The rest of this paper is organized as follows. Section 2 introduces PrT nets and shows how to model multi-agent problems using PrT nets. Section 3 describes the reachability analysis for PrT nets using the compact structure of planning graphs. Section 4 presents how to analyze parallel plans. Section 5 discusses the possibility of integrating dynamic planning into plan specification. Section 6 reviews related work. Section 7 concludes this paper.

2. Multi-Agent Modeling with PrT Nets

This section gives an introduction to PrT nets, and shows how to construct a multi-agent model using PrT nets.

2.1 Predicate/Transition Nets

A PrT net is a tuple $(P, T, F, \Sigma, L, \varphi, M_0)$, where:

- (1) P is a finite set of predicates (first order places), T is a finite set of transitions ($P \cap T = \emptyset$, $P \cup T \neq \emptyset$), and $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation, or simply a set of arcs. (P, T, F) forms a directed net.
- (2) Σ is a structure consisting of some sorts of individuals (constants) together with some operations and relations.
- (3) L is a labeling function on arcs. Given an arc $f \in F$, the labeling of f , $L(f)$, is a set of labels, which are tuples of individuals and variables (variables start with '?'). The tuples in $L(f)$ have the same length, representing the arity of the predicate connected to the arc. The zero tuple indicating a no-argument predicate (an ordinary place in Petri nets) is denoted by the special symbol $\langle \emptyset \rangle$.
- (4) φ is a mapping from transitions to a set of inscription formulae. The inscription on transition $t \in T$, $\varphi(t)$, is a logical formula built from variables and the individuals, operations, and relations in structure Σ ; Variables occurring free in a formula have to occur at an adjacent input arc of the transition.
- (5) M_0 is the initial marking. $M_0 = \bigcup_{p \in P} M_0(p)$, where $M_0(p)$ is the set of tokens residing in predicate p .

Each token is a tuple of symbolic individuals or structured terms constructed from individuals and operations in Σ .

The PrT nets above have simplified general PrT nets [6, 7] in two ways: 1) an arc labeling is a set of tuples (labels) $\{l_i\}$ rather than a formal sum $c_1l_1+c_2l_2+\dots+c_nl_n$ (i.e. coefficient or arc weight c_i of arc label l_i is 1 for all $1 \leq i \leq n$). 2) Accordingly, the marking of a specific predicate under a certain state is a set of tokens (i.e. items in [6, 7]) instead of a formal sum of tokens. Therefore, these nets have ruled out multiple sets of tokens that flow through arcs or held by a predicate. Similar to those in [8], they are more or less like first-order logic programs. From the formal verification perspective, the reachability analysis of such PrT nets is made more efficient for reachable states. Specifically, the planning graphs will be adapted as a compact structure.

To facilitate our discussion, we introduce some additional notation. For simplicity, we do not consider bi-directional arcs, which can be handled easily. Let $\bullet t = \{p \in P: (p, t) \in F\}$ and $t^\bullet = \{p \in P: (t, p) \in F\}$ be the precondition predicates and the post-condition predicates of transition t , respectively. Let $\bullet p = \{t \in T: (t, p) \in F\}$ and $p^\bullet = \{t \in T: (p, t) \in F\}$ be the sets of input transitions and output transitions of predicate p ,

respectively. A transition t in a PrT net is enabled under marking M if there is a substitution θ such that $l/\theta \in M(p)$ for any label $l \in L(p,t)$ for all $p \in \bullet t$ and $\varphi(t)$ evaluates true w.r.t. θ , where l/θ yields a token by substituting all variables in label l with the corresponding bound values w.r.t. θ . Under a certain marking, a transition may be enabled with different substitutions, but cannot be multiply enabled with the same substitution. The firing of enabled transition t w.r.t. θ (denoted as $t\theta$) removes all tokens in $\{l/\theta: l \in L(p,t)\}$ from each input predicate $p \in \bullet t$, and adds all tokens in $\{l/\theta: l \in L(t,p)\}$ to each output predicate $p \in t^\bullet$. Let $\bullet t\theta = \{l/\theta: l \in L(p,t) \text{ for any } p \in \bullet t\}$ and $t\theta^\bullet = \{l/\theta: l \in L(t,p) \text{ for any } p \in t^\bullet\}$ be the input (precondition) tokens and output (post-condition) tokens of firing $t\theta$, respectively.

Given a set of enabled firings $\gamma = \{t_1\theta_1, t_2\theta_2 \dots t_k\theta_k\}$ ($k > 0$) under M , γ is said to be concurrent firings if $(\bullet t_i\theta_i \cup t_i\theta_i^\bullet) \cap (\bullet t_j\theta_j \cup t_j\theta_j^\bullet) = \emptyset$, and $M \cap t_i\theta_i^\bullet = \emptyset$ for any $1 \leq i, j \leq k$ and $i \neq j$. After the firing of γ (called a step), we reach a new marking $M' = M - \{\bullet t_i\theta_i: 1 \leq i \leq k\} \cup \{t_i\theta_i^\bullet: 1 \leq i \leq k\}$. A sequence of firing steps that reaches marking M_n from M_0 is denoted as $\gamma_1\gamma_2 \dots \gamma_n$ or $M_0\gamma_1M_1\gamma_2M_2 \dots \gamma_nM_n$, where M_i is the marking after step γ_i ($1 \leq i \leq n$). A marking M is said to be reachable from M_0 if there is such a sequence of firing steps that transforms M_0 to M . Here, reachability is defined in terms of the truly concurrent semantics rather than the interleaving interpretation of concurrency used by existing methods for the reachability analysis of PrT nets [9, 10], where each step contains a single firing (and therefore a sequence of firing steps reaching some marking is called an occurrence sequence). As will be shown later, the planning graph-based reachability analysis complies with the concurrent semantics.

2.2 Modeling Multi-Agent Problems

For a multi-agent problem, the PrT model can be constructed by focusing on the actions of which agents are capable as well as the environment representation. The actions are specified by transitions with preconditions, post-conditions and inscriptions, whereas the environment is often represented by predicates.

As a case study, this paper extends the well-known blocks world problem [11] by introducing multiple agents with different capabilities and different types of blocks. This extension facilitates the comparison between two cases with the same state space: 1) there are multiple agents that can work in parallel, and 2) there is only one agent that is capable of moving any type of block but has no parallel operations (i.e. the same as the classical blocks world problem).

In the extended multi-agent blocks world, a block is specified by a pair, (x_1, x_2) , where $x_1 \in \{a, b, c\}$ and $x_2 \in \{1, 2, 3, \dots\}$ are its type and identification number, respectively. Two agents, r_1 and r_2 , as a team, are supposed to cooperatively move two stacks of blocks. Their capabilities are different in that r_1 alone

can move blocks of types a, whereas r_2 alone can only move blocks of type b. As a team, however, r_1 and r_2 can move heavier blocks of type c.

The transitions for the extended blocks world problem are specified in TABLE 1. The totality of these transitions constitutes a complete PrT net, where tokens may be interpreted as facts or agent beliefs in the logical sense. Fig.1 shows a part of the PrT net. Note that, r_1r_2 pickup is an abstract, joint action between agents r_1 and r_2 . The individual part of each agent involved in this joint action is not specified. It can be either decomposed in terms of certain cooperation patterns at runtime or refined stepwise at the design stage. This is similar for r_1r_2 putdown, r_1r_2 stack, and r_1r_2 unstack.

TABLE 1. THE PrT MODEL FOR THE MULTI-AGENT BLOCKS PROBLEM

<i>transition</i>	<i>precondition predicates & arc labels</i>	<i>post-condition predicates & arc labels</i>	<i>Inscription</i>
r_1r_2 pickup	ontable<?x1,?x2>, clear<?x1,?x2>, r1handempty < \emptyset >, r2handempty < \emptyset >	r1holding <?x1,?x2>, r2holding <?x1, ?x2>	equal(?x1,c)
r_1 pickup	ontable<?x1,?x2>,clear<?x1,?x2> r1handempty < \emptyset >	r1holding<?x1,?x2>	equal(?x1,a)
r_2 pickup	ontable<?x1,?x2>,clear<?x1,?x2>, r2handempty < \emptyset >	r2holding<?x1,?x2>	equal(?x1,b)
r_1r_2 putdown	r1holding<?x1,?x2>, r2holding<?x1,?x2>	ontable<?x1,?x2>,clear<?x1,?x2>, r1handempty< \emptyset >, r2handempty< \emptyset >	equal(?x1,c)
r_1 putdown	r1holding<?x1,?x2>	ontable<?x1,?x2>, clear<?x1,?x2>, r1handempty< \emptyset >	equal(?x1,a)
r_2 putdown	r2holding<?x1,?x2>	ontable<?x1,?x2>, clear<?x1,?x2>, r2handempty< \emptyset >	equal(?x1,b)
r_1r_2 stack	r1holding<?x1,?x2>, r2holding<?x1,?x2>, clear<?y1,?y2>	r1handempty< \emptyset >,r2handempty< \emptyset >, on<?x1,?x2,?y1,?y2>, clear<?x1,?x2>	equal(?x1,c)
r_1 stack	r1holding<?x1,?x2>,clear<?y1,?y2>	r1handempty< \emptyset >, on<?x1,?x2,?y1,?y2>,clear<?x1,?x2>	equal(?x1,a)
r_2 stack	r2holding<?x1,?x2>,clear<?y1,?y2>	r2handempty< \emptyset >, on<?x1,?x2,?y1,?y2>,clear<?x1,?x2>	equal(?x1,b)
r_1r_2 unstack	r1handempty< \emptyset >,r2handempty< \emptyset >, clear<?x1,?x2>,on<?x1,?x2,?y1,?y2>	r1holding<?x1,?x2>, r2holding<?x1,?x2>, clear<?y1,?y2>	equal(?x1,c)
r_1 unstack	r1handempty< \emptyset >,clear<?x1,?x2>, on<?x1,?x2,?y1,?y2>	r1holding<?x1,?x2>, clear<?y1,?y2>	equal(?x1,a)
r_2 unstack	r2handempty< \emptyset >,clear<?x1,?x2>, on<?x1,?x2,?y1,?y2>	r2holding<?x1,?x2>, clear<?y1,?y2>	equal(?x1,b)

From Table 1, we can tell that transitions bear much similarity to STRIPS operators [12] for planning problems. The main differences include: 1) Transitions are associated with logical formulae, and 2) A precondition of a transition becomes false (similar to a delete-effect) unless it is also a post-condition. 3) Tokens may represent structured data items. In fact, a set of STRIPS operators can be converted into a PrT net [8].

It is worth mentioning that, although a planning problem is essentially a reachability problem, planning and verification have different concerns. A planning algorithm intends to consider the quality

(e.g. number of steps) and optimization of the resulting plan. In addition to the reachability issue, a verification algorithm may need to analyze other system properties such as liveness, deadlock-freedom, etc.

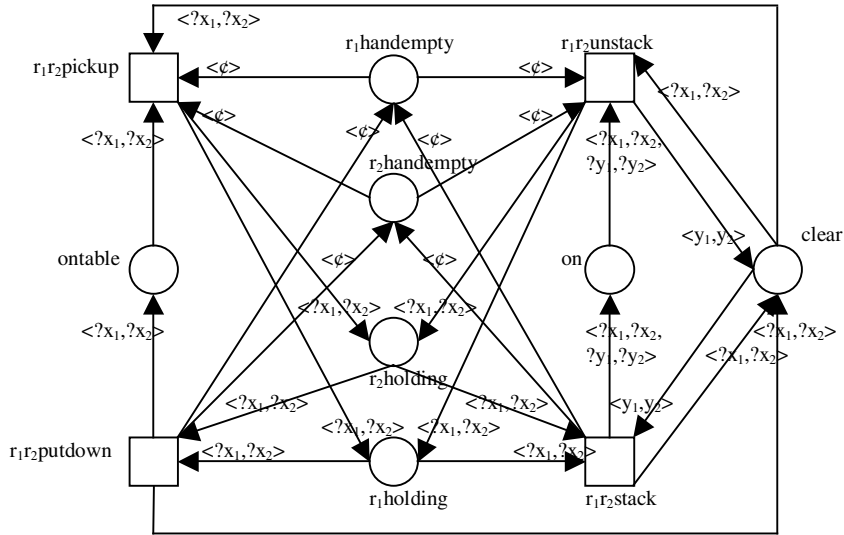


Fig.1 A part of the PrT net for the extended blocks world problem

3. Reachability Analysis of Multi-Agent Models

This section first reviews the planning graph analysis, and then describes how to adapt it for the reachability analysis of PrT models.

3.1 Overview of Planning Graph Analysis

The planning graph analysis in Graphplan [5] solves STRIPS planning problems [12] in which operators have preconditions, add-effects and delete-effects, all represented as conjuncts of (parameterized) propositions. Empirical studies have demonstrated the practical value of planning graph analysis in a dozen of benchmark planning domains. The planning graph analysis alternates between two phases: graph expansion and solution extraction. The graph expansion phase extends a planning graph until a necessary condition for plan existence is achieved (i.e. goal propositions all appear at the same level, and no pair of goal propositions is mutually exclusive). The solution extraction phase then performs a backward-chaining search for an actual solution. If no solution is found, the cycle repeats.

A planning graph is a directed graph with alternating levels of proposition nodes and action nodes. The first level is composed of the propositions in the initial state. Nodes in an action level consist of

actions, i.e., instances of operators, whose preconditions occur and no pair of propositions in the preconditions is determined mutually exclusive at the previous proposition level. Directed edges connect proposition nodes to corresponding action nodes whose preconditions reference those propositions, and connect action nodes to subsequent propositions created by the action's effects. Each proposition at a level is also connected to a no-op or frame action (doing nothing) at the next action level, which is in turn linked to the same proposition at the next proposition level.

To analyze a planning graph, the crucial work is reasoning about certain mutual exclusion relations among nodes, i.e. propositions or actions. An "actions-that-I-am-exclusive-of" list is created for each action node. Two actions at a given action level in a planning graph are marked mutually exclusive: 1) If one action deletes a precondition or add-effect of the other (interference), or 2) If one action has a precondition that is marked as mutually exclusive of a precondition of the other action (competing needs). Two propositions p and q in a proposition level are marked as exclusive if each action creating p is marked as exclusive of each action creating q .

3.2 Reachability Analysis of PrT Nets

The planning graph analysis for STRIPS planning problems can be adapted for the reachability analysis of PrT nets by including three major aspects:

- Mapping of terminology: each token at a certain predicate is a proposition, and each transition firing (a pair of transition and substitution) is an action. Each precondition of a firing in a PrT net is a delete-effect in the planning graph unless it is also a post-condition of the firing. Each post-condition that is not a precondition of a firing is an add-effect in the planning graph. The reasoning and propagation of mutual exclusion relations is adjusted accordingly.
- Generation of action level: a firing is defined by the firing rule described in subsection 2.1. The inscription formula must evaluate true with respect to the substitution. The unification algorithm between arc labels and token needs to consider the case where tokens are structured data (like terms in a logic language).
- The rule of interference is enhanced. Two actions are mutually exclusive (i.e. they cannot be concurrent) if they have a common effect (post-condition). This will be further illustrated through Fig.3 in subsection 4.2.

In the following, we first outline the procedure of graph expansion, and then the procedure of solution extraction for the analysis of PrT nets.

Procedure1 (Graph expansion)

Step 1: initialization, e.g. construction of token level 0 in terms of the initial marking;

Step 2: WHILE (solution extraction fails and level i (if $i > 0$) and level $i-1$ are not identical) DO

// create graph level $i+1$

Step 2.1: For any token in token level i , create a no-op firing (a frame action doing nothing) in firing level $i+1$, and the same token in token level $i+1$. The precondition and post-condition of the no-op are the token in level i and level $i+1$, respectively.

Step 2.2: Add firing $t\theta$ to firing level $i+1$ for any transition t and any substitution θ of t such that:

- 1) l/θ is contained in token level i for any label $l \in L(p,t)$ for all $p \in \bullet t$;
- 2) $\varphi(t)$ evaluates true w.r.t. θ ; and
- 3) There does not exist p_1, p_2, l_1 and l_2 such that $l_1 \in L(p_1,t), l_2 \in L(p_2,t), (l_1 \neq l_2 \text{ if } p_1 = p_2)$, and l_1/θ and l_2/θ are mutually exclusive at token level i .

Step 2.3: For any firing $t\theta$ created in step 2, add token l/θ to token level $i+1$ for any label $l \in L(t, p)$ for all $p \in t^\bullet$. Connect each precondition token in level i to the firing, and connect the firing to each of its post-condition tokens in level $i+1$.

Step 2.4: Any pair of firings $t_1\theta_1$ and $t_2\theta_2$ (including no-op firings) in level $i+1$ are marked as competing needs if there exists $p_1, p_2, l_1 \in L(p_1,t_1), l_2 \in L(p_2,t_2)$ such that l_1/θ_1 and l_2/θ_2 are mutually exclusive of each other.

Step 2.5: Any pair of tokens τ_1 and τ_2 in token level $i+1$ are marked as mutually exclusive if for any firing $t_1\theta_1$ creating τ_1 and any firing $t_2\theta_2$ creating τ_2 in firing level $i+1$, $t_1\theta_1$ and $t_2\theta_2$ either have competing needs or interfere (the precondition/post-condition of one firing contains a token in the precondition/post-condition of the other, i.e. $(\bullet t_1\theta_1 \cup t_1\theta_1^\bullet) \cap (\bullet t_2\theta_2 \cup t_2\theta_2^\bullet) = \emptyset$.)

Step 3: The goal is reachable if solution extraction succeeds; otherwise the goal is unreachable.

Procedure 2 (solution extraction)

Step 1: If there is a token in the given goal that is not contained in the newly created token level or if two tokens in the goal are mutually exclusive in the token level, return solution extraction fails (the goal is not reachable at this level).

Step 2: Initialization for solution extraction. Let current goal (variable) be the given system goal, and current level (variable) be the newly generated level.

Step 3: WHILE the first token level (initial state) is not reached DO

Step 3.1. IF current goal is already proven unsolvable in current level, THEN:

IF current level is the newly created level, THEN solution extraction fails ELSE backtrack to the previous level (i.e. current level++, reset the goal, and continue);

Step 3.2: Starting from the initial point of firing selection for the first time of current goal in current level or from the backtrack point of firing selection, select a minimum set of firings in current firing level such that these firings lead to current goal in current level;

Step 3.3: IF there exists such a set of firings, THEN set the backtrack point, and deduce current goal to a new set of goals in the next token level according to the preconditions of the selected firings; ELSE current goal is not solvable in current level (put it into the hash table) and IF current level is the newly created level, THEN solution extraction fails and return to the graph expansion, ELSE backtrack to the previous level (i.e. current level++, reset the goal, and continue);

Step 3.4: Set current level to the next one (i.e. current level--);

Step 3.5: IF current token level is the first token level THEN output solution for the reached goal.

The above algorithm follows the major features of Graphplan. For example, planning graphs have polynomial size and can be constructed in polynomial time (refer to [5] for more details). As a limitation, though not clearly stated in [5], planning graph analysis is more efficient only for problems where mutual exclusion relations can be fully determined by the rules (Finding all mutual exclusion relations for general problems is as hard as planning itself [5] or as hard as the reachability issue. This reflects the inherent complexity of high-level Petri nets). In particular, two parallel actions under a certain state are supposed not to produce an identical output, which in turn becomes a common precondition of multiple parallel actions (in many cases, this can be avoided by distinguishing identical tokens with an extra parameter). In this paper, the limitation is implied by the safeness assumption of PrT nets. Note that for simplicity safeness is just assumed but not guaranteed by the definition of fireability. To make sure a PrT is safe, we may explicitly impose $M_0 \cap t\theta^* = \emptyset$ as an additional necessary condition on the enabledness (i.e. post-conditions as inhibitor conditions). However, it is not trivial to extend the rules for the reasoning and propagation of mutual exclusion relations (The analysis of PrT nets with inhibitor arcs is currently under investigation). In conclusion, the practical value of planning graph analysis is that for a large class of problems the solution can often be found quickly if a goal is reachable. It does not change the difficult nature of the reachability problem of high-level Petri nets, however.

If a goal is reachable in the PrT model, the solution found by the algorithm consists of a number of steps, and each step may contain more than one firing. This is exactly coherent to the concurrent semantics of PrT nets defined in this paper. Suppose the initial marking of the PrT model for the multi-agent blocks world problem is (see Fig.2-a):

{ontable: <a,n3>, ontable: <b,n6>, clear:<b,n1>,
clear:<(a,n4)>, on: <c,n2,a,n3>, on: <b,n1,c,n2>},

```

on: <c,n5,b,n6>, on:<a,n4,c,n5>,
r1handempty: <ϕ>, r2handempty: <ϕ>
}

```

and the goal marking (see Fig.2-b) is

```

{ontable: <b,n1>, ontable: <a,n4>,
clear: <a,n3>, clear:<b,n6>,
on: <a,n3,b,n1>, on: <c,n5,a,n4>,
on(c,n2,c,n5), on(b,n6,c,n2)}

```

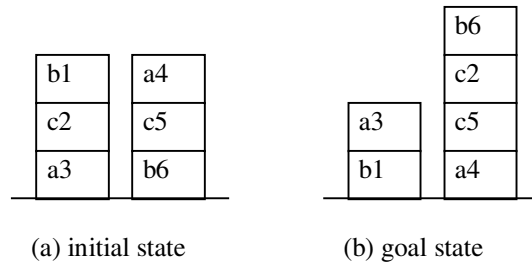


Fig.2 An extended blocks world

The solution found by the algorithm we have implemented in Java is as follows:

```

S1: r2unstack(?x1/b,?x2/n1,?y1/c,?y2/n2)
    r1unstack(?x1/a,?x2/n4,?y1/c,?y2/n5)
S2: r2putdown(?x1/b,?x2/n1)
    r1putdown(?x1/a,?x2/n4)
S3: r1r2unstack(?x1/c,?x2/n5,?y1/b,?y2/n6)
S4: r1r2stack(?x1/c,?x2/n5,?y1/a,?y2/n4)
S5: r1r2unstack(?x1/c,?x2/n2,?y1/a,?y2/n3)
S6: r1r2stack(?x1/c,?x2/n2,?y1/c,?y2/n5)
S7: r1pickup(?x1/a,?x2/n3)
    r2pickup(?x1/b,?x2/n6)
S8: r1stack(?x1/a,?x2/n3,?y1/b,?y2/n1)
    r2stack(?x1/b,?x2/n6,?y1/c,?y2/n2)

```

Note that the two firings within step S1 (S2, S7, or S8) can be executed in parallel by two agents, respectively. To illustrate the efficiency of the algorithm above for parallel systems, we briefly compare the example above with the classical, single agent case (the model can be obtained by only using four transitions for agent r_1 and removing all transition inscriptions in Table 1). The two cases share the same

state space. For the multi-agent case with the given initial state and goal state described above, it takes 0.297 seconds for the algorithm to find the given solution. For the single-agent case with the same initial state, goal state and running environment (Windows 2000, 800MHZ, 256MB), it takes 0.766 seconds to find the 12-step solution. In the multi-agent case, the size of action space is increased, but the reachability analysis is faster due to the parallelism among actions. This indicates that the planning graph analysis is more suitable for concurrent systems.

To support the verification of other system properties like deadlock freedom, we may specify a goal as a more general formula instead of a marking (or a set of facts in planning domains) so that some system properties can be formalized as reachability problems. This is accomplished by enhancing the first step in the procedure of solution extraction.

4. Analyzing Multi-Agent Plans

Multi-agent systems often exhibit rich parallelism among agents, which requires implicit or explicit representation of parallel actions in multi-agent plans. This section describes the specification and verification of parallel plans, regarding whether specified parallel actions can be actually executed in parallel, and whether they achieve the given goal.

4.1 Plan Specification

Multi-agent plans are essentially the process for moving from some initial state, through the system model (e.g. PrT model), to a goal state. The most simplistic form of multi-agent plans is a sequence of actions, i.e. transition firings. For convenience, we associate each transition with an ordered list of variables (formal parameters) that appear in the labels of the arcs connected to its precondition predicates. For example, $r2unstack$ is associated with formal parameters $(?x1, ?x2, ?y1, ?y2)$, thus firing $r2unstack(?x1/b, ?x2/n1, ?y1/c, ?y2/n2)$ can be simply represented by $r2unstack(b, n1, c, n2)$, which is like a procedure call. Verifying a given sequence of actions against the goal is a check to see if it is an occurrence sequence that transforms the initial state to the goal state in the PrT model. However, a plan specification language usually provides complex constructs, such as parallel, plan invocation (to form a plan hierarchy), condition, loops etc. This paper focuses on parallel actions in plan hierarchy, which reflect the core constructs of plan specification languages, such as [1-4]. Other control structures like condition and loop can be incorporated. In the following, we formally define multi-agent plans.

Definition 1. A plan ρ is a structure $\langle \text{name}, \text{arguments}, \text{process} \rangle$, where name, arguments and process are the plan identifier, the formal parameters and process. A process is an action (transition and substitution),

a plan invocation, or a finite sequence of processes (separated by ‘,’), and a parallel structure with a finite number of processes (separated by ‘|’). Formally, the BNF-style notation of processes is as follows:

```

<process> ::= <transition>[(<arguments>)]
           | <plan-name>(<arguments>)
           | <process>{,<process>}
           | (<process> { | <process> })

```

where arguments are actual parameters of an action or plan invocation.

For example, we may define the following plan with two sequential actions for the extended blocks world problem.

```

plan r1move1(?x1, ?x2, ?y1, ?y2) {
    r1unstack(?x1, ?x2, ?y1, ?y2),
    r1putdown(?x1, ?x2)
}

```

In the foregoing plan for agent r_1 , the first action is unstack block (x_1, x_2) on block (y_1, y_2) , and the second action is putdown block (x_1, x_2) on table. More complex plan examples will be introduced in subsection 4.3.

It is worth pointing out that our approach is concerned with multi-agent plan specifications at the system level, other than at the individual agent level. In essence, the multi-agent plans simulate the shared-mental model of a multi-agent team, which is one of the key components for effective teamwork [2]. Individual agent plans are generated automatically during plan execution according to the system level specifications, as well as coordination and communication patterns.

4.2 Plan Analysis

Although plans look like procedural programs, plan analysis is made possible and tractable by the complete specification of preconditions and post-conditions of actions. The crucial issue is how to determine if specified parallel actions in plan hierarchy can be executed in parallel.

The work on parallelizing execution of machine instructions has identified three types of dependency: procedural, operational and data [13]. Knoblock has adapted these dependency types for generating parallel execution plans with a partial order planner [14]. A procedural dependency occurs when one action is explicitly ordered after another. A data dependency occurs when the precondition of an operation depends on the effects of another actions. An operational dependency may occur when there are limited resources associated with an action. In our context, these dependencies are implied by the specifications of transitions and initial state.

Given two actions or firings $t_1\theta_1$ and $t_2\theta_2$ under marking M , they can be executed in parallel if: 1) t_1 and t_2 are firable w.r.t. θ_1 and θ_2 under marking M , respectively, and 2) the firing of either $t_1\theta_1$ or $t_2\theta_2$ will not disable the other, i.e. $\bullet t_1\theta_1 \cap \bullet t_2\theta_2 = \emptyset$. Since duplicate tokens are not allowed for the PrT nets defined in this paper, it also requires the unions of preconditions and post-conditions of $t_1\theta_1$ and $t_2\theta_2$ should not share any token, i.e. $(\bullet t_1\theta_1 \cup t_1\theta_1^\bullet) \cap (\bullet t_2\theta_2 \cup t_2\theta_2^\bullet) = \emptyset$. In other words, two firings can be executed in parallel only if they are independent. For example, suppose the initial state for the PrT net in Fig. 3 is $\{p_1\langle a \rangle, p_2\langle a \rangle\}$, $t_1(a)$ and $t_2(a)$ are not allowed to be fired in parallel though they are both enabled and do not have common preconditions, otherwise $\{p_4\langle a \rangle, p_5\langle a \rangle\}$ will never be reached (here a token of a predicate is like a proposition in logical sense). Furthermore, although “ $t_1(a), t_3(a)$ ” and “ $t_2(a), t_4(a)$ ” are both occurrence sequences from the initial state, they cannot be executed in parallel because the result depends on the ordering of actions. If t_1 and t_2 are expected to be concurrent, we may differentiate the common post-condition of t_1 and t_2 by introducing an extra parameter, i.e. using $\langle ?x, a \rangle$ and $\langle ?x, b \rangle$ as the labels for arcs (t_1, p_3) and (t_2, p_3) , respectively. In this case, the labels for arcs (p_3, t_3) and (p_3, t_4) need to be changed to $\langle ?x, ?y \rangle$.

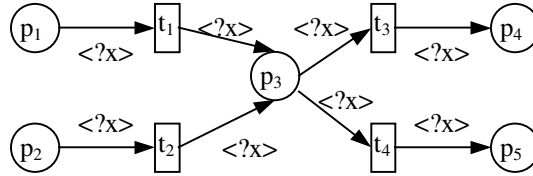


Fig.3 A PrT net

In the following definitions, we generalize the above requirement.

Definition 2. Given two actions $t_1\theta_1$ and $t_2\theta_2$. If $(\bullet t_1\theta_1 \cup t_1\theta_1^\bullet) \cap (\bullet t_2\theta_2 \cup t_2\theta_2^\bullet) = \emptyset$, then $t_1\theta_1$ and $t_2\theta_2$ are said to be independent.

Definition 3. Given marking M , and two sequences of actions $\delta_1 = t_{11}\theta_{11} t_{12}\theta_{12} \dots t_{1m}\theta_{1m}$ and $\delta_2 = t_{21}\theta_{21} t_{22}\theta_{22} \dots t_{2n}\theta_{2n}$. If $t_{1i}\theta_{1i}$ and $t_{2j}\theta_{2j}$ are independent for any i and j ($1 \leq i \leq m, 1 \leq j \leq n$), and both δ_1 and δ_2 are occurrence sequences starting from M . Then δ_1 and δ_2 are said to be independent and executable in parallel starting from M .

Given a sequence of actions $\delta = t_1\theta_1 t_2\theta_2 \dots t_m\theta_m$, let $\bullet \delta = \bigcup_{i=1}^m \bullet t_i\theta_i$ and $\delta^\bullet = \bigcup_{i=1}^m t_i\theta_i^\bullet$ denote the unions of all precondition and post-condition tokens of actions in δ . The following theorem shows that the complexity of determining the dependency between δ_1 and δ_2 is $O(m+n)$.

Theorem 1 Given marking M , and two sequences of actions δ_1 and δ_2 . δ_1 and δ_2 are independent if and only if $(\bullet \delta_1 \cup \delta_1^\bullet) \cap (\bullet \delta_2 \cup \delta_2^\bullet) = \emptyset$.

Proof: Let $\delta_1 = t_{11}\theta_{11} t_{12}\theta_{12} \dots t_{1m}\theta_{1m}$ and $\delta_2 = t_{21}\theta_{21} t_{22}\theta_{22} \dots t_{2n}\theta_{2n}$.

(1) Suppose $(\delta_1 \cup \delta_1) \cap (\delta_2 \cup \delta_2) = \emptyset$. For any i and j ($1 \leq i \leq m, 1 \leq j \leq n$), $(t_{1i}\theta_{1i} \cup t_{1i}\theta_{1i}) \subseteq (\delta_1 \cup \delta_1)$, and $(t_{2j}\theta_{2j} \cup t_{2j}\theta_{2j}) \subseteq (\delta_2 \cup \delta_2)$. so $(t_{1i}\theta_{1i} \cup t_{1i}\theta_{1i}) \cap (t_{2j}\theta_{2j} \cup t_{2j}\theta_{2j}) = \emptyset$. That is, $t_{1i}\theta_{1i}$ and $t_{2j}\theta_{2j}$ are independent, and therefore δ_1 and δ_2 are independent.

(2) Suppose δ_1 and δ_2 are independent. For any i and j ($1 \leq i \leq m, 1 \leq j \leq n$), $(t_{1i}\theta_{1i} \cup t_{1i}\theta_{1i}) \cap (t_{2j}\theta_{2j} \cup t_{2j}\theta_{2j}) = \emptyset$. For any $\alpha \in (\delta_1 \cup \delta_1)$, there exists t_{1i} such that $\alpha \in (t_{1i}\theta_{1i} \cup t_{1i}\theta_{1i})$. So $\alpha \notin (t_{2j}\theta_{2j} \cup t_{2j}\theta_{2j})$ for any $1 \leq j \leq n$, i.e., $\alpha \notin (\delta_2 \cup \delta_2)$. Similarly, for any $\beta \in (\delta_2 \cup \delta_2)$, $\beta \notin (\delta_1 \cup \delta_1)$. Therefore $(\delta_1 \cup \delta_1) \cap (\delta_2 \cup \delta_2) = \emptyset$.

According to (1) and (2), the theorem holds. QED

In order to deal with parallel actions in plan hierarchy, we present another theorem.

Theorem 2. Suppose two sequences of actions δ_1 and δ_2 are independent. Action sequence δ_3 is independent of both δ_1 and δ_2 if and only if δ_3 is independent of $\delta_1 + \delta_2$, the sequential concatenation of δ_1 and δ_2 .

Proof: Let $\delta_1 = t_{11}\theta_{11} t_{12}\theta_{12} \dots t_{1m}\theta_{1m}$, $\delta_2 = t_{21}\theta_{21} t_{22}\theta_{22} \dots t_{2n}\theta_{2n}$, and $\delta_3 = t_{31}\theta_{31} t_{32}\theta_{32} \dots t_{3k}\theta_{3k}$. Note that $(\delta_1 + \delta_2) \cup (\delta_1 + \delta_2) = \delta_1 \cup \delta_1 \cup \delta_2 \cup \delta_2$.

(1) Suppose δ_3 is independent of $\delta_1 + \delta_2$. $(\delta_3 \cup \delta_3) \cap ((\delta_1 + \delta_2) \cup (\delta_1 + \delta_2)) = \emptyset$. It is trivial to show $(\delta_3 \cup \delta_3) \cap (\delta_1 \cup \delta_1) = \emptyset$ and $(\delta_3 \cup \delta_3) \cap (\delta_2 \cup \delta_2) = \emptyset$. That is, δ_3 is independent of both δ_1 and δ_2 .

(2) Suppose δ_3 is independent of both δ_1 and δ_2 . $(\delta_3 \cup \delta_3) \cap (\delta_1 \cup \delta_1) = \emptyset$ and $(\delta_3 \cup \delta_3) \cap (\delta_2 \cup \delta_2) = \emptyset$. So $(\delta_3 \cup \delta_3) \cap (\delta_1 \cup \delta_1 \cup \delta_2 \cup \delta_2) = \emptyset$. That is, $(\delta_3 \cup \delta_3) \cap ((\delta_1 + \delta_2) \cup (\delta_1 + \delta_2)) = \emptyset$. So δ_3 is independent of $\delta_1 + \delta_2$.

According to (1) and (2), the theorem holds. QED.

Theorem 2 shows that whenever a parallel plan or structure has been proven correct, it can be flattened into a sequence for the verification of outer level plans in which it is invoked or nested. For instance, the verification of plan $(\delta_4, (\delta_1 | \delta_2)) | \delta_3$ can be reduced to the verification of $(\delta_4, \delta_1, \delta_2) | \delta_3$ if δ_1 and δ_2 are already proven independent.

In the following, we outline the algorithm for checking the parallelism in plan hierarchy.

Procedure 3 (checking parallelism in plan hierarchy)

Input: starting state M_0 for plan invocation $\rho(\theta)$, where ρ and θ are plan name and actual parameters, respectively.

Output: a) $\sigma(M_0, \rho, \theta)$ = the sequence of actions implied by plan invocation $\rho(\theta)$ or $\sigma(M_0, \rho, \theta)$ = empty if some specified parallel actions/sub-plan invocations inside $\rho(\theta)$ cannot be executed in parallel, and b) $M'(M_0, \rho, \theta)$: the ending marking of executing $\rho(\theta)$.

Step1: If the process is a sequence structure of the form $\rho_1(v_1), \rho_2(v_2), \dots, \rho_m(v_m)$, Then *For* $i=1$ TO m DO

Case 1: $\rho_i(v_i)$ is an action, i.e. ρ_i is a transition.

If $\rho_i(v_i/\theta)$ is not firable under M

Then plan ρ is not feasible, and return $\sigma(M_0, \rho, \theta) = \text{empty}$

Else $M_i = M_{i-1} - \bullet \rho_i(v_i/\theta) \cup \rho_i(v_i/\theta) \bullet$

Case 2: $\rho_i(v_i)$ is a plan invocation.

Recursively call with input (M_{i-1}, ρ_i, v_i) .

If $\sigma(M_{i-1}, \rho_i, v_i)$ is empty

Then return $\sigma(M_0, \rho, \theta) = \text{empty}$

Else $M_i = M'(M_{i-1}, \rho_i, v_i)$.

Step2: If the process is a parallel structure of the form $\rho_1(v_1) \mid \rho_2(v_2) \mid \dots \mid \rho_m(v_m)$

Step2.1 *For* $i=1$ TO m DO

Recursively call with input $(M_0, \rho_i, v_i/\theta)$;

1) Let $\sigma_i = \sigma(M_0, \rho_i, v_i/\theta)$;

2) If σ_i is empty

Then return $\sigma(M_0, \rho, \theta) = \text{empty}$

3) If $i=1$

Then let $M_1 = M'(M_0, \rho_i, v_i/\theta)$,

Step 2.2 If there exist i, j ($1 \leq i, j \leq m$ and $i \neq j$) such that $(\bullet \sigma_i \cup \sigma_i \bullet) \cap (\bullet \sigma_j \cup \sigma_j \bullet) \neq \emptyset$ Then $\rho_i(v_i/\theta)$ and $\rho_j(v_j/\theta)$ cannot be executed in parallel, and return $\sigma(M_0, \rho, \theta) = \text{empty}$;

Step2.3 *For* $i=2$ TO m DO

Let $\sigma_i = t_1\theta_1, t_2\theta_2, \dots, t_k\theta_k$

For $j=1$ to k DO

Let $M_1 = M_1 - \bullet t_j\theta_j \cup t_j\theta_j \bullet$

Step2.4: Let $\sigma(M_0, \rho, \theta) = \sigma_1 + \sigma_2 + \dots + \sigma_m$, let $M'(M_0, \rho, \theta) = M_1$, and then return.

Whether a given goal G is reachable through an invocation with actual parameters θ of plan ρ from some initial state M_0 can be analyzed by checking whether G is included in $M'(M_0, \rho, \theta)$.

In summary, the plan analysis can report following three types of feedback to the designer: 1) A specific action cannot be performed because its precondition is not satisfied; 2) A finite number of plan invocations or actions cannot be executed in parallel as specified because of the dependencies among them; 3) A given goal is not reachable through the execution of a certain plan. Such feedback may indicate design defects in the plan specifications or in the definitions of agent capabilities.

When the given goal is not reachable by specified plans, the designer may also check the PrT model presented in last section to see if the goal is reachable at all. Note that, if reachable, the solution found by the planning graph analysis is not necessarily the plan the designer intends to use for the system, although such a solution has a minimum number of steps for achieving the goal [5].

Since the reachability analysis requires the specification of an initial state, the plan analysis algorithm applies to concrete plan invocations, which sounds more like plan testing.

4.3 An Example

For the initial marking and the goal marking in last section, the goal marking can be reached by executing plan move-blocks defined as follows:

```

plan move-blocks {
    r1r2parallelmove1(a, 4, c, 5, b, 1, c, 2),
    r1r2move(c, 5, a, 4),
    r1r2move(c, 2, c, 5),
    r1r2parallelmove2(a, 3, b, 1, b, 6, c, 2)
}
plan r1move1(?x1, ?x2, ?y1, ?y2) {
    r1unstack(?x1, ?x2, ?y1, ?y2), r1putdown(?x1, ?x2)
}
plan r2move1(?x1, ?x2, ?y1, ?y2){
    r2unstack(?x1, ?x2, ?y1, ?y2), r2putdown(?x1, ?x2)
}
plan r1move2(?x1, ?x2, ?y1, ?y2){
    r1pickup (?x1, ?x2), r1stack(?x1, ?x2, ?y1, ?y2)
}
plan r2move2(?x1, ?x2, ?y1, ?y2) {
    r2pickup (?x1, ?x2), r2stack(?x1, ?x2, ?y1, ?y2)
}
plan r1r2move (?x1, ?x2, ?y1, ?y2){

```

```

    r1r2unstack(?x1, ?x2), r1r2stack(?x1, ?x2, ?y1, ?y2)
}
plan r1r2parallelmove1(?x1,?x2,?y1,?y2, ?x3,?x4,?y3,?y4){
    r1move1 (?x1,?x2,?y1,?y2) |
    r2move1 (?x3,?x4,?y3,?y4)
}
plan r1r2parallelmove2(?x1,?x2,?y1,?y2,?x3,?x4,?y3,?y4){
    r1move2(?x1, ?x2, ?y1, ?y2) |
    r2move2(?x3, ?x4, ?y3, ?y4)
}

```

In above plans, $r1r2parallelmove_1$ consists of parallel plan invocations $r1move_1 (?x1,?x2,?y1,?y2)$ and $r2move_1 (?x3,?x4,?y3,?y4)$, whereas $r1r2parallelmove_2$ consists of parallel plan invocations $r1move_2 (?x1,?x2,?y1,?y2)$ and $r2move_2 (?x3,?x4,?y3,?y4)$. The parallel execution depends on the system state and the actual parameters (e.g. $?x1$ and $?x2$). In the given example, parallel actions are possible. $r1move_1$ and $r2move_1$ are independent because $r1unstack(?x1, ?x2, ?y1, ?y2)$ and $r1putdown(?x1, ?x2)$ are independent of $r2unstack(?x1, ?x2, ?y1, ?y2)$ and $r2putdown(?x1, ?x2)$. Similarly, $r1move_2$ and $r2move_2$ are independent. Therefore, both $r1r2parallelmove_1$ and $r1r2parallelmove_2$ are well-defined parallel plans.

5. Discussion

Although multi-agent plans can be viewed as a computational simulation of shared mental model of team members for effective teamwork, predefined agent plans are less flexible, e.g. for dealing with unexpected events in dynamic, non-deterministic environments. As a matter of fact, it is possible to enhance our plan analysis algorithm to handle embedded dynamic planning in plan specifications. The basic idea is to introduce a new clause ‘achieve <agent-group> <sub-goal>’ to the process in Definition 1 in subsection 4.1. This clause means the agents in <agent-group> are responsible for achieving the goal state <sub-goal> from the current state. From the viewpoint of plan analysis, this is similar to plan invocation. The difference is that the plan implied by a dynamic planning clause is not predefined, but dynamic generated.

The dynamic planning can be realized by following three steps: 1) Construct a PrT net in terms of the capabilities of the agents in the <agent-group>. This net is usually a part of the whole system model; 2) Obtain the current state, i.e. the state at the point where the dynamic planning is specified. This state determines the initial state of the PrT net constructed in the previous step; 3) Apply the planning graph analysis to the PrT net constructed in the first step and then get a firing sequence (i.e. a dynamic plan) if

the sub-goal is reachable from the current state. From the perspective of plan analysis, if the dynamic planning statement is contained in a parallel structure, we have to check to see if the dynamic plan (similar to a plan invocation) is independent of the actions in all other parallel branches. To do so, the dynamic plan can be unfolded into strictly sequential firings according to theorem 2 in subsection 4.2. From the perspective of dynamic plan execution, however, we may convert the dynamic plan into a plan with more complex and explicit parallel structures at runtime. As we have shown in [15], this will reduce communication among the agents and achieve better performance.

6. Related Work

Historically, the most common and familiar formalisms for multi-agent systems are logic theories. Temporal and modal logics are suitable for formally defining the semantics of multi-agent theories and languages. For example, modal logic has been used to characterize the joint intention theory [16], the belief-desire-intention (BDI) model [17], shared plans [3], planned team activity [1], agent-oriented programming [18], and representation of beliefs [19]. Logic theories are also a major means for system verification through either theorem-proving (e.g. [20-22]) or model checking (e.g. [23]). For instance, a temporal belief logic has been proposed to axiomatize the properties of the multi-agent programming language Concurrent METATEM [20, 21]. This language has a simple logical semantics, closely related to rule-based systems. For more complex agents, however, an axiomatization is not so straightforward, and capturing the semantics of concurrent execution of agents is not easy [24]. A similar approach using temporal multi-epistemic logic [22] is investigated for DESIRE [25, 26], a modeling framework for compositional specification of multi-agent systems. The semantics and verification proofs of DESIRE specifications can be formalized in three-valued temporal multi-epistemic logic [22]. Generally, axiomatic verification is inherently limited due to the well-known difficulty in automatic theorem-proving. Verification by model checking has the advantage over theorem-proving in complexity [24, 27]. The basic idea is to model a multi-agent system as a Kripke structure and to check system properties represented as formulas against the Kripke model. Since Kripke structures are less expressive with respect to high-level system specification, a front-end formalism (e.g. modal logic) instead of a Kripke structure is often expected for the specification of a multi-agent system. In [28], the algorithm for model checking BDI systems takes a logical model for their propositional BDI logic and a formula of their language, and determines whether the formula is valid in the model. Despite the inclusion of BDI modalities into the CTL branching time framework, it is claimed that the algorithm is still quite efficient. Similar algorithms have been reported for BDI-like logic [23]. Unfortunately, as stated in [24], it is not clear yet how to find the logical model for the front-end BDI logic because there is no clear relationship between such front-

end BDI logic and concrete computational models used to implement agents. A logic theory is a property-oriented formalism, and thus more suitable for specifying static properties of a multi-agent system [29].

Petri nets are a well-known model-oriented formal method, which is suitable for modeling dynamic behaviors of a system. As a major type of high-level Petri nets, colored Petri nets [30] have been used for modeling multi-agent systems [31] and agent interactions [32]. The approach of Agent-Oriented Colored Petri nets [33] redesigns Shoham's paradigm of agent-oriented programming [18] by means of object-oriented colored Petri nets. [34] models the multi-agent prey/predator game in terms of concurrent cooperative objects (COO). The behavior of an object and its cooperation with others are described by a Petri net with objects, an extension of Petri nets in which tokens are data structures (like colored Petri nets). The COO formalism provides an algorithm for gathering the behavior of all objects of a system into a single net. The behavior of this synthetic net is equivalent to the behavior of the whole system. [35] extends G-net to support inheritance modeling of agent classes in multi-agent systems. This extension provides a clean interface between agents with asynchronous communication ability and supports formal reasoning. [36] specifies a multi-agent system for resource allocation problems using Concurrent Object-Oriented Petri Nets (CO-OPN). Agents and agent societies are defined in the object-oriented paradigm. [32] uses colored Petri nets to model and analyze the social system (allocation, coordination, cooperation etc.) of an agent for a multi-agent based intelligent learning environment. The common problem with these approaches is that performing reachability analysis through occurrence graphs (if supported) is highly complex.

As another major type of high-level Petri nets, PrT nets have been employed to model multi-agent planning domains. Murata et al [8] used a simplified version of PrT nets to address the planning issue. Based on the traditional means-end analysis, the planning algorithm conducts bi-directional search through the state space (i.e. a combination of forward-chaining search from the initial state and backward chaining search from the goal state). The backward chaining search requires complete specification of the goal state. Although a planning problem is essentially a reachability problem, the means-end analysis is ineffective for verifying PrT nets. It is also worth mentioning the reachability analysis techniques for general PrT nets. Relying on the existence of internal symmetries in high-level Petri nets, symbolic reachability graphs [10] introduce a level of symbolic treatment of markings under conditions where internal symmetries hold. Parameterized reachability trees [9] exploit parameterized markings as a means for folding reachability trees of PrT nets so that a number of concrete states can be condensed into a generic state. Unfortunately, most of these approaches have not shown much promise in practical analysis of systems modeled in high-level Petri nets because they still suffer from high complexity. In particular, parameterized reachability trees may cost extra time to instantiate the parameters for generating successor states and for comparing states, although it does save a large amount of space.

7. Conclusion

We have presented the PrT net approach to modeling and analyzing multi-agent behaviors defined by agent capabilities and plans. The totality of agent capabilities specified by preconditions and post-conditions determines the behavior model of what the agents as a group can do. Multi-agent plans are treated as a description of the process for moving from some start state, through this behavior model, to a goal state. The contributions of this paper include: 1) the demonstration that PrT nets are an effective formalism for the formal modeling of multi-agent behaviors and can capture the concurrency among agents; 2) the planning graph based reachability analysis for PrT models. Since the complexity of constructing planning graphs is polynomial, the reachability analysis of multi-agent models is much more efficient than traditional approaches, such as parameterized reachability trees [9] and symbolic reachability graphs [10]. Obviously, this technique is also applicable to the analysis of general distributed concurrent systems modeled by PrT nets, and 3) the algorithm for the formal analysis of hierarchical multi-agent plans with explicit parallel actions. This provides a useful means for static analysis of multi-agent plan design.

In this paper, a single PrT net is used to model the whole multi-agent problem, including agent actions as well as their environment. How to model dynamic multi-agent environments and how to support compositional analysis of complex multi-agent problems are two major concerns of our future work. Dynamic planning is one approach to dealing with contingency situations in dynamic environments. We are currently incorporating the idea of dynamic planning into the plan analysis algorithm, and investigating how to analyze more complex plans specified in MALLETT, a multi-agent logic language for encoding teamwork knowledge [2]. The next step we are planning on is to introduce explicit timing constraints into the specification of agent capabilities. Although the planning graph analysis has been improved to handle time durations [37], it remains to be seen how complex it is to enable the planning graph analysis to deal with timing constraints represented by time intervals. The introduction of such timing constraints in PrT nets also offers a potential for compositional analysis.

Acknowledgement

This work was supported in part by AFOSR (MURI) grant #F49620-00-1-0326.

References

- [1] D. Kinny, M. Ljungberg, A. S. Rao, E. A. Sonenberg, G. Tidhar, and E. Werner, "Planned Team Activity," *Proceedings of the fourth European Workshop on Modeling Autonomous Agents in a Multi-Agent World (MAAMAW'92)*, 1992.
- [2] J. Yen, J. Yin, T. R. Ioerger, M. S. Miller, D. Xu, and R. A. Volz, "CAST: Collaborative Agents for Simulating Teamwork," *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI'2001)*, Seattle, WA, pp. 1135-1142, 2001.
- [3] B. Grosz and S. Kraus, "Collaborative Plans for Complex Group Actions," *Artificial Intelligence*, vol. 86, no. 2, pp. 269-357, 1996.
- [4] M. J. Katz and J. S. Rosenschein, "Plans for Multiple Agents," in *Distributed Artificial Intelligence*, vol. II, L. Gasser and M. N. Huhns, Eds. London: Pitman/Morgan Kaufmann Publishers, 1989, pp. 197-228.
- [5] Blum and M. L. Furst, "Fast Planning through Planning Graph Analysis," *Artificial Intelligence*, vol. 90, pp. 281-300, 1997.
- [6] H. J. Genrich, "Predicate/Transitions Nets," in *Petri Nets: Central Models and Their Properties: Advances in Petri Nets*, vol. 254, *Lecture Notes in Computer Science*: Springer-Verlag, 1987, pp. 207-247.
- [7] H. J. Genrich and K. Lautenbach, "System Modeling with High-Level Petri Nets," *Theoretical Computer Science*, vol. 13, pp. 109-136, 1981.
- [8] T. Murata, P. C. Nelson, and J. Yim, "Predicate-Transition Net Model for Multiple Agent Planning," *Information Science*, vol. 57/58, pp. 361-384, 1991.
- [9] M. Lindqvist, "Parameterized Reachability Trees for Predicate/Transition Nets," *Proceedings of the 11th International Conference on Applications and Theory of Petri Nets*, Paris, 1990.
- [10] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Hadda, "On Well-Formed Coloured Nets and Their Symbolic Reachability Graph," *Proceedings of the 11th International Conference on Applications and Theory of Petri Nets*, Paris, 1990.
- [11] M. R. Genesereth and N. J. Nilsson, *Logical Foundations of Artificial Intelligence*. Los Altos: Morgan Kaufmann, 1987.
- [12] R. Fikes and N. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence*, vol. 2, pp. 189-208, 1971.
- [13] G. S. Tjaden and M. J. Flynn, "Detection and Parallel Execution of Independent Instructions," *IEEE Transactions Software Engineering*, vol. C-19, no. 10, pp. 889-895, 1970.
- [14] C. Knoblock, "Generating Parallel Execution Plans with a Partial-Order Planner," *Proceedings of Artificial Intelligence Planning and Scheduling (AIPS'94)*, Chicago, pp. 98-103, 1994.

- [15] D. Xu, R. A. Volz, and T. R. Ioerger, "Generating Parallel Plans Based on Planning Graph Analysis of Predicate/Transition Nets," *Proceedings of the 2002 International Conference on Artificial Intelligence (IC-AI'02)*, Las Vegas, pp. 440-446, 2002.
- [16] P. R. Cohen and H. J. Levesque, "Intention Is Choice With Commitment," *Artificial Intelligence*, vol. 42, no. 3, pp. 213-261, 1990.
- [17] A. S. Rao and M. P. Georgeff, "Modeling Rational Agents within a BDI-Architecture," *Proceedings of the 2nd International Conference on Principles on Knowledge Representation and Reasoning*, Cambridge, MA, pp. 473-484, 1991.
- [18] Y. Shoham, "Agent-Oriented Programming," *Artificial Intelligence*, vol. 60, pp. 51-92, 1993.
- [19] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi, *Reasoning About Knowledge*. Cambridge, MA: The MIT Press, 1995.
- [20] M. Fisher and M. Wooldridge, "On the Formal Specification and Verification of Multi-Agent Systems," *International Journal of Cooperative Information Systems*, vol. 6, no. 1, 1997.
- [21] M. Fisher and M. Wooldridge, "Specifying and Verifying Distributed Intelligent Systems," *Progress in Artificial Intelligence - Sixth Portuguese Conference on Artificial Intelligence*, 1993.
- [22] J. Engelfriet, C. M. Jonker, and J. Treur, "Compositional Verification of Multi-Agent Systems in Temporal Multi-Epistemic Logic," in *Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Language (ATAL'98)*, LNAI 1555: Springer-Verlag, 1999.
- [23] M. Benerecetti, F. Giunchiglia, and L. Serafini, "A Model Checking Algorithm for Multiagent Systems," in *Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL'98)*, LNAI 1555: Springer-Verlag, 1999.
- [24] M. Wooldridge and P. Ciancarini, "Agent-Oriented Software Engineering: The State of the Art," in *Handbook of Software Engineering and Knowledge Engineering*: World Scientific Publishing Co., 2001.
- [25] F. Brazier, B. Dunin-Keplicz, N. R. Jennings, and J. Treur, "Formal Specifications of Multiagent Systems: A Real-World Case Study," *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*, pp. 25-32, 1995.
- [26] F. Brazier, B. Dunin-Keplicz, N. R. Jennings, and J. Treur, "Desire: Modeling Multiagent Systems in a Compositional Formal Framework," *International Journal of Cooperative Information Systems*, vol. 6, no. 1, pp. 67-94, 1997.
- [27] J. Y. Halpern and M. Y. Vardi, "Modeling Checking vs. Theorem Proving: A Manifesto," *Proceedings of KR-91*. Also in Lifshitz V. *Artificial Intelligence and Mathematical Theory of Computation, Papers in Honor of John McCarthy*, San Diego, 1991.

- [28] A. Rao and M. P. Georgeff, "A Model-Theoretic Approach to the Verification of Situated Reasoning Systems," *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI'93)*, Chambrey, France, pp. 318-324, 1993.
- [29] J. Wing, "A Specifier's Introduction to Formal Methods," *IEEE Computer*, vol. 23, no. 9, pp. 8-24, 1990.
- [30] K. Jensen, *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, vol. 26: Springer-Verlag, 1992.
- [31] J. Ferber, *Multi-Agent Systems*. Reading, MA: Addison-Wesley, 1999.
- [32] M. Miranda and A. Perkusich, "Modeling and Analysis of a Multi-Agent System Using Colored Petri Nets," <http://www.dee.ufpb.br/~perkusich/publicacoes/wpndisd99.ps.gz>, 1999.
- [33] D. Moldt and F. Wienberg, "Multi-Agent-Systems Based on Colored Petri Nets," *Proceedings of the 18th International Conference on Applications and Theory of Petri Nets (ICATPN'97)*, Toulouse, France, pp. 82-101, 1997.
- [34] W. Chainbi, C. Hanachi, and C. Sibertin-Blanc, "The Multi-agent Prey/Predator Problem: A Petri Net Solution," *Proceedings of the Symposium on Discrete Events and Manufacturing Systems, CESA'96 IMACS Multiconference Computational Engineering in Systems Application*, 1996.
- [35] H. Xu and S. M. Shatz, "A Framework for Modeling Agent-Oriented Software," *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS)*, Phoenix, Arizona, 2001.
- [36] D. Friha, D. Buchs, and P. Berry, "Multi-Agent System Specification using CO-OPN," University of Geneva, Technical Report CUI No. TR-93/80 1991.
- [37] S. Smith and D. Weld, "Temporal Planning with Mutual Exclusion Reasoning," *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pp. 326-333, 1999.