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

Dianxiang Xu, Richard Volz, Thomas Ioerger
Department of Computer Science
301 H. R. Bright Building
Texas A&M University
College Station, TX 77843, 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
In a multi-agent system, how agents accomplish a goal task is usually specified by multi-agent plans built from basic actions (e.g. operators) of which the agents are capable. A critical problem with such an approach is how can the designer make sure the plans are reliable. To tackle this problem, this paper presents a formal approach for modeling and analyzing multi-agent behaviors using Predicate/Transition (PrT) nets, a high-level formalism of Petri nets. We construct a multi-agent model by representing agent capabilities as transitions. To verify a multi-agent PrT model, we adapt the planning graphs as a compact structure for the reachability analysis. We also demonstrate that, based on the PrT model, whether parallel actions specified in multi-agent plans can be executed in parallel and whether the plans guarantee the achievement of the goal can be verified by analyzing the dependency relations among the transitions.

Categories and Subject Descriptors
D.2 [**Software Engineering**]: Design Tools and Techniques – *Petri nets.* Software/Program Verification – *Formal methods*. I.2 [**Artificial Intelligence**]: Distributed Artificial Intelligence – *Multiagent systems, Intelligent agents.*

General Terms
Design, Verification.

Keywords
Formal methods, predicate/transition nets, multiagent systems, verification, Petri nets.

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 or operators [1-4]. A major motivation of this approach is that for a complex multi-agent system it is hard for a general-purpose planning algorithm to generate a plan that is optimal in certain sense. The measurement of optimum may vary from applications to applications. 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 plans may achieve better performance by taking into consideration specific application requirements. However, a critical problem is how a plan designer can make sure the specified plans are reliable? 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 for modeling and analyzing multi-agent behaviors using Predicate/Transition (PrT) nets [6], a high-level formalism of Petri nets [7]. 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 PrT nets are well suited for modeling multi-agent systems by representing agent capabilities as transitions. In particular, the reachability problem of a simplified PrT net model can be analyzed efficiently using a compact structure called planning graphs [5]. We also demonstrate that, based on the PrT model, whether specified parallel actions can be executed in parallel and whether the parallel plans guarantee the achievement of the goal can be verified by analyzing the dependency relations among the transitions.

The rest of this paper is organized as follows. Section 2 briefly reviews related work. Section 3 introduces PrT nets and shows how to built multi-agent models using PrT nets. Section 4 describes the reachability analysis of PrT nets using planning graphs. Section 5 presents how to verify parallel plans. Section 6 contains concluding remarks.

2. RELATED WORK
Historically, the most common and familiar formalisms for multi-agent systems are logic theories. Temporal or modal logic is suitable for formally defining the semantics of multi-agent

SEKE '02, July 15-19, 2002, Ischia, Italy.

theories and languages. For example, modal logic has been used to characterize the joint intention theory [8], shared plans [3], planned team activity [1], agent-oriented programming [9], etc. Logic theories are also a major means for system verification through theorem-proving or model checking [10]. Generally, the theorem-proving approach is inherently limited due to the well-known difficulty. Verification by model checking [11] has the advantage over theorem-proving in complexity. 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.

Petri nets, as a well-known model-oriented formal method, are more suitable for modeling dynamic system behaviors [12]. The approach of Agent-Oriented Colored Petri nets [13] redesigns Shoham's paradigm of agent-oriented programming [9] by means of object-oriented colored Petri nets. [14] extends G-net to model inheritance of agent classes in multi-agent systems, which provides a clean interface between agents with asynchronous communication ability and supports formal reasoning. [15] specifies a multi-agent system for resource allocation problems using concurrent object-oriented Petri nets. The common issues with the above approaches are: 1) they suffer from high complexity of reachability analysis through occurrence graphs (if supported), and 2) they are not concerned with agent plans.

As another major type of high-level Petri nets, PrT nets have been employed to model multi-agent planning domains. Murata et al [16] use a simplified version of PrT nets for multi-agent planning. 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). Although a planning problem is in essence a reachability problem, the means-end analysis is ineffective for the verification of PrT nets. It is also worth mentioning the reachability analysis techniques for general PrT nets. Parameterized reachability trees [17] 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. This approach has not shown much help in practical analysis of systems due to the fact that 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, though it does save a large amount of space.

3. MULTIAGENT MODELING

This section gives an introduction to the PrT nets to be used, and shows how to construct multi-agent models using PrT nets.

3.1 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 ($P \cap T = \varnothing$, $P \cup T \neq \varnothing$), and $F \subseteq (P\times T) \cup (T\times P)$ is 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 <¢>.

(4) φ is a mapping from a set of inscription formulae to transitions. 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 or current 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 above PrT nets have simplified general PrT nets [6] in two ways: 1) an arc labeling is a set of tuples (labels) $\{l_i\}$ rather than a formal sum $c_1l_1+c_2l_2+..+c_nl_n$ (i.e. coefficient or arc weight c_i of arc label l_i is 1 for all $1\le i\le n$). 2) Accordingly, the marking of a predicate under a certain state is a set of tokens (i.e. items in [6]) instead of a formal sum of tokens. These nets, similar to those in [16], 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 easily handled. 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 postcondition 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_0 if there is a substitution θ such that $l/\theta \in M_0(p)$ 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 ${}^\bullet t\theta = \{l/\theta: l\in L(p,t)$ for any $p\in {}^\bullet t\}$ and $t\theta^\bullet = \{l/\theta: l\in L(t,p)$ 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 $\gamma = \{t_1\theta_1, t_2\theta_2 \ldots t_k\theta_k\}$ $(k>0)$ under M_0, γ is said to be concurrent firings if $({}^\bullet t_i\theta_i \cup t_i\theta_i^\bullet) \cap ({}^\bullet t_j\theta_j \cup t_j\theta_j^\bullet) = \varnothing$, and $M_0 \cap t_i\theta_i^\bullet = \varnothing$ for any $1\le i, j \le k$ and $i\neq j$. After the firing of γ (called a step), we reach a new marking $M_1 = M_0 - \{{}^\bullet t_i\theta_i: 1\le i\le k\} \cup \{t_i\theta_i^\bullet: 1\le i\le k\}$. A firing sequence of steps that reaches marking M_n is denoted as $\gamma_1\gamma_2 \ldots\gamma_n$ or $M_0\gamma_1M_1\gamma_2M_2 \ldots\gamma_nM_n$, where M_i is the marking after step γ_i $(1\le i\le n)$. A marking M is said to be reachable from M_0 if there is such a sequence of steps that transforms M_0 to M.

3.2 Modeling Multi-Agents

For a multi-agent system, the PrT model can be constructed by focusing on the actions of which the agents are capable. These actions are represented by transitions with preconditions, post-conditions and inscriptions.

As a case study, this paper extends the well-known blocks world problem by introducing multiple agents with different capabilities and different types of blocks. This extension facilitates the comparison between two cases: 1) there are multiple agents that can work in parallel, and 2) there is only one agent who is capable of moving any type of blocks 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 $x_2 \in \{1, 2, 3,\ldots\}$ are its type and identification number, respectively. Two agents, r_1 and r_2 are supposed to cooperatively move stacks of blocks. Their capabilities are different: r_1 alone can move a-type blocks, whereas r_2 alone can only move b-type blocks. As a team, r_1 and r_2 can jointly move c-type blocks.

The transitions for the extended blocks world problem are specified in TABEL 1. From Table 1, we can see that transitions bear much similarity to STRIPS operators for planning problems. The main differences include: 1) transitions are associated with logical formulae, and 2) a precondition of a transition becomes false (like a delete effect) unless it is also a post-condition. 3) tokens may represent structured date items. As a matter of fact, a set of STRIPS operators can be converted into a PrT net [16].

It is worth mentioning, although a planning problem is in essence 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 deadlock-freedom, liveness, etc.

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

Transition	*Precondition predicates & arc labels*	*Post-condition predicates & arc labels*	*Inscription*
r1r2pickup	ontable <?x1,?x2>, clear<?x1,?x2>, r1handempty <¢>, r2handempty <¢>	r1holding <?x1,?x2>, r2holding <?x1, ?x2>	equal(?x1,c)
r1pickup	ontable<?x1,?x2>,clear<?x1,?x2> r1handempty <¢>	r1holding<?x1,?x2>	equal(?x1,a)
r2pickup	ontable<?x1,?x2>,clear<?x1,?x2>, r2handempty <¢>	r2holding<?x1,?x2>	equal(?x1,b)
r1r2putdown	r1holding<?x1,?x2>, r2holding<?x1,?x2>	ontable<?x1,?x2>, clear<?x1,?x2>, r1handempty<¢>, r2handempty<¢>	equal(?x1,c)
r1putdown	r1holding<?x1,?x2>	ontable<?x1,?x2>,clear<?x1,?x2>, r1handempty<¢>	equal(?x1,a)
r2putdown	r2holding<?x1,?x2>	ontable<?x1,?x2>,clear<?x1,?x2>, r2handempty<¢>	equal(?x1,b)
r1r2stack	r1holding<?x1,?x2>, r2holding<?x1,?x2>, clear<?y1,?y2>	r1handempty<¢>, r2handempty<¢>, on<?x1,?x2,?y1,?y2>, clear<?x1,?x2>	equal(?x1,c)
r1stack	r1holding<?x1,?x2>,clear<?y1,?y2>	r1handempty<¢>, on<?x1,?x2,?y1,?y2>,clear<?x1,?x2>	equal(?x1,a)
r2stack	r2holding<?x1,?x2>,clear<?y1,?y2>	r2handempty<¢>, on<?x1,?x2,?y1,?y2>,clear<?x1,?x2>	equal(?x1,b)
r1r2unstack	r1handempty<¢>, r2handempty<¢>, clear<?x1,?x2>,on<?x1,?x2,?y1,?y2>	r1holding<?x1,?x2>, r2holding<?x1,?x2>, clear<?y1,?y2>	equal(?x1,c)
r1unstack	r1handempty<¢>,clear<?x1,?x2>, on<?x1,?x2,?y1,?y2>	r1holding<?x1,?x2>, clear<?y1,?y2>	equal(?x1,a)
r2unstack	r2handempty<¢>,clear<?x1,?x2>, on<?x1,?x2,?y1,?y2>	r2holding<?x1,?x2>, clear<?y1,?y2>	equal(?x1,b)

4. REACHABILITY ANALYSIS OF MULTIAGENT MODELS

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

4.1 Overview of Planning Graph Analysis

Planning graph analysis, originally developed in Graphplan [5], solves STRIPS planning problems in which operators have preconditions, add-effects and delete-effects, all represented as conjuncts of (parameterized) propositions. Empirical studies have demonstrated its practical value in a dozen of benchmark planning problems.

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.

4.2 Reachability Analysis of PrT Nets

Planning graph analysis can be easily 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 but 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 section 3.1. 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 needs to be enhanced. We think two actions are mutually exclusive (i.e. they can not be concurrent) if they have a common effect (postcondition).

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. token level i (=0) is the initial state;

Step 2: WHILE (solution extraction fails and level i (if >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 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:

- l/θ in contained in token level i for any label $l \in L(p,t)$ for all $p \in {}^\bullet t$;
- $\varphi(t)$ evaluates true w.r.t. θ; and
- there does not exist p_1, p_2, l_1 and l_2 such that $l_1 \in L(p_1,t)$, $l_2 \in L(p_2,t)$, ($l_1 \neq l_2$ 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 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 token in level i+1.

Step 2.4: For any pair of firings $t_1\theta_1$ and $t_2\theta_2$ (including no-op firings) in level i+1, they 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: For any pair of tokens τ_1 and τ_2 in token level i+1, they 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/postcondition of one firing contains a token in the precondition/postcondition 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 not reachable (the solution extraction fails and the current firing/token level and mutual exclusion relations are identical to the previous firing/token level and mutual exclusion relations).

Procedure 2 (solution extraction)

Step 1: If there is a token in the given goal is not contained in the token level newly created or there are two tokens in the goal are mutually exclusive in the token level, 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 level newly generated

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 non-mutually-exclusive 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 hashtable) 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 only applicable (sound and compete) for the 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 indicated by the safeness assumption of PrT nets. Note that for simplicity safeness is just assumed other than specified by the definition of fireability. To make sure a PrT is safe, we may explicitly impose $M_0 \cap t\theta^{\bullet}=\varnothing$ as an additional necessary condition on the enabledness (i.e. postconditions as inhibitor conditions). However, it is not trivial to extend the rules for the reasoning and propagation of mutual exclusion relations. We would like to address this issue in a subsequent paper. In conclusion, the practical value of planning graph analysis is that for a large class of parallel tasks the solution can often be found quickly if a goal is reachable. It does not change the hard nature of reachability problems, though.

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. Suppose the initial marking of the PrT model for the multi-agent blocks world problem is:

{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: <¢>, r2handempty: <¢>
}

and the goal marking 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)
}

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 firings within step S1 (S2, S7, S8) can be performed in arbitrary order. To illustrate the algorithm's efficiency for parallel systems, we briefly compare the above example with the classical case with a single agent (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 our algorithm to find the above solution (Environment: Windows 2000, 800MHZ, 256MB). For the single-agent case with the same initial state and goal state, it takes 0.766 seconds to find the 12-step solution. This indicates that, although the size of action space is increased, the introduction of parallel actions accelerates the reachability analysis.

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 can be achieved by enhancing the first step in the procedure of solution extraction.

5. VERIFYING MULTIAGENT 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 verification of parallel plans, regarding whether specified parallel actions can be actually executed in parallel, and whether they achieve the given goal.

5.1 Plan Specification

Multi-agent plans are in essence the specification of 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 as formal parameters that appear in the labels of the arcs connected with 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 check to see if it is an occurrence sequence that transforms the initial state to the goal state in the PrT model. Nevertheless, a plan specification language usually provides complex constructs,

such as parallel, plan invocation (to form a plan hierarchy), condition, loops etc. In this paper, we focus on parallel actions in plan hierarchy. Other control structures like condition and loop can be easily incorporated. In the following, we formally define multi-agent plans.

Definition 1. A plan is a structure <name, arguments, process>, where name, arguments and process are the plan identifier, the formal parameters and the process. A process is either an action (transition and substitution) or a plan invocation, or a finite sequence (denoted as ',') of processes, or a parallel (denoted as '|') composition of a finite number of processes. Formally, the BNF-style notation of process 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 specify the following plan with two sequential actions for the extended blocks world problem.

plan $r_1move_1(?x_1, ?x_2, ?y_1, ?y_2)$ {
$r_1unstack(?x_1, ?x_2, ?y_1, ?y_2)$, $r_1putdown(?x_1, ?x_2)$
}

5.2 Plan Verification

Although agent plans look like procedural programs, plan verification is made possible and tractable due to 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 [18]. Knoblock has adapted these dependency types for generating parallel execution plans with a partial order planner [19]. 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 interplay between a PrT model and its 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. ${}^\bullet t_1\theta_1 \cap {}^\bullet t_2\theta_2 = \varnothing$. 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. $({}^\bullet t_1\theta_1 \cup t_1\theta_1{}^\bullet) \cap ({}^\bullet t_2\theta_2 \cup t_2\theta_2{}^\bullet) = \varnothing$. In other words, two firings can be executed in parallel only if they are independent. In the following definitions, we generalize this requirement.

Definition 2. Given two actions $t_1\theta_1$ and $t_2\theta_2$. If $({}^\bullet t_1\theta_1 \cup t_1\theta_1{}^\bullet) \cap ({}^\bullet t_2\theta_2 \cup t_2\theta_2{}^\bullet) = \varnothing$, 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 $\delta_1 = t_{11}\theta_{11}\ t_{12}\theta_{12}\ldots\ t_{1m}\theta_{1m}$ and $\delta_2 = t_{21}\theta_{21}\ t_{22}\theta_{22}\ldots\ t_{2n}\theta_{2n}$. If $t_{1i}\theta_{1i}$ and $t_{2j}\theta_{2j}$ are independent for any i and j ($1 \le i \le m$, $1 \le j \le n$), 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 $\delta = t_1\theta_1\ t_2\theta_2\ldots\ t_m\theta_m$, 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 tokens and postcondition tokens of actions in δ. The following theorem shows the complexity of determining the dependency relation between δ_1 and δ_2 can be reduced from $O(m \times n)$ to $O(m+n)$. The proof is omitted due to the limit of paper length.

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 $({}^\bullet\delta_1 \cup \delta_1{}^\bullet) \cap ({}^\bullet\delta_2 \cup \delta_2{}^\bullet) = \varnothing$.

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.

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

In the following, we outline the recursive algorithm for checking parallel actions 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 state of executing $\rho(\theta)$.

Step1: If the process is a sequence structure of the form $\rho_1(\nu_1)$, $\rho_2(\nu_2),\ldots, \rho_m(\nu_m)$, Then *For* i=1 TO m DO

Case 1: $\rho_i(\nu_i)$ is an action, i.e. ρ_i is a transition.
If $\rho_i(\nu_i/\theta)$ is not firable under M
Then plan ρ is not feasible
Else $M_i = M_{i-1} - {}^\bullet\rho_i(\nu_i/\theta) \cup \rho_i(\nu_i/\theta)^\bullet$

Case 2: $\rho_i(\nu_i)$ is a plan invocation.
Recursive call with input (M_{i-1}, ρ_i, ν_i).
If $\sigma(M_{i-1}, \rho_i, \nu_i)$ is empty
Then return $\sigma(M_0, \rho, \theta)$ = empty
Else $M_i = M'(M_{i-1}, \rho_i, \nu_i)$.

Step2: If the process is a parallel structure of the form $\rho_1(\nu_1) \mid \rho_2(\nu_2) \mid \ldots \mid \rho_m(\nu_m)$
Step2.1: For i=1 TO m DO
Recursive call with input $(M_0, \rho_i, \nu_i/\theta)$;
1) Let $\sigma_i = \sigma(M_0, \rho_i, \nu_i/\theta)$;

2) If σ_i is empty
Then return $\sigma(M_0, \rho, \theta)$ = empty
3) If i=1
Then let $M_1=M'(M_0, \rho_i, \nu_i/\theta)$,

Step 2.2: If there exist i,j ($1 \le i, j \le m$ and $i \ne j$) such that $({}^\bullet\sigma_i \cup \sigma_i^\bullet) \cap ({}^\bullet\sigma_j \cup \sigma_j^\bullet) \neq \varnothing$ Then $\rho_l(\nu_i/\theta)$ and $\rho_l(\nu_i/\theta)$ cannot be executed in parallel, and return $\sigma(M_0, \rho, \theta)$ = empty;

Step2.3: For i=2 TO m DO
Let $\sigma_i = t_1\theta_1, t_2\theta_2, \ldots, t_k\theta_k$
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 $\sigma(M_0, \rho, \theta) = \sigma_1 + \sigma_2 + \ldots + \sigma_m$, let $M'(M_0, \rho, \theta) = M_1$, and then return.

Whether a given goal G is reachable through the invocation of plan ρ with actual parameters θ from some initial state M_0 can be simply verified by checking whether marking G is included in $M'(M_0, \rho, \theta)$ or formula G evaluates true with respect to $M'(M_0, \rho, \theta)$.

In summary, the plan verification can report following three types of feedback to the designer: 1) A specific action cannot be performed if its preconditions are 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 on plan specifications as well as definitions of agent capabilities.

If a given goal is unreachable by specified plans, the designer may 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].

5.3 An Example

Given the initial marking and the goal marking described in last section for the extended blocks problem, the goal can be reached by invoking plan move-blocks defined below:

plan move-blocks {
$r_1r_2$parallelmove$_1$(a, 4, c, 5, b, 1, c, 2),
$r_1r_2$move(c, 5, a, 4),
$r_1r_2$move(c, 2, c, 5),
$r_1r_2$parallelmove$_2$(a, 3, b, 1,b, 6, c, 2)
}

The plans invoked in move-blocks are specified as follows:

plan $r_1$move$_1$($?x_1$, $?x_2$, $?y_1$, $?y_2$) {
$r_1$unstack($?x_1$, $?x_2$, $?y_1$, $?y_2$), $r_1$putdown($?x_1$, $?x_2$)
}
plan $r_2$move$_1$($?x_1$, $?x_2$, $?y_1$, $?y_2$){
$r_2$unstack($?x_1$, $?x_2$, $?y_1$, $?y_2$), $r_2$putdown($?x_1$, $?x_2$)
}
plan $r_1$move$_2$($?x_1$, $?x_2$, $?y_1$, $?y_2$){
$r_1$pickup ($?x_1$, $?x_2$), $r_1$stack($?x_1$, $?x_2$, $?y_1$, $?y_2$)
}
plan $r_2$move$_2$($?x_1$, $?x_2$, $?y_1$, $?y_2$) {
$r_2$pickup ($?x_1$, $?x_2$), $r_2$stack($?x_1$, $?x_2$, $?y_1$, $?y_2$)
}
plan $r_1r_2$move ($?x_1$, $?x_2$, $?y_1$, $?y_2$){
$r_1r_2$unstack($?x_1$, $?x_2$), $r_1r_2$stack($?x_1$, $?x_2$, $?y_1$, $?y_2$)
}
plan $r_1r_2$parallelmove$_1$($?x_1$,$?x_2$,$?y_1$,$?y_2$, $?x_3$,$?x_4$,$?y_3$,$?y_4$){
$r_1$move$_1$ ($?x_1$,$?x_2$,$?y_1$,$?y_2$) |
$r_2$move$_1$ ($?x_3$,$?x_4$,$?y_3$,$?y_4$)
}
plan $r_1r_2$parallelmove$_2$($?x_1$,$?x_2$,$?y_1$,$?y_2$,$?x_3$,$?x_4$,$?y_3$,$?y_4$){
$r_1$move$_2$($?x_1$, $?x_2$, $?y_1$, $?y_2$) |
$r_2$move$_2$($?x_3$, $?x_4$, $?y_3$, $?y_4$)
}

In above plans, $r_1r_2$parallelmove$_1$ consists of parallel plan invocations $r_1$move$_1$ ($?x_1$,$?x_2$,$?y_1$,$?y_2$) and $r_2$move$_1$ ($?x_3$,$?x_4$, $?y_3$,$?y_4$), whereas $r_1r_2$parallelmove$_2$ consists of parallel plan invocations $r_1$move$_2$ ($?x_1$,$?x_2$,$?y_1$,$?y_2$) and $r_2$move$_2$ ($?x_3$,$?x_4$, $?y_3$,$?y_4$). It is easy to verify that parallel actions in this example are feasible.

6. CONCLUSIONS

To summarize, the main contributions of this paper are: 1) the demonstration that PrT nets are an effective formalism for the formal modeling of multi-agent behaviors, 2) the planning graph based reachability analysis of PrT models, which allows for concurrent firings, and 3) formal verification of hierarchical multi-agent plans with explicit parallel actions.

One of the major criticisms about predefined agent plans is that they are less flexible, e.g. for dealing with unexpected events in dynamic, uncertain environments. In the future work, we will embed dynamic planning in plan specification and investigate the interplay between dynamic planning and plan verification.

7. ACKNOWLEDGMENTS

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

8. 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 Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'2001), Seattle, WA, 2001.

[3] B. Grosz and S. Kraus, "Collaborative Plans for Complex Group Actions," *Artificial Intelligence*, vol. 86, 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 *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] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proceedings of the Institute of Electrical and Electronics Engineers*, vol. 77, pp. 541-580, 1989.

[8] P. R. Cohen and H. J. Levesque, "Intention Is Choice With Commitment," *Artificial Intelligence*, vol. 42, pp. 213-261, 1990.

[9] Y. Shoham, "Agent-Oriented Programming," *Artificial Intelligence*, vol. 60, pp. 51-92, 1993.

[10] 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.

[11] 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.

[12] J. Wing, "A Specifier's Introduction to Formal Methods," *Computer*, pp. 8-24, 1990.

[13] D. Moldt and F. Wienberg, "Multi-Agent-Systems Based on Colored Petri Nets," Proceedings of the18th International Conference ICATPN'97, Toulouse, France, 1997.

[14] 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.

[15] D. Friha, D. Buchs, and P. Berry, "Multi-Agent System Specification using CO-OPN," University of Geneva, Technical Report CUI No. TR-93/80 Oct. 1991 1991.

[16] 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.

[17] M. Lindqvist, "Parameterized Reachability Trees for Predicate/Transition Nets," Proceedings of the 11th International Conference on Applications and Theory of Petri Nets, Paris, 1990.

[18] G. S. T. a. M. J. Flynn, "Detection and Parallel Execution of Independent Instructions," *IEEE Transactions Software Engineering*, vol. C-19, pp. 889-895, 1970.

[19] C. Knoblock, "Generating Parallel Exectuion Plans with a Partial-Order Planner," Proceedings of Artificial Intelligence Planning and Scheduling 1994, Chicago, 1994.