From: AAAI-99 Proceedings. Copyright © 1999, AAAI (www.aaai.org). All rights reserved.

Theory for Coordinating Concurrent Hierarchical Planning Agents Using Summary Information

Bradley J. Clement and Edmund H. Durfee
University of Michigan
Ann Arbor, MI 48109
{bradc, durfee}@umich.edu

Abstract

Interacting agents that interleave planning, plan coordination, and plan execution for hierarchical plans (e.g. HTNs or procedures for PRS) should reason about abstract plans and their concurrent execution before they are fully refined. Poor decisions made at abstract levels can lead to costly backtracking or even failure. We claim that better decisions require information at abstract levels that summarizes the preconditions and effects that must or may apply when a plan is refined. Here we formally characterize concurrent hierarchical plans and a method for deriving summary information for them, and we illustrate how summary conditions can be used to coordinate the concurrent interactions of plans at different levels of abstraction. The properties of summary conditions and rules determining what interactions can or might hold among asynchronously executing plans are proven to support the construction of sound and complete coordination mechanisms for concurrent hierarchical planning agents.

Introduction

The study of concurrent action in relation to planning (Georgeff 1984) has improved our understanding of how agents can reason about their interactions in order to avoid conflicts during concurrent plan execution. Conflicts can be avoided by reducing or eliminating interactions by localizing plan effects to particular agents (Lansky 1990), and by merging the individual plans of agents by introducing synchronization actions (Georgeff 1983). In fact, planning and merging can be interleaved, such that agents can propose next-step extensions to their current plans and reconcile conflicts before considering extensions for subsequent steps. By formulating extensions in terms of constraints rather than specific actions, a "least commitment" policy can be retained (Ephrati & Rosenschein 1994).

For many applications, planning efficiency can be enhanced by exploiting the hierarchical structure of planning operations. Rather than building a plan from the beginning forward (or end backward), hierarchical planners identify promising classes of long-term activities (abstract plans), and incrementally refine these to eventually converge on specific actions. Planners such as NOAH (Sacerdoti 1977) and NONLIN (Tate 1977) have this character, and are often considered instances of a class of planners called Hierarchical Task Network (HTN) planners. By exploiting the hierarchical task structure to focus search, HTN planners often converge much more quickly to effective plans. They are also becoming increasingly well understood (Erol, Hendler, & Nau 1994).

Using HTN planning for concurrently-executing agents is less well understood, however. If several HTN planning agents are each generating their own plans, how and when should these be merged? Certainly, merging could wait until the plans were fully refined, and techniques like those of Georgeff (mentioned previously) would work. But interleaving planning and merging holds greater promise for identifying and resolving key conflicts as early in the process as possible to try to avoid backtracking or failure. Such interleaving, however, requires the ability to identify potential conflicts among abstract plans.

Corkill (Corkill 1979) studied interleaved planning and merging in a distributed version of the NOAH planner. He recognized that, while most of the conditions affected by an abstract plan operator might be unknown until further refinement, those that deal with the overall effects and preconditions that hold no matter how the operator is refined can be captured and used to identify and resolve some conflicts. He recognized that further choices of refinement or synchronization choices at more abstract levels could lead to unresolvable conflicts at deeper levels, and backtracking could be necessary. Our work is directed toward avoiding such backtracking by improving how an abstract plan operator represents all of the potential needs and effects of all of its potential refinements.

Our motivation for doing this is not simply to make interleaved planning and merging with HTNs more efficient, but also to support another crucial use of HTN concepts–specifically, flexible plan execution systems such as PRS (Georgeff & Lansky 1986), RAPS

Copyright ©1999, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

This work was supported in part by NSF (IRI-9158473) and DARPA (F30602-98-2-0142).

(Firby 1989), etc., that similarly exploit hierarchical plan spaces. Rather than refine abstract plan operators into a detailed end-to-end plan, however, these systems interleave refinement with execution. By postponing refinement until absolutely necessary, such systems leave themselves flexibility to choose refinements that best match current circumstances. However, this means that refinement decisions at abstract levels are made and acted upon before all of the detailed refinements need be made. If such refinements at abstract levels introduce unresolvable conflicts at detailed levels, the system ultimately gets stuck part way through a plan that cannot be completed. While backtracking is possible for HTN planning (since no actions are taken until plans are completely formed), it might not be possible when some (irreversible) plan steps have already been taken. It is therefore critical that the specifications of abstract plan operators be rich enough to summarize all of the relevant refinements to anticipate and avoid such conflicts. In this paper, we formally characterize methods for deriving and exploiting such rich summaries to support interleaved local planning, coordination (plan merging), and execution.

Simple Example

This example illustrates the use of summary information, explains some terminology, and further motivates the formalism of a theory for concurrent hierarchical plans (CHiPs) and summary information.

Suppose that two agents wish to go through a doorway to another location, (*row*, *column*), as shown in Figure 1. Agent A has a hierarchical plan, p, to move from (0,0) to (0,4), and B also has a plan, q, to move from (2,0) to (2,4), but they need to coordinate their plans to avoid collision. Agent A could have preprocessed plan p to derive its summary information. The set of *summary preconditions* of p includes all its preconditions and those of its subplans that must be met external to p in order for p to execute successfully: $\{At(A,0,0)$, $\neg At(B,0,1)$, $\neg At(B,1,0)$, ..., $\neg At(B,0,4)\}$. The proposition $At(A,0,0)$ is a *must* condition because no matter how p is executed, the condition must hold. $\neg At(B,1,0)$ is *may* because it may be required depending on the path A takes. Likewise, the *summary postconditions* of p are its effects and those of its subplans that are seen externally: $\{At(A,2,0)$, $\neg At(A,0,0)$, $\neg At(A,1,0)$, $\ldots\}$. The *summary inconditions* are any conditions that must hold within the interval of time that the plan is executing and can be *must* or *may* and *always* or *sometimes*. An *always* condition is required to hold throughout the duration of any execution of the plan. For example, a *must*, *always* incondition of p could be $PowerOn(A)$– the power must always be on. $At(A,1,0)$ is a *may*, *sometimes* incondition of p because A *may* choose that path and would only be there at *some time*. These conditions and descriptors, such as *must* and *always*, provide the necessary information to reason about what conditions must or may be achieved or clobbered when ordering a set of plan executions.

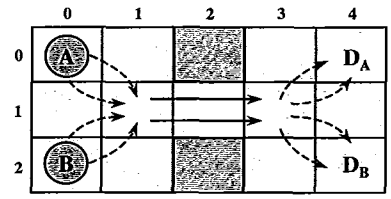

Figure 1: Agents A and B go through a doorway.

Now suppose A sends B p_{sum}, the summary information for p. Agent B can now reason about the interactions of their plans based on their combined summary information. For instance, based only on the summary information, B can determine that if p is restricted to execute before q, then the plans *can* be executed (refined) in *any way*, or $CanAnyWay(\text{b},p_{sum},q_{sum})$.[1] So, B could tell A to go ahead and start execution and to send back a message when p is finished executing. However, B may instead wish to overlap their plan executions for better efficiency. Although $CanAnyWay(\text{o},p_{sum},q_{sum})$ is not true, B could use the summary conditions to determine that there *might* be *some way* to overlap them, or $MightSomeWay(\text{o},p_{sum},q_{sum})$. Then, B could ask A for the summary information of each of p's subplans, reason about the interactions of lower level actions in the same way, and find a way to synchronize the subplans for a more fine-grained solution.

Consider another case where A and B plan to move to the spot directly between them, (1,0), and can choose from different routes. $MightSomeWay(\text{b}, p_{sum}, q_{sum})$ would be false since the postconditions of p must always clobber the preconditions of q. If we wanted to describe a rule for determining whether two actions can or might overlap, it is not obvious how this should be done. The difficulty of composing such rules stems from an imprecise specification of concurrent plan execution and the large space of potential plans that have the same summary information. If the $MightSomeWay(\text{o}, p_{sum}, q_{sum})$ rule is not specified in a complete way, the agent may not determine that the *overlaps* relation cannot hold until it has exhaustively checked all synchronizations of p and q's primitive subplans. As the number of subplans grows, this becomes an intractable procedure (Vilain & Kautz 1986). Even worse would be if, for the sake of trying to be complete, a rule is specified in an unsound way leading to a synchronization choice that causes failure. We give an example of this in (Clement & Durfee 1999b), where we also implement a hierarchical plan coordi-

[1] We will often abbreviate Allen's thirteen temporal relations (Allen 1983). Here, "b" is for the *before* relation. "o" is for *overlaps*.

nation algorithm that uses summary information in the manner described above. Our evaluations show that coordinating at different levels of abstraction for different cost scenarios results in better performance. Thus, formalizing concurrent hierarchical plans, their execution, and the derivation of summary conditions is necessary to avoid costly, irreversible decisions made during planning, plan execution, and coordination.

Overview

In the next section we describe the semantics of hierarchical plans and their concurrent execution to ground our theory. The simple theory of action consistently describes all temporal interactions among primitive or hierarchical plans. We basically add a set of *inconditions* to popular STRIPS-style plan representations to reason about concurrent plan execution. In addition, we formalize traditional planning concepts, such as *clobbers* and *achieves*, and reintroduce *external conditions* (Tsuneto, Hendler, & Nau 1998) for reasoning about CHiPs. We then describe the semantics of plan summary information and a correct method for deriving it efficiently. This, in turn, is used to describe the construction of sound and complete rules for determining how plans can definitely or might possibly be temporally related. The result is a theory for proving correct coordination and planning mechanisms.

A Model of Hierarchical Plans and their Concurrent Execution

The original purpose of developing the following theory was to provide, as simply as possible, a consistent model of execution to generally reason about the concurrent interaction of hierarchical plans. However, we also wanted the model to share important aspects of plans used by PRSs, HTNs, Allen's temporal plans, and many STRIPS-style plan representations. As such, this theory of action tries to distill appropriate aspects of other theories, including (Allen & Koomen 1983), (Georgeff 1984), and (Fagin *et al.* 1995).

CHiPs

A concurrent hierarchical plan p is a tuple $\langle pre, in, post, type, subplans, order\rangle$. $pre(p)$, $in(p)$, and $post(p)$ are sets of literals (v or $\neg v$ for some propositional variable v) representing the preconditions, inconditions, and postconditions defined for plan p.[2] The *type* of plan p, $type(p)$, has a value of either *primitive*, *and*, or *or*. An *and* plan is a non-primitive plan that is accomplished by carrying out all of its subplans. An *or* plan is a non-primitive plan that is accomplished by carrying out one of its subplans. So, *subplans* is a set of plans, and a *primitive* plan's *subplans* is the empty set. $order(p)$ is only defined for an *and* plan p and is a set of temporal relations (Allen 1983) over pairs of subplans that together are consistent; for example, $before(p_i, p_j)$ and $before(p_j, p_i)$ could not both be in *order*. Plans left unordered with respect to each other are interpreted to potentially execute in concurrently. For the example in Figure 1, A's highest level plan p is the tuple $\langle\{\}, \{\}, \{\}, and, \{m_1, m_2, m_3\}, \{before(m_1, m_2), before(m_2, m_3)\}\rangle$. Here, m_1, m_2, and m_3 correspond to p's subplans for moving to (1,1), (1,3), and (0,4) respectively. There are no conditions defined because p can rely on the conditions defined for the primitive plans for moving between grid locations. The primitive plan for moving agent A from (1,3) to (0,3) is the tuple $\langle\{At(A,1,3)\}, \{At(A,1,3), \neg At(B,1,3), \neg At(B,0,3)\}, \{At(A,0,3), \neg At(A,1,3), \neg At(B,0,3), \neg At(B,1,3)\}, primitive, \{\}, \{\}\rangle$.

We also require postconditions to specify whether the inconditions change or not. This helps simplify the notion of inconditions as conditions that hold only *during* plan execution whether because they are *caused* by the action or because they are *necessary conditions* for successful execution. If a plan's postconditions did not specify the truth values of the inconditions' variables at the end of execution, then it is not intuitive how those values should be determined in the presence of concurrently executing plans. By requiring postconditions to specify such values, we resolve all ambiguity and simplify state transitions (described in the section below on Histories and Runs).

The decomposition of a CHiP is in the same style as that of an HTN as described by Erol *et al.* (Erol, Hendler, & Nau 1994). An *and* plan is a task network, and an *or* plan is an extra construct representing a set of all tasks that accomplish the same goal or compound task. Tasks in a network are subplans of the plan corresponding to the network. High-level effects (Erol, Hendler, & Nau 1994) are simply the postconditions of a non-primitive CHiP. CHiPs can also represent a variety of interesting procedures executable by PRSs.

Executions

We recursively describe an execution of a plan as an instance of a decomposition and ordering of its subplans' executions. This helps us reason about the outcomes of different ways to execute a group of plans, describe state transitions, and formalize other terms.

The possible executions of a plan p is the set $\mathcal{E}(p)$. An *execution* of p, $e \in \mathcal{E}(p)$, is a triple $\langle d, t_s, t_f\rangle$. $t_s(e)$ and $t_f(e)$ are positive, non-zero real numbers representing the start and finish times of execution e, and $t_s < t_f$. $d(e)$ is a set of subplan executions representing the decomposition of plan p under this execution e. Specifically, if p is an *and* plan, then it contains one execution from each of the subplans; if it is an *or* plan, then it contains only one execution of one of the subplans; and it is empty if it is *primitive*. In addition, for all subplan executions, $e' \in d$, $t_s(e')$ and $t_f(e')$ must be consistent with the relations specified

[2] Functions such as $pre(p)$ are used for referential convenience throughout this paper. Here, *pre* and $pre(p)$ are the same, and $pre(p)$ is read as "the preconditions of p."

in *order*(p). Also, the first subplan(s) to start must start at the same time as p, $t_s(e') = t_s(e)$; and the last subplan(s) to finish must finish at the same time as the p, $t_f(e') = t_f(e)$. An execution for agent A's top-level plan p (described previously in the section on CHiPs) would be some $e \in \mathcal{E}(p)$. e might be $\langle\{e_1, e_2, e_3\}, 4.0, 10.0\rangle$ where $e_1 \in \mathcal{E}(m_1)$, $e_2 \in \mathcal{E}(m_2)$, $e_3 \in \mathcal{E}(m_3)$, and e begins at time 4.0 and ends at time 10.0. e_1 also starts at 4.0, and e_3 ends at 10.0.

The *subexecutions* of an execution e, sometimes referred to as *subex*(e), is defined recursively as the set of subplan executions in e's decomposition unioned with their subexecutions. For agent A, *subex*(e) = $\{e_1, e_2, e_3\} \cup$ *subex*(e_1) $\cup$ *subex*(e_2) $\cup$ *subex*(e_3). For convenience, we say that a condition of a plan with an execution in the set containing e and e's subexecutions is a *condition of* e. So, if A executes its top-level plan, since $\neg At(B, 1, 2)$ is an incondition of the primitive for A to move from (1,1) to (1,2), it is also an incondition of the primitive's execution, e_2, and e.

Histories and Runs

We describe hypothetical possible worlds, called histories, so that we can determine what happens in all worlds, some, or none. We then can describe how the state transforms according to a particular history. A state of the world, s, is a truth assignment to a set of propositions, each representing an aspect of the environment. We treat a state as the set of true propositional variables.

A *history*, h, is a tuple $\langle E, s_I\rangle$. E is a set of plan executions including those of all plans and subplans executed by all agents, and s_I is the initial state of the world before any plan is begun. So, a history h is a hypothetical world that begins with s_I as the initial state and where only executions in $E(h)$ occur.

A *run*, r, is a function mapping time to states. It gives a complete description of how the state of the world evolves over time. We take time to range over the positive real numbers. $r(t)$ denotes the state of the world at time t in run r. So, a condition is *met* at time t if the condition is a non-negated propositional variable v, and $v \in r(t)$ or if the condition is a negated propositional variable $\neg v$, and $v \notin r(t)$.

For each history h there is exactly one run, $r(h)$[3], that specifies the state transitions *caused* by the plan executions in $E(h)$. The interpretation of a history by its run is defined as follows. The world is in the initial state at time zero: $r(h)(0) = s_I(h)$. In the smallest interval after any point where one or more executions start and before any other start or end of an execution, the state is updated by adding all non-negated inconditions of the plans and then removing all negated inconditions. Similarly, at the point where one or more executions finish, the state is updated by adding all non-negated postconditions of the plans and then removing all negated postconditions. Lastly, if no execution of a plan begins or ends between two points in time, then the state must be the same at those points. First order logic sentences for these axioms are specified in a larger report (Clement & Durfee 1999a).

Now we can define what it means for a plan to execute successfully. An execution $e = \langle d, t_s, t_f\rangle$ *succeeds in* h if and only if the plan's preconditions are met at t_s; the inconditions are met throughout the interval (t_s, t_f); the postconditions are met at t_f; and all executions in e's decomposition are in $E(h)$ and succeed. Otherwise, e *fails*. So, in a history h where agent A successfully executes a plan (as described previously in the section on Executions) to traverse the room, $E(h)$ = $\{e\} \cup$ *subex*(e), and all conditions of all plans with executions in $E(h)$ are met at the appropriate times. Given the example primitive conditions in the section on CHiPs and the axioms just described for state transitions, if agent B happened to start moving into A's target location, (0,4), at the same time as A, then either A's primitive plan execution e_A finishes before B's and the $\neg At(A, 0, 4)$ incondition of B's primitive execution e_B is not met (clobbered) at $t_f(e_A)$; e_B finishes before e_A and similarly clobbers e_A's incondition; or they both finish simultaneously clobbering each others' $At(A/B, 0, 4)$ postconditions. If e_A fails, then e_3 and the top-level execution e must also fail.

Asserting, Clobbering, and Achieving

In conventional planning, we often speak of *clobbering* and *achieving* preconditions of plans (Weld 1994). In CHiPs, these notions are slightly different since inconditions can clobber and be clobbered, as seen in the previous section. Formalizing these concepts helps prove properties of summary conditions. However, it will be convenient to define first what it means to *assert* a condition.

An execution e of plan p is said to *assert* a condition ℓ at time t in a history h if and only if ℓ is an incondition of p, t is in the smallest interval beginning after $t_s(e)$ and ending before a following start or finish time of any execution in $E(h)$, and ℓ is satisfied by $r(h)(t)$; or ℓ is a postcondition of p, $t = t_f(e)$, and ℓ is satisfied by $r(t)$. So, asserting a condition only *causes* it to hold if the condition was not previously met. Otherwise, the condition was already satisfied and the action requiring it did not really *cause* it.

A *precondition* ℓ of plan p_1 is [*clobbered, achieved*][4] in e_1 (an execution of p_1) by e_2 (an execution of plan p_2) at time t if and only if e_2 asserts $[\ell', \ell]$ at t; $\ell \Leftrightarrow \neg\ell'$; and e_2 is the last execution to assert ℓ or ℓ' before or at $t_s(e_1)$. An [*incondition, postcondition*] ℓ of plan p_1 is *clobbered* in e_1 by e_2 at time t if and only if e_2 asserts ℓ' at t; $\ell \Leftrightarrow \neg\ell'$; and $[t_s(e_1) < t < t_f(e_1)$,

[3] For convenience, we now treat r as a function mapping histories to runs, so $r(h)(t)$ is a mapping of a history and a time to a state.

[4] We use braces [ ] as a shorthand when defining similar terms and procedures. For example, saying "[a, b] implies [c, d]" means a implies c, and b implies d.

$t = t_f(e_1)$]. Achieving inconditions and postconditions does not make sense for this formalism, so it is not defined. In the previous section when e_A finished first and asserted $At(A,0,4)$, it clobbered the incondition $\neg At(A,0,4)$ of B's primitive plan in e_B at $t_f(e_A)$.

External Conditions

As recognized in (Tsuneto, Hendler, & Nau 1998), external conditions are important for reasoning about potential refinements of abstract plans. Although the basic idea is the same, we define them a little differently and call them *external preconditions* to differentiate them from other conditions we call *external postconditions*. Intuitively, an external precondition of a group of partially ordered plans is a precondition of one of the plans that is not achieved by another in the group and must be met external to the group. External postconditions, similarly, are those that are not undone by plans in the group and are net effects of the group.

Formally, an *external precondition* ℓ of an *interval* (t_1, t_2) in history h is a precondition of a plan p with some execution $e \in E(h)$ for which $t_1 \leq t_s(e) < t_2$, and ℓ is neither achieved nor clobbered by an execution at a time t where $t_1 \leq t \leq t_s(e)$. An *external precondition* of an *execution* $e = \langle d, t_s, t_f \rangle$ is an external precondition of an interval (t_1, t_2) in some history where $t_1 \leq t_s$; $t_f \leq t_2$; and there are no other plan executions other than the subexecutions of e. An *external precondition* of a *plan* p is an external precondition of any of p's executions. It is called a *must* precondition if it is an external precondition of all executions; otherwise it is called a *may* precondition. $At(A,0,0)$ is an external precondition of agent A's top-level plan p (Figure 1) since no subplan in p's hierarchy achieves $At(A,0,0)$. $At(A,1,1)$ is not an external precondition of p because it is achieved internally by the execution of subplan m_1 (described in the section on CHiPs).

Similarly, an *external postcondition* ℓ of an *interval* (t_1, t_2) in h is a postcondition of a plan p with some execution $e \in E(h)$ for which $t_1 \leq t_f(e) \leq t_2$; ℓ is asserted by e; and ℓ is not clobbered by any execution at a time t where $t_f(e) < t \leq t_2$. External postconditions of executions and plans can be defined in the same way as external preconditions. $At(A,0,4)$ is an external postcondition of agent A's top-level plan p since no subplan in p's hierarchy cancels the effect. $At(A,1,3)$, an external postcondition of m_2 is not an external postcondition of p because it is cancelled internally by the execution of subplan m_3 when it later asserts $\neg At(A,1,3)$.

Plan Summary Information

With the previous formalisms, we can now define summary information and describe a method for computing it for non-primitive plans. The *summary information* for a plan p is p_{sum}. Its syntax is given as a tuple $\langle pre_{sum}, in_{sum}, post_{sum} \rangle$, whose members are sets of *summary conditions*. The *summary* [*pre*, *post*] *conditions* of p, [$pre_{sum}(p)$, $post_{sum}(p)$], contain the external [pre, post] conditions of p. The *summary inconditions* of p, $in_{sum}(p)$, contain all conditions that must hold within some execution of p for it to be successful. A condition c in one of these sets is a tuple $\langle \ell, existence, timing \rangle$. $\ell(c)$ is a literal. The *existence* of c can be *must* or *may*. If $existence(c) = must$, then c is called a *must* condition because ℓ holds for every successful plan execution (ℓ *must* hold). For convenience we usually write $must(c)$. c is a *may* condition ($may(c)$ is *true*) if there is at least one plan execution where $\ell(c)$ must hold. The *timing* of c can take the values *always*, *sometimes*, *first*, *last*. $timing(c)$ is *always* for $c \in in_{sum}$ if $\ell(c)$ is an in-condition that must hold throughout the execution of p (ℓ holds *always*); otherwise, $timing(c) = sometimes$ meaning $\ell(c)$ holds at one point, at least, within an execution of p. The *timing* is *first* for $c \in pre_{sum}$ if $\ell(c)$ holds at the beginning of an execution of p; otherwise, $timing = sometimes$. Similarly, *timing* is *last* for $c \in post_{sum}$ if $\ell(c)$ holds at the end of an execution of p; otherwise, it is *sometimes*. Although *existence* and *timing* syntactically only take one value, semantically $must(c) \Rightarrow may(c)$, and $always(c) \Rightarrow sometimes(c)$. See the Introduction for an example of summary conditions derived for an abstract plan.

Deriving Summary Conditions

The method for deriving the summary conditions of a plan p is recursive. First, summary information must be derived for each of p's subplans, and then the following procedure derives p's summary conditions from those of its subplans and its own sets of conditions. This procedure only apply to plans whose expansion is finite and which have the downward solution property. This is the property where every *or* plan in the hierarchy can be refined successfully through one or more of its subplans.

Summary conditions for primitives and non-primitives

- First, for each literal ℓ in $pre(p)$, $in(p)$, and $post(p)$, add a condition c with literal ℓ to the respective set of summary conditions for plan p. $existence(c)$ is *must*, and $timing(c)$ is *first*, *always*, or *last* if ℓ is a pre-, in-, or postcondition respectively.

Summary [pre, post] conditions for *and* plan

- Add a condition c to the summary [pre, post] conditions of *and* plan p for each summary [pre, post] condition c' of p's subplans that is not [*must-achieved*, *must-undone*][5] by another of p's subplans, setting $\ell(c) = \ell(c')$.[6]
- Set $existence(c) = must$ if $\ell(c)$ is a [pre, post] condition of p or is the literal of a *must* summary [pre, post]

[5] See (Clement & Durfee 1999a) and the proof ending this section about how to determine *must-achieved*, *may-achieved*, *must-undone*, and *may-undone*.

[6] To resolve ambiguity with set membership, we say that any two summary conditions, c and c' are equal if $\ell(c) = \ell(c')$ and they belong to the same set of summary conditions for some plan.

condition in a subplan of p that is not [*may-achieved*, *may-undone*] by any other subplans. Otherwise, set $existence(c) = may$.

- Set $timing(c) = [first, last]$ if $\ell(c)$ is a [pre, post] condition of p or the literal of a [*first*, *last*] summary [pre, post] condition of a [least, greatest] temporally ordered subplan (i.e. no others are constrained by $order(p)$ to [begin before, end after] it). Otherwise, set $timing(c) = sometimes$.

Summary [pre, post] conditions for *or* plan

- Add a condition c to the summary [pre, post] conditions of *or* plan p for each summary [pre, post] condition c' in p's subplans, setting $\ell(c) = \ell(c')$.
- Set $existence(c) = must$ if $\ell(c)$ is a [pre, post] condition of p or a *must* summary [pre, post] condition of all of p's subplans. Otherwise, set $existence(c) = may$.
- Set $timing(c) = [first, last]$ if $\ell(c)$ is a [pre, post] condition of p or the literal of a [*first*, *last*] summary [pre, post] condition in a subplan. Otherwise, set $timing(c) = sometimes$.

Summary inconditions for *and* plan

- Add a condition c to the summary inconditions of *and* plan p for each c' in C defined as the set of summary inconditions of p's subplans unioned with the set of summary preconditions of the subplans that are not *first* in a least temporally ordered subplan and with the set of summary postconditions of the subplans that are not *last* in a greatest temporally ordered subplan, and set $\ell(c) = \ell(c')$.
- Set $existence(c) = must$ if $\ell(c)$ is an incondition of p or a literal of a *must* summary condition $c' \in C$, as defined above. Otherwise, set $existence(c) = may$.
- Set $timing(c) = always$ if $\ell(c)$ is an incondition of p or a literal in an *always* summary incondition in every subplan of p. Otherwise, set $timing(c) = sometimes$.

Summary inconditions for *or* plan

- Add a condition c to the summary inconditions of *or* plan p for for each summary incondition c' in p's subplans, setting $\ell(c) = \ell(c')$.
- Set $existence(c) = must$ if $\ell(c)$ is an incondition of p or a *must* summary incondition of all of p's subplans. Otherwise, set $existence(c) = may$.
- Set $timing(c) = always$ if $\ell(c)$ is an incondition of p or an *always* summary incondition of all of p's subplans. Otherwise, set $timing(c) = sometimes$.

Consider deriving the summary conditions of m_2 from its two primitive subplans (as introduced in the section on CHiPs). Suppose p_1 is the primitive subplan for moving agent A from (1,1) to (1,2), and p_2 moves A from (1,2) to (1,3). First, the summary conditions of the primitives must be derived. These are simply the conditions already defined for the primitives according to the first step of the procedure. m_2 has no conditions defined for itself, so all will come from p_1 and p_2. Since m_2 is an *and* plan, its only summary precondition is $At(A,1,1)$ from p_1 because p_1 *must achieve* p_2's only precondition $At(A,1,2)$. $At(A,1,1)$ is a *must* summary condition because it is a *must* summary precondition in p_1, and no other subplan (p_2) *may achieve* $At(A,1,1)$. $At(A,1,1)$ is also *first* because it is a *first* summary precondition of p_1, and p_1 precedes p_2.

The procedure above ensures that external conditions are captured by summary conditions and *must*, *always*, *first*, and *last* have their intended meanings. The actual proof of these properties is all-inclusive since the truth of each property depends on those of others. However, we give a proof of one (assuming the others) to illustrate how we verify these properties using the language developed in this paper. The full proof is given in an extended report (Clement & Durfee 1999a). These results ease the proofs of soundness and completeness for inference rules determining how CHiPs can definitely or potentially interact so that good planning and coordination decisions can be made at various levels within and among plan hierarchies.

Theorem The set of external preconditions for a plan is equivalent to the set of all literals in the plan's summary preconditions.

Proof by induction over the maximum subplan depth. The base case is a primitive plan p (subplan depth zero). The summary preconditions include a condition for every precondition of p, which must be an external precondition of p, so this case is satisfied. Assume that the theorem is true for all plans of maximum depth $\leq k$. Any plan p of maximum depth $k+1$ must have subplans with maximum depths $\leq k$. It is not difficult to show that the external preconditions of p must be the preconditions of p and those external preconditions (call them pre_x) of p's subplans that are not *must-achieved* (achieved in all executions) by another subplan of p.

By the inductive hypothesis, the external conditions of the subplans are captured in their summary conditions, and the *existence* and *timing* information together with the *order* of p's subplans can be used to determine whether some external precondition of a subplan is *must-achieved*. Table 1 shows this by describing for all cases the constraints on $order(p)$ where $\ell(c')$ of p' is *must-achieved* by p''. If we did not assume the downward solution property, then we would additionally need to make sure that no other plan could clobber $\ell(c')$ after p'' asserts the condition. Hence, the external preconditions in pre_x that are not *must-achieved* are exactly those determined in the rule for determining the summary preconditions of an *and* plan. Therefore, the external conditions of p are exactly those described as the summary preconditions of p in the procedure described above. □

Complexity

The procedure for deriving summary conditions works by basically propagating the conditions from the primitives up the hierarchy to the most abstract plans. Because the conditions of any non-primitive plan depend only on those of its immediate subplans, deriving summary conditions can be done quickly. Given that an agent has an instantiated plan hierarchy with n non-primitive plans, each of which have b subplans and c conditions in each of their summary pre-, in-, and

$c'' \in post_{sum}(p'')$		$c' \in pre_{sum}(p')$		p'' *must-achieve* c' $\forall e' \in \mathcal{E}(p')$, $e'' \in \mathcal{E}(p'')$
must	*last*	*must*	*first*	
F	?	?	?	*false*
T	?	?	?	$t_f(e'') \leq t_s(e')$
$c'' \in in_{sum}(p'')$				
must	*always*	*must*	*first*	
F	?	?	?	*false*
?	F	?	?	*false*
T	T	?	F	$t_s(e'') \leq t_s(e') \wedge$ $t_f(e') \leq t_f(e'')$
		?	T	$t_s(e'') < t_s(e') < t_f(e'')$

Table 1: Ordering constraints necessary for subplan p'' to *must-achieve* $c' \in pre_{sum}$ of subplan p'. "?" means that the constraints hold for both truth values. *false* means that there are no ordering constraints guaranteeing that p' is achieved by p''.

postconditions, deriving the summary conditions of the non-primitive plans can be bounded by $O(nb^2c^2)$ operations. This comes from the worst case in which all plans are *and* plans requiring the procedure to test each of c conditions in each of b subplans to see if they are achieved/clobbered by any those same conditions in any of the same subplans for each of the n *and* plans. However, $n = O(b^d)$ for hierarchies of depth d, so the complexity of the procedure for deriving summary conditions is more simply $O(n(log^2 n)c^2)$.

Soundness and Completeness of Determining Temporal Relations

With the properties of summary information proven, we can safely reason about the interactions of plans without information about their subplans. Based on the *existence* and *timing* information carried by summary conditions, we can determine what relations can or might hold between CHiPs without searching their subplans. In many coordination tasks, if it could be determined that certain temporal relations can hold among plans no matter how they are decomposed ($CanAnyWay$) or that certain relations cannot hold for any decomposition ($\neg MightSomeWay$), then coordination decisions can be made at abstract levels without entering a potentially costly search for valid plan merges. Here we prove the soundness and completeness of rules determining $CanAnyWay$ and $MightSomeWay$ relations based on summary information. The use of these rules is illustrated in the Introduction and explored further in (Clement & Durfee 1999b). For convenience, we will abbreviate *Can* to C, *Any* to A, *Way* to W, and so on.

Informally, [$CAW(rel, p_{sum}, q_{sum})$, $MSW(rel, p_{sum}, q_{sum})$] says that the temporal relation *rel* [*can*, *might*] hold for any CHiPs p and q whose summary information is p_{sum} and q_{sum} for [*any way*, *some way*] that p and q may be executed. We formally describe these relations by explaining what soundness and completeness mean for rules to determine them.

Let us define $AW(P, s_I)$ to mean that in any history h with initial conditions s_I and where $E(h)$ includes an execution of each plan in set P as well as its subexecutions, all executions succeed. Also, let [$AW(rel, p, q)$, $SW(rel, p, q)$] be true iff for [any, some] history h where $AW(\{p\}, s_I(h))$ and $AW(\{q\}, s_I(h))$ are true, and $E(h)$ includes executions of p and q and their subexecutions satisfying relation *rel*, all executions succeed. So, a rule for determining [$CAW(rel, p_{sum}, q_{sum})$, $MSW(rel, p_{sum}, q_{sum})$] is sound if whenever the rule returns *true*, [$AW(rel, p, q)$, $SW(rel, p, q)$] is true for [all pairs, some pair] of plans whose summary information is p_{sum} and q_{sum}. The rule is complete if whenever [$AW(rel, p, q)$, $SW(rel, p, q)$] is true for [all, some pair of] plans p and q with summary conditions p_{sum} and q_{sum}, the rule also returns *true*.

Now we state the rules for determining *overlaps*. $CAW(\mathrm{o}, p_{sum}, q_{sum})$ returns *true* iff there is no c_p and c_q such that $\ell(c_p) \Leftrightarrow \neg\ell(c_q)$ and either $c_p \in in_{sum}(p)$ and $c_q \in pre_{sum}(q) \cup in_{sum}(q)$ or $c_p \in post_{sum}(p)$ and $c_q \in in_{sum}(q)$. $MSW(\mathrm{o}, p_{sum}, q_{sum})$ returns *true* iff there is no c_p and c_q such that $\ell(c_p) \Leftrightarrow \neg\ell(c_q)$; $must(c_p)$ and $must(c_q)$ is true; the *timing* of c_p and c_q is either *first*, *always*, or *last*; and either $c_p \in in_{sum}(p)$ and $c_q \in pre_{sum}(q) \cup in_{sum}(q)$ or $c_p \in post_{sum}(p)$ and $c_q \in in_{sum}(q)$. Note that for n conditions in each of the pre-, in-, and postconditions of p and q, in the worst case each condition in one plan must be compared with each in another plan, so the complexity of determining CAW and MSW is $O(n^2)$. Now we will show that the above rules are both sound and complete. The proofs of rules for other temporal relations are no more difficult than this one and can be done constructively in the same style.

Theorem $CAW(\mathrm{o}, p_{sum}, q_{sum})$ and $MSW(\mathrm{o}, p_{sum}, q_{sum})$ are sound and complete.

Proof Since we assume $AW(\{p\}, s_I)$ and $AW(\{q\}, s_I)$, the only way executions of p or q could fail is if a condition in one is clobbered by the other. However, a condition is clobbered only by the assertion of the negation of its literal. Thus, the presence of conditions involving the propositional variable v cannot clobber or achieve conditions involving the propositional variable v' if $v \neq v'$. In addition, all cases of execution failure can be broken down into inconditions of some execution e clobbering conditions that must be met in the interval $(t_s(e), t_f(e))$ and postconditions of some execution e clobbering conditions at or after $t_f(e)$. With this established it is not difficult to show that the only pairs of interacting sets where clobbering occurs for the *overlaps* relation include $(in_{sum}(p), pre_{sum}(q))$, $(in_{sum}(p), in_{sum}(q))$, and $(post_{sum}(p), in_{sum}(q))$. And, because the clobbering of a single literal causes execution failure, we can describe all histories in terms of the presence and absence of summary conditions based on a single propositional variable.

This is done for the *overlaps* relation in Table 2. Here, we give all instances of these interacting sets and claim that the listed truth values for CAW and MSW

	A		B		$A = in_{sum}(p)$ $B = pre_{sum}(q)$		$A = in_{sum}(p)$ $B = in_{sum}(q)$		$A = post_{sum}(p)$ $B = in_{sum}(q)$	
					CAW	MSW	CAW	MSW	CAW	MSW
1	-		?		T	T	T	T	T	T
	?		-		T	T	T	T	T	T
2	ℓ		ℓ		T	T	T	T	T	T
3	ℓ		$\neg\ell$							
	m	a/l	m	f/a						
a	T	T	T	T	F	F	F	F	F	F
b	F	?	?	?	F	T	F	T	F	T
c	?	F	?	?	F	T	F	T	F	T
d	?	?	F	?	F	T	F	T	F	T
e	?	?	?	F	F	T	F	T	F	T
4	$\ell, \neg\ell$		ℓ		F	T	F	T	F	T
5	ℓ		$\ell, \neg\ell$		F	T	F	T	F	T
6	$\ell, \neg\ell$		$\ell, \neg\ell$		F	T	F	T	F	T

Table 2: Truth values of $CanAnyWay(o, p, q)$ and $MightSomeWay(o, p, q)$ for all interactions of conditions. ℓ and $\neg\ell$ are literals of summary conditions in sets A and B. The condition is *must* if m = T, *first* for f = T, *last* for l = T, and *always* for a = T.

are correct for the space of plan pairs (p, q) that elicit the instances represented by each row. The literal ℓ in the first two columns represents any literal that appears in any condition of the set in whose column it appears. For example, row 4 for the interaction of $(in_{sum}(p), pre_{sum}(q))$ is interpreted as the case where there is a literal that appears in both sets, and its negation also appears in just $in_{sum}(p)$. [F, T] in the $[CAW, MSW]$ column means that [*not all* histories, there is at least one history] with only the executions and subexecutions of [any, some] pair of plans whose summary conditions have literals appearing this way are such that all executions succeed. Thus, $[CAW(o, p_{sum}, q_{sum}), MSW(o, p_{sum}, q_{sum})]$ is true iff there are no summary conditions matching cases in the table where there is an F entry in a $[CAW, MSW]$ column.

The validity of most entries in Table 2 is simple to verify. In row 1 either there is no condition to be clobbered or no condition to clobber it, so all executions must succeed. In row 3 unless the conditions are all *must* and either *first*, *always*, or *last* (row 3a), there is always a history containing the executions of some plans where the two conflicting conditions are not required to be met at the same time, and MSW is true. Because rules defined for $CAW(o, p_{sum}, q_{sum})$ and $MSW(o, p_{sum}, q_{sum})$ return *true* whenever the table has a T entry, they are complete. And because the table has a T entry for every case in which the rules return *true*, the rules are sound. □

Conclusions and Future Work

Coordination and planning for hierarchical plans often involve the refinement of abstract plans into more detail. However, the cost of backtracking or making irreversible commitments makes it critical that the specification of abstract plan operators be rich enough to anticipate and avoid costly planning/execution decisions. We have addressed this directly by offering a formalism for describing concurrent hierarchical plan execution and methods for deriving summary conditions and determining legal plan interactions in a sound and complete fashion. This provides a foundation upon which provably correct coordination and planning mechanisms can be built. In prior work that motivates and validates this formalism, we describe a general algorithm for merging hierarchical plans using summary information, a specific implementation, and a preliminary evaluation of the approach (Clement & Durfee 1999b). Future work includes relaxing assumptions, such as the downward solution property, investigating other types of plan summary information, and constructing sound and complete algorithms for concurrent hierarchical planning and for interleaving planning, plan coordination, and plan execution.

References

Allen, J. F., and Koomen, J. A. 1983. Planning using a temporal world model. In *Proc. IJCAI*, 741–747.

Allen, J. F. 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM* 26(11):832–843.

Clement, B., and Durfee, E. 1999a. Theory for coordinating concurrent hierarchical planning agents. http://www.eecs.umich.edu/~bradc/papers/aaai99.

Clement, B., and Durfee, E. 1999b. Top-down search for coordinating the hierarchical plans of multiple agents. In *Proc. Intl. Conf. Autonomous Agents.*

Corkill, D. 1979. Heirarchical planning in a distributed environment. In *Proc. IJCAI*, 168–175.

Ephrati, E., and Rosenschein, J. 1994. Divide and conquer in multi-agent planning. In *Proc. AAAI*, 375–380.

Erol, K.; Hendler, J.; and Nau, D. 1994. Semantics for hierarchical task-network planning. Technical Report CS-TR-3239, University of Maryland.

Fagin, R.; Halpern, J.; Moses, Y.; and Vardi, M. 1995. *Reasoning about knowledge.* MIT Press.

Firby, J. 1989. *Adaptive Execution in Complex Dynamic Domains.* Ph.D. Dissertation, Yale University.

Georgeff, M. P., and Lansky, A. 1986. Procedural knowledge. *Proc. IEEE* 74(10):1383–1398.

Georgeff, M. P. 1983. Communication and interaction in multiagent planning. In *Proc. AAAI*, 125–129.

Georgeff, M. P. 1984. A theory of action for multiagent planning. In *Proc. AAAI*, 121–125.

Lansky, A. 1990. Localized search for controlling automated reasoning. In *Proc. DARPA Workshop on Innov. Approaches to Planning, Scheduling and Control*, 115–125.

Sacerdoti, E. D. 1977. *A structure for plans and behavior.* Elsevier-North Holland.

Tate, A. 1977. Generating project networks. In *Proc. IJCAI*, 888–893.

Tsuneto, R.; Hendler, J.; and Nau, D. 1998. Analyzing external conditions to improve the efficiency of htn planning. In *Proc. AAAI*, 913–920.

Vilain, and Kautz, H. 1986. Constraint propagation algorithms for temporal reasoning. In *Proc. AAAI*, 377–382.

Weld, D. 1994. An introduction to least commitment planning. *AI Magazine* 15(4):27–61.