SUPPORTING COLLABORATIVE PLANNING:  
THE PLAN INTEGRATION PROBLEM

by

David Ari Rosenblitt

Submitted to the

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

in partial fulfillment of the requirements

for the Degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1991

(c) Massachusetts Institute of Technology

Signature of

Author

Department of Electrical Engineering and Computer Science

Certified

by

Thesis Supervisor

Accepted

by

Arthur C. Smith

Chairman, Departmental Committee on Graduate Students
SUPPORTING COLLABORATIVE PLANNING:
THE PLAN INTEGRATION PROBLEM

by

David Ari Rosenblitt

Submitted to the Department of Electrical Engineering and Computer Science on January 17, 1991 in partial fulfillment of the requirements for the Degree of Doctor of Philosophy

Abstract

When different members of a work group develop their own individual plans, or sets of tasks to achieve desired goals, there may be conflicting and synergistic interactions among these plans. Conflicts may arise when one task negates the effect of another task, or two tasks compete for the same resource. Synergies may arise if the desired effects of some tasks are also accomplished by other tasks, allowing some of the tasks to be deleted. In many organizations, plans are often poorly integrated: conflict detection and resolution are performed late in the planning cycle, resulting in costly revisions and delays, and potential synergies are overlooked and unexploited, resulting in wasted resources.

This dissertation details a framework for solving the plan integration problem, and shows how this capability can support an important aspect of cooperative work: collaborative planning. The utility of plan integration in supporting collaborative planning is illustrated in a construction planning scenario, based on an actual project. The planning framework is domain-independent and provably correct. Unlike previous work in AI planning theory, it includes a general mechanism for reasoning about resources. The planning algorithms are implemented in Synapse, a prototype collaborative planning tool.

Thesis Supervisor: Dr. Thomas W. Malone

Title: Patrick J. McGovern Professor of Information Systems
Acknowledgments

First and foremost, I wish to thank Tom Malone, my research advisor of many years, for his consistently penetrating insights and criticisms, willingness to take time out of a hectic schedule for yet another thesis discussion, and most of all, encouragement and support (including, but by no means limited to financial support) during this long journey toward a doctorate. I also thank David McAllester, who served as an unofficial member of my thesis committee, above and beyond the call of duty, and under whose tutelage I became a struggling novice in the black arts of theoretical rigor and algorithmic correctness proofs. My other readers, Ramesh Patil and Patrick Henry Winston also contributed important insights along the way.

I also wish to thank my friends and colleagues at the Center for Coordination Science, Laboratory for Computer Science, and Artificial Intelligence Laboratory, who contributed significantly to the quality of this research, including Franklyn Turbak, Paul Resnick, Ian Horswill, Kevin Crowston, Jintae Lee, Kum-Yew Lai, Mark Ackerman, and Brian Pentland.

I thank Randy Davis and Karl Ulrich for helping to clarify and focus my thesis ideas, Dave Gifford for his crucial support during the beginning of my graduate school career, and Ron Rivest for consistent good advice in his capacity as academic advisor. A special thanks to Bob Logcher, who helped identify the collaborative construction planning scenario.

I also wish to acknowledge those people outside of MIT who contributed to the sharpening of these research ideas, including Candy Sidner and Mark Fox.
I also wish to express my heartfelt appreciation for the friendship of those past and present members of Lens project, especially Ken Grant, Wendy Mackay, Keh-Chiang Yu, Cheryl Clark, Maya Bernstein, Kum-Yew Lai, and Ramana Rao.

I also thank Jolene Galegher, for sound advice on how to present my research.

I also wish to acknowledge the friendship of various MIT administrative assistants I have known over the years, including Emmeli Adler, Elesse Brown, Mary Spollen, Jessica Balaban, Rebecca Bisbee, and Jean Wolff.

Where would I be without my friends, at MIT, Boston, and elsewhere? I acknowledge the friends I've retained from my undergraduate days, including Martin (who was consistently insightful and humorous in our frequent electronic mail conversations), Rob, Morgan, Steve, Dean and Leslie, John, Doug, Paul, Glenn, and Cheryl. I also thank Bett for adding some spice to the life of an jaded grad student. I'll always remember my friends in the study group, especially Doug, Scott, Leslie, Nancy, Angela, and Mary. And I thank various other special friends, including Dennis, Garret, Peter, Lyn, and Chuck.

A special thanks to my parents, William and Florence Rosenblitt, and sisters Alice and Ruth, who loved and supported me in more ways over these many years than can be recounted, and never lost their faith in me. I am very grateful and indebted to them all.
Table of Contents

Abstract .............................................................................................................................................. 2
Acknowledgments ............................................................................................................................. 3
Table of Contents ............................................................................................................................. 5
List of Figures and Tables .................................................................................................................. 7
1. Introduction .................................................................................................................................. 8
   1.1. Research Contributions .......................................................................................................... 9
   1.2. A Collaborative Planning Scenario .......................................................................................10
   1.3. Dissertation Roadmap ............................................................................................................11
2. Planning With Resources .............................................................................................................13
   2.1. Summary ................................................................................................................................13
   2.2. Representing Plans ..................................................................................................................16
       2.2.1. The STRIPS Task Representation .....................................................................................16
       2.2.2. Extending the STRIPS Representation for Resources ....................................................17
   2.3. A Linear Resource-Manipulating Planner .................................................................................18
       2.3.1. Soundness and Completeness Invariants ...........................................................................20
       2.3.2. The Linear Planning Algorithm ..........................................................................................22
   2.4. A Nonlinear Resource-Manipulating Planner ...........................................................................22
       2.4.1. Nonlinear Plan Representation and Terminology ..............................................................23
       2.4.2. Soundness and Completeness ............................................................................................28
       2.4.3. The Nonlinear Planning Algorithm ....................................................................................31
       2.4.4. Performance .......................................................................................................................33
       2.4.5. Extensions ..........................................................................................................................34
           2.4.5.1. Scalable Operators ......................................................................................................34
               2.4.5.1.1. Extending the Nonlinear Planner for Scalable Operators .....................................35
           2.4.5.2. Reformulating Unsolvable Planning Problems ...........................................................36
               2.4.5.2.1. The Problem Reformulation Algorithm .................................................................36
   2.5. An Example: Planning the Construction of a Warehouse and a School .................................37
3. Plan Integration ..............................................................................................................................45
   3.1. Summary ................................................................................................................................45
   3.2. The Plan Integration Problem ................................................................................................46
   3.3. A Plan Checker .........................................................................................................................48
       3.3.1. Redundant Nodes ...............................................................................................................49
       3.3.2. The Plan Checking Algorithm ...........................................................................................51
       3.3.3. A Single-Plan Checking Algorithm ....................................................................................52
   3.4. Two Approaches to Plan Integration ........................................................................................52
       3.4.1. A Provably Correct Plan Integrator ....................................................................................52
       3.4.2. A Heuristic Plan Integrator ................................................................................................54
   3.5. A Collaborative Planning Example: Checking and Integrating Plans to Construct a Warehouse and a School ...........................................................................................................55
       3.5.1. Checking the Plans .............................................................................................................55
           3.5.1.1. Conflicts ........................................................................................................................57
           3.5.1.2. Redundancies ................................................................................................................57
       3.5.2. Integrating the Plans ...........................................................................................................58
4. The Synapse Implementation ..........................................................................................................61
   4.1. Overview ................................................................................................................................61
   4.2. The User Interface ....................................................................................................................61
       4.2.1. System Functionality ..........................................................................................................62
       4.2.2. User Models of Tasks .........................................................................................................63
           4.2.2.1. Names of Object Types and Fields ..............................................................................63
           4.2.2.2. The Context Mechanism ..............................................................................................64
List of Figures and Tables

Figure 2.1: A Causal Link 25
Figure 2.2: An Open Precondition 25
Figure 2.3: An Unsafe Causal Link 26
Figure 2.4: Demoting the Clobberer 26
Figure 2.5: Promoting the Clobberer 26
Figure 2.6: A Supplier 27
Figure 2.7: Mutual Competitors 27
Figure 2.8: A Linear Interpretation 30
Figure 2.9: Initial State of the Warehouse Plan 38
Figure 2.10: Goal of the Warehouse Plan 39
Figure 2.11: The Warehouse Plan 40
Figure 2.12: Resources Used in the Warehouse Plan 41
Figure 2.13: The Goal of the School Plan 42
Figure 2.14: The Resources Used in the School Plan 43
Figure 2.15: The School Plan 44
Figure 3.1: Two Redundant Nodes 50
Figure 3.2: The Warehouse and School Plans 56
Figure 3.3: Integrating the Warehouse and School Plans 58
Figure 3.4: Resources Used in the Integrated Plan 59
Table 4.1: New Object Lens Types Required by Synapse 64
Table 4.2: Running Times for the Construction Example 69
Figure A.1: The Warehouse Construction Plan 88
1. **Introduction**

When different members of a work group develop their own individual plans, or sets of tasks to achieve desired goals, there may be conflicting and synergistic interactions among these plans. In many organizations, plans are often poorly integrated: conflict detection and resolution are performed late in the planning cycle, resulting in costly revisions and delays, and potential synergies are overlooked and unexploited, resulting in wasted resources. Although existing computer-supported cooperative work tools may be used to support the development and execution of individual plans (Croft, 1988), no system to date is capable of integrating separately developed plans so that conflicts are resolved and synergies are exploited. This dissertation details a framework for solving the *plan integration problem*, a capability that can support an important aspect of cooperative work: *collaborative planning*. Collaborative planning occurs when multiple agents iteratively develop and exchange their plans, as they discover and integrate conflicting and synergistic interactions. The planning framework is domain-independent, provably correct, and includes the capability for reasoning about *resources*. Most previous planners either ignored resources, or were only able to detect a limited type of resource conflict: when a binary reusable resource (i.e. a resource that is either available or unavailable) is utilized by possibly simultaneous steps.

Plan integration is useful in domains where the development of high-quality plans is important. Plan integration may be especially useful in domains characterized by long periods of plan development (e.g., process planning, construction planning), or repeated execution of plans in a relatively static environment (e.g., manufacturing). In contrast, plan integration techniques are not likely to be applicable in domains requiring a real-time response to a dynamically changing world. These domains require a reactive planning approach (Agre and Chapman, 1987, Georgeff and Lansky, 1987), where an intelligent
plan execution module is crucial, and the planner itself may be of secondary importance, because assumptions made by a planner may be invalidated by later changes in the execution environment, and most planning problems are too complex to solve in real-time (Chapman, 1987). However, like Agre and Chapman (Agre and Chapman, 1988), I view plans as a reference guide that may be improvised upon, rather than as a script that is rigidly followed.

1.1. Research Contributions

This dissertation makes the following research contributions to artificial intelligence planning techniques:

1) It identifies and provides a solution for the plan integration problem. Unlike previous multi-agent planning systems (Corkill, 1979, Rosenschein, 1982, Georgeff, 1987, Durfee and Lesser, 1987) and hierarchical planners (e.g., iNOAH (Sacerdoti, 1975), NONLIN (Tate, 1977), SIPE (Wilkins, 1988)), Synapse provides a domain-independent, provably correct mechanism for handling conflicting and synergistic resource interactions.

2) It supports planning with resources. Resources are objects that are manipulated by tasks. Like Wilkins (Wilkins, 1988), I distinguish between consumable and reusable resources. Consumable resources may be consumed by tasks, in which case they are no longer available to other tasks. Reusable resources may be utilized by tasks, in which case they become available to other tasks when the utilizing task is finished. Both types of resources may be produced by tasks. Most previous planners either ignored resources, or were only able to detect a limited type of resource conflict: when a binary reusable resource (i.e. a resource that is either available or unavailable) is utilized by possibly simultaneous
steps. Synapse also pioneers the concept of reformulating unsolvable planning problems, by making incremental changes in initial resource allocations or production goals.

This dissertation also illustrates how a plan integration system can support one aspect of collaborative work: collaborative planning. An implemented prototype, Synapse, has been used to demonstrate collaborative planning in a scenario based on an actual construction project (Barrie and Paulson, 1984).

1.2. A Collaborative Planning Scenario

I illustrate plan integration techniques in a simple collaborative construction planning scenario, based on an actual construction project (Barrie and Paulson, 1984). The scenario is outlined below, and will be fleshed out in detail as various parts of the system are described. In this scenario, two contractors develop plans to achieve their respective goals: the construction of a warehouse and a school. Each plan includes generic construction tasks, as well as tasks that are specific to warehouse or school construction. The following conflicting and synergistic interactions occur in this scenario:

1) A task in the warehouse plan competes with a task in the school plan for a shared resource. For example, both the roofing task in the warehouse plan and the precast walls task in the school plan may require use of the crane at the same time. This situation represents a conflict that may be resolved by imposing a precedence constraint between the two tasks, or by increasing the supply of the shared resource.

2) Although the initial supply of some resource is sufficient to meet the demand of either plan, it is insufficient to meet their combined demand. For example, there is enough
concrete and steel on hand to construct either the warehouse or the school, but not both. This represents a conflict that may be resolved by increasing the initial allocation of the resource.

3) There is a setup task that is present in both plans and redundant. For example, there is a concrete setup task in both the warehouse and school construction plans, that need only be done once. This situation represents a potential synergy that may be exploited by deleting the redundant task from one of the plans.

4) A pair of tasks may be merged to achieve an economy of scale. For example, it may be cheaper or quicker to perform the lay concrete task in the warehouse plan simultaneously with the lay concrete task in the school plan. This situation represents a potential synergy that may be exploited by merging the two tasks into a single task that accomplishes the results of the original tasks.

1.3. Dissertation Roadmap

The reader primarily interested in the collaborative work aspects of Synapse will probably be most interested in Sections 2.5 and 3.5, which illustrate its functionality in an actual scenario, in addition to Chapter 4, which discusses user interface and implementation issues.

The reader primarily interested in this dissertation's contribution to artificial intelligence, will probably be most interested in Chapter 2, which describes the concepts, representations, and algorithms used in a planner that supports planning with resources and
problem reformulation, Chapter 3, which does the same for the plan integrator, and Chapter 5, which compares Synapse to previous planners.

Both audiences will probably be interested in Chapter 6, which concludes with a dissertation summary and a discussion of future research directions.
2. Planning With Resources

2.1. Summary

Section 2.2 introduces some basic planning terminology, and extends the classical STRIPS-based (Fikes, 1971) approach to include operators that consume and produce quantities of resources, which are ubiquitous in many real-world planning domains. The key ideas of Section 2.2 are:

1) a plan consists of a set of steps that achieve some goal when executed in some initial state

2) a linear plan specifies a total order on steps; a nonlinear plan specifies a partial order on steps

3) each step specifies a set of preconditions and effects, which may refer to propositions required or asserted by the step, or resources produced or consumed by the step

Section 2.3 extends McAllester's linear propositional planner (McAllester, 1991) to handle resource-manipulating operators. Two key concepts defined in this section are:

1) Weakest precondition, a minimal precondition for a partial plan to achieve its goal. The weakest precondition is useful in reasoning about the correctness of
plans (e.g., a plan \( f \) is a solution if \( G' \), the weakest precondition for \( f \) to achieve the goal \( G \) is satisfied by the initial state).

2) *Guaranteed resource availability* (GRA), or the amount of a resource that can be guaranteed to be available when some step is performed. The GRA is useful in determining when there may be a potential deficit of some resource when some step requiring that resource is performed.

Section 2.4 extends McAllester's nonlinear propositional planner (McAllester, 1991) to handle resource-manipulating operators. Nonlinear planning uses an equivalence relation on plans to divide the search space (i.e. nonlinear planning searches the space of equivalence classes of plans rather than the space of plans). Since many different plans can all be equivalent, searching the space of equivalence classes (nonlinear plans) can be much more efficient than searching the space of linear plans.

McAllester's nonlinear planner is used as the starting point for Synapse for two reasons:

1) Its sophisticated representation of nonlinear plans.

2) It represents a performance improvement over TWEAK (Chapman, 1987) without sacrificing theoretical rigor. McAllester exploits the observation that certain choices in the search space need never be backtracked.

The two most important data structures defined in Section 2.4 are:

1) *casual links*, which indicate the propositional dependency structure of a nonlinear plan (McAllester, 1991)
2) guaranteed resource availability (GRA) values (which are computed differently than by the linear planner), which are used to detect resource deficits

Section 2.4 also includes two useful extensions of the nonlinear planning with resources framework:

1) Allowing scalable operators, whose consumption and production may vary according to consumption and production constraints. A new plan modification is now available to reduce a deficit: rescaling, or increasing the production of an existing node. This capability is also important in the plan integration framework (described in Chapter 3).

2) Reformulating unsolvable planning problems so that they become solvable. In particular, if all partial plans in PARTIAL-PLANS contain an un producible resource deficit with respect to r such that no step produces r, then no solution exists. However, incremental changes in production goals or initial resource allocations may yield a solvable problem.

Finally, Section 2.5 briefly illustrates the operation of the Synapse nonlinear planner with two example plans from the building construction domain: a plan for building a warehouse, and a plan for building a school. (Section 3.5 will illustrate conflicting and synergistic interactions between these two plans.)
2.2. Representing Plans

Section 2.2.1 describes the classical STRIPS task representation, and Section 2.2.2 extends this representation to handle steps that manipulate resources.

2.2.1. The STRIPS Task Representation

A planning problem may be represented as a triple \(<I, G, \{S\}>\), where \(I\) is the initial state, \(G\) is the goal state, and \(\{S\}\) is a set of steps. (The terms step, operator, and task are synonyms.) A linear plan consists of a totally ordered set of operators; a nonlinear plan consists of a partially ordered set of operators. In the STRIPS model (Fikes, 1971), which has been the basis of all classical planners (e.g., Sacerdoti, 1975, Tate, 1977, Vere, 1983, Chapman, 1987, Wilkins, 1988), each step has a set of preconditions and effects. Preconditions are propositions representing conditions that must be true before the step is performed. Effects are divided into an add-list, containing propositions that are made true by the step, and a delete-list, containing propositions that are made false by the step. The initial state, \(I\), specifies the initial truth values of propositions; the goal state, \(G\), specifies the desired final truth values of propositions.

Situation-dependent operators, or operators whose effects are dependent on the situation in which they are performed, are disallowed in most previous planners as the associated reasoning mechanisms become computationally intractable (Chapman, 1987). SIPE (Wilkins, 1988) is the notable exception, and provides a variety of heuristics for managing the complexity resulting from an expressive representation. However, if we restrict our focus to situation-dependent operators that consume and produce resources, which are
ubiquitous in many real-world domains (e.g., construction planning, process planning),
efficient and provably correct planners can be developed.

2.2.2. Extending the STRIPS Representation for Resources

It is often natural to express goals (e.g., the desired quantity of some product) in terms of
resources. The STRIPS representation may be extended to allow the specification of
effects to consume, utilize, or produce quantities of resources. Consumable resources,
one once consumed are unavailable to other tasks, while reusable resources are available to
other tasks after the task that utilized them is completed. Note that a reusable resource is
equivalent to a consumable resource that is both consumed and produced in equal amounts
by every step that utilizes it. For example, a lay concrete step first "consumes" the concrete
mixer, and then "produces" it upon completion, making it available for use by other steps.

Step preconditions must now also specify, in addition to a set of propositions, the amounts
of resources that must be available in order for a step to be performed. Effects must now
specify the amounts of resources that are consumed, utilized, or produced by a step.
Similarly, the initial state, I, must now specify the initial resource allocations, and the goal
state, G, must specify the desired final amounts of resources. Note that the initial resource
allocations are de facto upper bounds on the consumption (or maximum utilization) of those
resources.
2.3. A Linear Resource-Manipulating Planner

The key concepts in this section are those of:

1) *Weakest precondition*, a minimal precondition for a partial plan to achieve its goal. The weakest precondition is useful in reasoning about the correctness of plans (e.g., a plan $f$ is a solution if $G'$, the weakest precondition for $f$ to achieve $G$ is satisfied by the initial state).

2) *Guaranteed resource availability* (GRA), or the amount of a resource that can be guaranteed to be available when some step is performed. The GRA is useful in determining when there may be a potential deficit of some resource when some step is performed.

I build upon the work of McAllester (McAllester, 1991), who developed a simple, formally precise, provably sound and complete, linear, STRIPS-based propositional planner. McAllester's planner is based on ideas developed by Rosenschein (Rosenschein, 1981) who pioneered the use of a propositional dynamic logic representation for plans. Rosenschein uses the concept of a plan's *weakest precondition*\(^1\), or a minimal precondition for a partial plan to achieve its goal (i.e. every other sufficient precondition is a superset of the weakest precondition). The weakest precondition specifies what must be true in order for a plan to achieve its goal, and is useful in reasoning about the correctness of plans. In a planning with resources problem, a plan's weakest precondition would include not only a set of propositions, but also the amounts of the resources that are required in order for the

---

\(^1\) The concept of weakest precondition was first introduced by (Dijkstra, 1975) in reasoning about the correctness of programs.
plan to achieve its goal. If the level of some resource $r$ specified in the plan's weakest precondition is greater than the amount of $r$ specified in the initial state, then the plan has a resource deficit with respect to $r$, and more producers of $r$ must be prefixed to the plan in order to satisfy its resource requirements.

Before defining the resource requirement of a plan, I need to first introduce the concept of guaranteed resource availability (GRA).

**Definition.** GRA($S, r$), for some step $S$ and some resource $r$ is the amount of $r$ present in the state immediately preceding the execution of $S$, which is equal to the amount of $r$ produced by steps preceding $S$ minus the amount of $r$ consumed by steps preceding $S$.

GRA($S, r$) is the amount of $r$ that can be guaranteed to be available when step $S$ is performed.

**Definition.** A plan's resource requirement for a resource $r$ is equal to the minimum over steps $C$ that compete for $r$, of the quantity GRA($C, r$) - PRECONDITION($C, r$), where PRECONDITION($C, r$) is the amount of $r$ required by step $C$. The resource availability requirement will either be the maximum deficit of any step with respect to $r$, or the minimum surplus (if there are no deficits).

**Definition.** $V$ is the weakest precondition for a plan $h$ to achieve $G$ if there is no other sufficient precondition for $W$ for $h$ to achieve $G$, such that $W$ is a weaker precondition than $V$. 
Definition. V is *weaker* than W if the following are true:

1) the propositions contained in V are a subset of the propositions contained in W

2) each resource requirement in V is less than or equal to the corresponding resource requirement in W

3) V \models W

2.3.1. Soundness and Completeness Invariants

This section defines the concepts of soundness and completeness with respect to linear planning with resources, and gives an overview of the linear planning algorithm.

The planner maintains a list PARTIAL-PLANS that contains all of the candidate plans under construction. The following two invariants (also used in McAllester's linear propositional planner) are the crux of the resource-manipulating linear planner:

**Soundness Invariant.** If <G', h> is an element of PARTIAL-PLANS then G' is the weakest precondition for h to achieve G.

**Completeness Invariant.** If f is a finite minimum length solution (i.e. no other solution has fewer steps than f) to the planning problem then there exists some entry <G', h> in PARTIAL-PLANS such that h is a suffix of f (e.g. if f were the linear plan a;b;c, then h could be the null plan, the plan c, the plan b;c, or the plan a;b;c).
The soundness invariant implies that if there is an entry \(<G', h>\) in PARTIAL-PLANS, such that \(G'\) is satisfied by \(I\) (the initial state of the planning problem), then \(h\) is a solution to the planning problem. \(G'\) will be satisfied by \(I\) if the propositions in \(G'\) are a subset of the propositions in \(I\), and the amount of each resource \(r\) required by \(G'\) is less than the amount of \(r\) allocated in \(I\). The soundness criterion only requires that \(G'\) is a sufficient precondition for \(h\) to achieve \(G\). The completeness invariant guarantees that there exists some entry in PARTIAL-PLANS such that a solution can be found by adding new steps to the front of the plan.

The search procedure runs by iteratively removing elements from PARTIAL-PLANS. When an element \(<G', h>\) is removed from PARTIAL-PLANS, new elements are added to PARTIAL-PLANS of the form \(<G'', a;h>\) where \(a\) is a new step being prefixed to the plan, and both of the above invariants are maintained. This removal process makes the plans that appear in PARTIAL-PLANS get longer. A breadth first search (in which PARTIAL-PLANS is treated as a queue) will eventually remove all the short plans and, assuming that some solution exists, the completeness invariant ensures that a solution will eventually be found (McAllester, 1991).\(^2\)

\(^2\) Korf (Korf, 1985) has shown that an iterative-deepening-A* search is significantly more space efficient (it uses linear space) than breadth-first-search (which uses exponential space). However, to simplify the description of the Synapse algorithms, I have used breadth-first-search.
2.3.2. The Linear Planning Algorithm

The linear planning with resources algorithm proceeds as follows: ³

1) Initialize PARTIAL-PLANS to the set \(<G, \text{the null plan}>\).

2) Remove a subgoal \(<G', h>\) from PARTIAL-PLANS.

3) If \(G'\) is satisfied by \(I\) then terminate and return \(h\).

4) Otherwise, add elements to PARTIAL-PLANS in a way that maintains the invariants. For each step \(S\) such that \(S\) does not delete any proposition in \(G'\), and either \(S\) produces a resource for which a deficit exists (i.e. a positive resource requirement in the weakest precondition \(G'\)) or \(S\) asserts a proposition in \(G'\):

   a) Construct a new weakest precondition \(G''\) as follows:

      i) The set of propositions in \(G''\) will be those contained in \(G'\) minus the propositions asserted by \(S\), plus the propositional preconditions of \(S\).

      ii) For each resource \(r\) that \(S\) produces in quantity \(k\), decrease the amount of \(r\) required in \(G'\) by \(k\) (since the deficit with respect to \(r\) has been decreased). (If \(G'\) does not specify a requirement for some resource, it is assumed to be zero.⁴)

      iii) For each resource \(r\) that \(S\) consumes in quantity \(k\), increase the amount of \(r\) required in \(G'\) by \(k\) (since the deficit with respect to \(r\) has been increased).

   b) Add \(<G'', S;h>\) to PARTIAL-PLANS.

   c) Go to step 2.

2.4. A Nonlinear Resource-Manipulating Planner

Nonlinear planning uses an equivalence relation on plans to divide the search space (i.e. nonlinear planning searches the space of equivalence classes of plans rather than the space of plans). Since many different plans can all be equivalent, searching the space of equivalence classes (nonlinear plans) can be much more efficient than searching the space of linear plans (McAllester, 1991). McAllester's nonlinear planner is used as the starting point for Synapse for two reasons:

³ The linear planning with resources algorithm is proved sound and complete in Appendix B.
⁴ A negative resource requirement indicates a surplus.
1) Its sophisticated representation of nonlinear plans.

2) It represents a performance improvement over TWEAK (Chapman, 1987) without sacrificing theoretical rigor. McAllester exploits the observation that certain choices in the search space need never be backtracked.

Two linear plans may be considered equivalent if they are different representations of the same nonlinear plan. That is, all (totally ordered) linearizations of a (partially ordered) nonlinear plan are equivalent, provided that the total order respects the precedence constraints in the partial order.

2.4.1. Nonlinear Plan Representation and Terminology

This section defines several key concepts in nonlinear planning, including: nonlinear plan, causal link, establisher, establishee, open precondition, unsafe causal link, clobberer, supplier, competitor, guaranteed resource availability (GRA), and resource deficit. The two most important data structures defined in this section are:

1) casual links, which indicate the propositional dependency structure of a nonlinear plan (McAllester, 1991)

2) guaranteed resource availability (GRA) values, which are used to detect resource deficits

The nonlinear planner and plan integrator (described in Section 3.4) are described in terms of these concepts (as is the completeness proof in Appendix B).
The partial order is the element of a nonlinear plan that distinguishes it from a linear (totally ordered) plan. Causal links indicate the propositional dependency structure of a nonlinear plan (McAllester, 1991). Guaranteed resource availabilitys (GRA values) are a key component of the mechanism for reasoning about resources. The GRA is compared to the amount specified in the step precondition to determine whether a resource deficit exists.

**Definition.** A nonlinear plan (for a planning with resources problem) consists of:

1) A set of nodes, called NODES. The initial node I has no preconditions, and has the effect of asserting the propositions and allocating the resources specified in the initial state, and the final node G has no effects, and has preconditions specifying the propositions and resource quantities specified in the goal state (McAllester, 1991).

2) A partial order < on nodes, represented by a set of precedence constraints, called CONSTRAINTS. If i < j then i must precede j and j must follow i, otherwise i might follow j and j might precede i. The initial node I must precede every other node, and the final node G must follow every other node.

3) A set of causal links, called LINKS.

4) A resource availability table, which indicates the GRA values for each node with respect to each resource.

**Definition.** A causal link in a nonlinear plan is of the form <i, p, j> where i and j are nodes such that i must precede j, i asserts p, and p is a precondition of j. i is the establisher of j, the establishee.
Definition. An open precondition is a pair \(<p, j>\) where \(p\) is a precondition of \(j\) and there is no causal link of the form \(<i, p, j>\).

Definition. A causal link \(<i, p, j>\) is unsafe if there is some clobberer \(k\) that deletes \(p\) which might occur between \(i\) and \(j\).

The dashed lines in Figure 2.3 below denote two possible precedence constraints that could be added to make the causal link safe. Figure 2.4 shows how the causal link can be made safe by demoting the clobberer; or adding the constraint \(k < i\). Figure 2.5 shows how the causal link can be made safe by promoting the clobberer, or adding the constraint \(j < k\).

---

5 The terms clobberer, demotion, and promotion were coined by Chapman (Chapman, 1987).
Definition. A causal link $<i, p, j>$ in a linear plan is never unsafe. That is, there is no step $k$, such that $k$ is between $i$ and $j$, and $k$ negates $p$. 
**Definition.** A producer $S$ of some resource $r$ is a *supplier* for each consumer or utilizor $C$ of $r$ that must follow $S$.

![Diagram of a supplier](image)

**FIGURE 2.6: A Supplier**

**Definition.** A consumer $C$ of some resource $r$ is a *competitor* of any consumer $C'$ of $r$ that might follow $C$.

![Diagram of mutual competitors](image)

**FIGURE 2.7: Mutual Competitors**

**Definition.** The *guaranteed resource availability (GRA)* of a node $S$ with respect to a resource $r$ is the amount of $r$ produced by the suppliers of $S$ minus the amount of $r$ consumed by the competitors of $S$.\(^6\)

\(^6\) The GRA for nonlinear plans is essentially the resource-cliche truth criterion discussed by Chapman (Chapman, 1987).
Since the exact order of execution is unspecified in a nonlinear plan (i.e. it can't always be determined whether i precedes j, or vice versa), the best we can do is compute a lower bound on the guaranteed minimum level of a resource store in a given state (whereas exact GRA values may be computed for a linear plan).

**Definition.** A *resource deficit* occurs when there is a consumer C of some resource r such that \( \text{GRA}(C, r) < \text{PRECONDITION}(C, r) \), where \( \text{PRECONDITION}(C,r) \) is the amount of r required by node C.

### 2.4.2. Soundness and Completeness

This section defines the concepts of soundness and completeness with respect to nonlinear planning with resources, and gives an overview of the nonlinear planning algorithm.

The nonlinear planner also maintains a list PARTIAL-PLANS that contains all of the candidate plans under construction. The following *soundness lemma* captures the notion of correctness for nonlinear resource-manipulating plans. The intuition is that all preconditions must be satisfied in a correct plan: propositional preconditions are satisfied when there is a safe causal link, and resource preconditions are satisfied when there is no deficit.

**Soundness Lemma for Nonlinear Plans.** *If P is a nonlinear plan with no open preconditions, unsafe causal links, or resource deficits, then any linearization f of P is a solution to the planning problem.*
This soundness lemma removes the need for a soundness invariant on the search process. A nonlinear planner can be based on the following completeness invariant (McAllester, 1991).

**Completeness Invariant.** If \( f \) is a minimal finite sequence of steps that solves the planning problem, then there exists some nonlinear plan \( P \) in PARTIAL-PLANS such that \( f \) is a *linear interpretation* of \( P \).

Intuitively, \( f \) is a linear interpretation of a nonlinear plan \( P \) if \( f \) has at least as many steps as there are nodes in \( P \), such that:

1) every node in \( P \) corresponds to a step in \( f \)

2) \( f \) respects the partial order of \( P \)

3) every causal link in \( P \) has a corresponding causal link in \( f \)

**Definition.** A sequence of operations \( f = a_1;...;a_n \) is a *linear interpretation* of a nonlinear plan \( P \) if there are at least as many steps in \( f \) as nodes in \( P \) and there exists an assignment of integers to the nodes of \( P \) such that (McAllester, 1991):

1) node 1 has number zero, node \( G \) has number \( n + 1 \) (where \( n \) is the length of the sequence \( f \)), and every other node is assigned a number between 1 and \( n \) inclusive.

2) the step \( a_i \) in the sequence \( f \) is associated with node \( i \) in \( P \).

3) the partial order of \( P \) is respected (and no two nodes have the same number). Thus, if \( P \) contains the precedence constraint \( i < j \), then \( a_i \) must precede \( a_j \) in \( f \).

4) every causal link \( <i, p, j> \) in \( P \) has a corresponding causal link \( <a_i, p, a_j> \) in \( f \). That is, if \( i \) is an establisher of \( j \), then \( a_i \) is an establisher of \( a_j \).
The search procedure runs by iteratively removing elements from PARTIAL-PLANS. When a plan $P$ is removed from PARTIAL-PLANS, new plans are added to PARTIAL-PLANS, where each new plan is obtained by applying one of the following plan modifications to $P$:

1) adding a new precedence constraint

2) adding a new node and a precedence constraint
This removal process makes the plans that appear in PARTIAL-PLANS get longer. A breadth first search (in which PARTIAL-PLANS is treated as a queue) will eventually remove all the short plans and, assuming that some solution exists, the completeness invariant ensures that a solution will eventually be found.

2.4.3. The Nonlinear Planning Algorithm

The nonlinear planning with resources algorithm proceeds as follows:7

1) Initialize PARTIAL-PLANS to the set \( P_0 \), where \( P_0 \) contains only the nodes I and G.

2) Remove some plan \( P \) in PARTIAL-PLANS.

3) Return \( P \) if it has no open preconditions, unsafe causal links, or resource deficits.

4) If \( P \) has unsafe causal links, explore all ways of eliminating these unsafe links as follows:
   a) Select some causal link \(<i, p, j>\) and a node \( C \) that deletes \( p \) and might be between \( i \) and \( j \).
   b) If \( C \) might precede \( i \) add the plan \( P' \) to PARTIAL-PLANS where \( P' \) is \( P \) with the additional precedence constraint \( C < i \) (this is called denoting the clobberer).
   c) If \( C \) might follow \( j \) add the plan \( P' \) to PARTIAL-PLANS where \( P' \) is \( P \) with the additional precedence constraint \( j < C \) (this is called promoting the clobberer).
   d) Go to step 2.

5) If \( P \) has open preconditions, explore all ways of eliminating these open preconditions as follows:
   a) Select some open precondition \(<p, j>\).
   b) For each node \( i \) in \( P \) that adds \( p \) and might precede \( j \), create a new plan \( P' \) by adding the new causal link \(<i, p, j>\), and add \( P' \) to PARTIAL-PLANS (i.e. choose an existing node \( i \), and make it the establisher of \( j \)).
   c) For each step \( a \) that asserts \( p \), create a new plan \( P' \) by adding a new node \( i \) that corresponds to step \( a \), the precedence constraint \( i < j \), and the causal link \(<i, p, j>\), and add \( P' \) to PARTIAL-PLANS (i.e. add a new node \( i \), and make it the establisher of \( j \)).
   d) Go to step 2.

6) If \( P \) has resource deficits, explore all ways of reducing these deficits as follows:
   a) Select a deficit at some node \( i \) with respect to resource \( r \).

---

7 The nonlinear planning with resources algorithm is proved complete in Appendix B.
b) For each competitor node C of i, add the plan P' to PARTIAL-PLANS where P' is P with the additional precedence constraint i < C (i.e. choose a competitor C, and force it to follow i).

c) For each node S in P that produces r and might precede i, add the plan P' to PARTIAL-PLANS where P' is P with the additional precedence constraint S < i (i.e. choose a producer S, and force it to precede i, thus becoming a supplier of i).

d) For each step a that produces r, create a new plan P' by adding a new node S that corresponds to step a, and the precedence constraint S < i, and add P' to PARTIAL-PLANS (i.e. add a new producer S, and make it a supplier of i).

e) Go to step 2.

The algorithm initially adds new nodes at step 5, if there are any open preconditions, or step 6, if there are resource deficits and no remaining open preconditions. If adding a step causes an existing causal link to become unsafe, the algorithm will add precedence constraints at step 4 until all causal links are safe. Unsafe links are handled first because the branching factor for handling unsafe links is bounded (there are either 0, 1, or 2 ways to secure an unsafe link). Open preconditions are handled before resource deficits because the propositional dependency structure of a plan is more constraining than its resource dependency structure.

Whenever a plan is modified, the set of unsafe links, open preconditions, and GRA values (and therefore the set of resource deficits) may change. For example, if a new node is added, its preconditions will denote new open preconditions or resource deficits, unless they are satisfied by the initial state (the only node that is guaranteed to precede a new node). The new node may also cause new unsafe links by clobbering a propositional precondition of some other node, or resource deficits by competing for the same resource as some other node. Similarly, when a new precedence constraint i < j is added, open preconditions, unsafe causal links, and resource deficits may be eliminated (e.g., a deficit at node j will be reduced if i now supplies j, and a deficit at node i will be reduced if j no longer competes with i).
2.4.4. Performance

Since planning is NP-hard (Chapman, 1987), in order to achieve reasonable efficiency on large problems, it is necessary to use one of two approaches:

1) Employ heuristics to reduce the size of the search space. This is the approach taken by SIPE (Wilkins, 1988), one of the only previous planners that has been applied to real-world problems of significant complexity. SIPE uses many heuristics (necessitated by its use of an expressive representation), including hooks for domain-specific heuristics. The heuristics used in Synapse are discussed in Section 4.3.1.

2) Decompose the problem into subproblems, and integrate the solutions to these subproblems. The plan integration framework described in Chapter 3 would be useful in such an approach.

2.4.5. Extensions

In this section, I consider two useful extensions to the nonlinear planning with resources framework:

1) Allowing scalable operators, whose consumption and production may vary according to consumption and production constraints. A new plan modification is now available to reduce a deficit: rescaling, or increasing the production of an existing node. This capability is also important in the plan integrator (described in Section 3.4).
2) Reformulating unsolvable planning problems so that they become solvable.

These are described in the following two subsections.

2.4.5.1. Scalable Operators

Allowing scalable operators gives the nonlinear planning algorithm greater flexibility in step 6 (which reduces resource deficits):

1) When adding a new node \( i \) to address a resource deficit, \( i \) may now specify any level of production consistent with the production constraints. The actual amount produced should minimize the discrepancy between the size of the deficit and the amount produced by \( i \). This will allow fewer nodes to be used in the plan, if the production constraints allow production levels that approximate or match the actual deficit.

2) Resource deficits may also be addressed by increasing the production of, or rescaling, some node in \( P \). Consumption constraints determine how the consumption of a node should be modified when its production is increased.

A scalable operator representation scheme allows operators to consume and produce variable (rather than fixed) amounts of resources, as specified by consumption and production constraints. Consumption constraints associated with an operator schema specify how the amount of resource consumed depends on the level of some other
quantity. For example, the lay concrete task (described in Appendix A), specifies that the consumption of concrete is the size (in square feet) of the building under construction, scaled by the constant .029. Consumption constraints are analogous to function constraints in SIPE (Wilkins, 1988), where the value of a variable is constrained to be the value of some function. Production constraints specify limitations on a step's production capacity, which may be represented as restrictions on the amount of resource produced to either a discrete set of alternative values (e.g., discrete lot-sizing constraints) or a continuous range of alternative values (this is analogous to numerical range constraints in SIPE). Such generalized operators more closely model the behavior of real-world operations (e.g., in manufacturing).

2.4.5.1.1. Extending the Nonlinear Planner for Scalable Operators

Scalable operators give the nonlinear planning algorithm a new deficit reduction option in step 6:

6c) For each node \( S \) in \( P \) such that \( S \) is already a supplier of \( r \) to \( i \), construct a new plan \( P' \) by increasing the production of \( S \) by the minimal amount allowed by the production constraints of \( S \), and add the plan \( P' \) to PARTIAL-PLANS.

The definition of linear interpretation used in the completeness invariant would now need an additional condition to reflect the new plan modification of rescaling a node (previously the only modifications allowed were adding a constraint, or adding a node and a constraint):

5) if \( i \) produces some amount \( K \) of a resource \( r \), then \( a_i \) must produce \( K' \) such that \( K' \geq K \). That is, the resource quantities produced by nodes in \( P \) are lower bounds on the resource quantities manipulated by steps in \( f \). (The resource quantities produced by a node will increase if it is rescaled.)

---

8 To guarantee completeness, I adopt the convention that all resource quantities are integral values. This ensures that there is always some minimum amount by which production will be increased (and enables us to prove that the planner always performs a finite number of production increases).
2.4.5.2. Reformulating Unsolvable Planning Problems

There will not always be a solution to a given planning problem. For example, if all partial plans in PARTIAL-PLANS contain an *unproducible resource deficit* with respect to r such that no step produces r, then no solution exists. Rather than merely signalling failure at this point, it would be useful to suggest how the original planning problem could be reformulated so that a solution exists. A problem reformulation capability is especially useful in domains with resources, since incremental changes in production goals or initial resource allocations may yield a solvable problem.

If a partial plan P achieves G but has an unproducible deficit resource r, we can make P into a solution of a related planning problem in which the initial level of r in I is increased by the amount of the deficit. An alternative approach would be to scale down some of the steps consuming (or utilizing) the unproducible resource, resulting in a plan that achieves a new goal G' that is weaker than G.

2.4.5.2.1. The Problem Reformulation Algorithm

This section extends the nonlinear planning with resources algorithm for problem reformulation in the presence of unproducible resource deficits. Step 6 is modified so that a plan is added to the list SUSPENDED-PLANS (instead of the list PARTIAL-PLANS) if it contains an unproducible resource deficit which cannot be reduced by reordering other competitors. Now if PARTIAL-PLANS ever becomes empty, then the planning problem with initial state I and goal state G has no solution. However, by analyzing and modifying the entries in SUSPENDED-PLANS, the planner can suggest reformulations of the original problem, in the form of modifications to I and/or G, so that a solution can be found. The
problem reformulation algorithm is the same as the planning algorithm described earlier, except that step 6 is replaced by the following:

6) If P has resource deficits, explore the following ways of reducing these deficits:

   a) Select a deficit resource r.

   b) For each node C in P that consumes r construct a new plan P' by decreasing the consumption of C by the minimal amount allowed by the production and consumption constraints of C, and add the plan P' to SUSPENDED-PLANS. This will reduce or remove the deficit with respect to r.

   c) If the initial node I produces r, construct a new plan P' by increasing the production of I by the minimal amount needed to remove the deficit, and add the plan P' to SUSPENDED-PLANS.

   d) Go to step 2.

2.5. An Example: Planning the Construction of a Warehouse and a School

This section illustrates the process of plan construction for two plans in the construction domain: a plan to build a warehouse and a plan to build a school. (Section 3.5 will illustrate conflicting and synergistic interactions between these two plans.)

Below I describe a simple construction planning scenario that will later be used to illustrate collaborative planning:

Two contractors working for the same company develop separate plans to build a warehouse and a school. The two plans both contain some generic construction tasks, as well as some tasks that are specific to the particular building under construction.
The initial state of the warehouse construction plan, shown below in Figure 2.9 (a Synapse screen image), specifies the initial allocations of four resources:

<table>
<thead>
<tr>
<th>Value</th>
<th>Resource or Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[Cement Mixer]</td>
</tr>
<tr>
<td>10000</td>
<td>[Concrete]</td>
</tr>
<tr>
<td>2</td>
<td>[Cranes]</td>
</tr>
<tr>
<td>600</td>
<td>[Steel]</td>
</tr>
</tbody>
</table>

**FIGURE 2.9: Initial State of the Warehouse Plan**
Similarly, Figure 2.10 below shows the goal state of the warehouse construction plan, which specifies that various propositions relating to the construction of the warehouse should be true:

<table>
<thead>
<tr>
<th>Value</th>
<th>Resource or Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>[Building Finish Completed?]</td>
</tr>
<tr>
<td>TRUE</td>
<td>[Electrical Completed?]</td>
</tr>
<tr>
<td>TRUE</td>
<td>[Fire Protection Completed?]</td>
</tr>
<tr>
<td>TRUE</td>
<td>[Roofing Completed?]</td>
</tr>
<tr>
<td>TRUE</td>
<td>[Special Floors Completed?]</td>
</tr>
</tbody>
</table>

**FIGURE 2.10: Goal of the Warehouse Plan**

The tasks used in this planning problem include some generic construction tasks (e.g., lay concrete, roofing), as well as some tasks that are specific to the particular building under construction (e.g., start special floors, start special rooms). All of these tasks are described in Appendix A.
Successive iterations of the planning algorithm will add (at step 5c, which addresses open preconditions) new task instances to establish the goal propositions. Each of these tasks, in turn, may have open preconditions, which require additional tasks to establish those preconditions, etc., until the plan shown below (computed by Synapse) in Figure 2.11 is obtained:

FIGURE 2.11: The Warehouse Plan

---

9 These task instances are parameterized according to the process described in Sections 4.2.2.2 and 4.2.2.3. Descriptions of the task types can be found in Appendix A.
The resource usage of the warehouse plan (taken from the plan's resource manipulation table) is shown below in Figure 2.12:

![Table showing resource usage in the warehouse plan](image)

**FIGURE 2.12: Resources Used in the Warehouse Plan**

In this example, there are no unsafe links (since no task negates any precondition of another task), and there are no resource deficits (since initial state satisfies all resource preconditions).

If the initial state had allocated 4000 tons of concrete, instead of 10000 tons, then the warehouse construction problem would be unsolvable, since the plan requires 4350 tons of concrete. In this had been the case, the problem reformulation algorithm would increase the allocation of concrete to 4350 tons, and then solve the reformulated problem.
Synapse can plan the construction of the school in a similar fashion. The school and warehouse plans share the same initial state. The goal of the school plan, shown below in Figure 2.13, is similar to the warehouse plan's goal, except that the school plan omits the special floors complete proposition, and adds two extra propositions: special rooms complete and brickwork complete.

FIGURE 2.13: The Goal of the School Plan

```plaintext
<table>
<thead>
<tr>
<th>Value</th>
<th>Resource or Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>[Brickwork Completed?]</td>
</tr>
<tr>
<td>TRUE</td>
<td>[Building Finish Completed?]</td>
</tr>
<tr>
<td>TRUE</td>
<td>[Electrical Completed?]</td>
</tr>
<tr>
<td>TRUE</td>
<td>[Fire Protection Completed?]</td>
</tr>
<tr>
<td>TRUE</td>
<td>[Roofing Completed?]</td>
</tr>
<tr>
<td>TRUE</td>
<td>[Special Rooms Completed?]</td>
</tr>
</tbody>
</table>
```
The resource usage of the school plan is shown below in Figure 2.14.

<table>
<thead>
<tr>
<th>Value</th>
<th>Resource or Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>18420</td>
<td>[Carpenter Hours]</td>
</tr>
<tr>
<td>1</td>
<td>[Cement Mixer]</td>
</tr>
<tr>
<td>7250</td>
<td>[Concrete]</td>
</tr>
<tr>
<td>2</td>
<td>[Cranes]</td>
</tr>
<tr>
<td>10490</td>
<td>[Electrician Hours]</td>
</tr>
<tr>
<td>9910</td>
<td>[Iron Worker Hours]</td>
</tr>
<tr>
<td>25930</td>
<td>[Laborer Hours]</td>
</tr>
<tr>
<td>8000</td>
<td>[Mason Hours]</td>
</tr>
<tr>
<td>4283000</td>
<td>[Money]</td>
</tr>
<tr>
<td>6750</td>
<td>[Plumber hours]</td>
</tr>
<tr>
<td>467.5</td>
<td>[Steel]</td>
</tr>
</tbody>
</table>

**FIGURE 2.14: The Resources Used in the School Plan**
The school plan is shown below in Figure 2.15:

FIGURE 2.15: The School Plan

Alternatively, both the warehouse and school plans could have been developed manually by each contractor. Each contractor would then need to create and order their respective tasks.
3. Plan Integration

3.1. Summary

Section 3.2 defines the plan integration problem, and what it means for a plan integration problem to be solvable or satisfiable. The plan integration problem is the problem of combining two plans, $P_A$ and $P_B$, to resolve potential conflicts and exploit potential synergies, where the two plans are solutions to their own respective planning problems. A plan integration problem is satisfiable if the problem can be solved by deleting, reordering, and rescaling\textsuperscript{10} the nodes of $P_A$ and $P_B$. A plan integration problem is solvable if the problem can be solved by adding in new nodes that do not appear in $P_A$ and $P_B$, in addition to deleting, reordering, and rescaling the nodes of $P_A$ and $P_B$. Thus the concept of satisfiability is more restrictive than the concept of solvability.

Section 3.3 describes the plan checker, which detects conflicts and synergies due to interactions between plans. The conflicts are due to unsafe causal links and resource deficits, and the synergies are due to redundant nodes. A node is redundant if its desired effects may be achieved by other nodes in the plan.

Section 3.4 describes the plan integrator, which is identical to the nonlinear planner described in Section 2.4, except for the restriction that only the nodes present in the original plans may be added during the planning process. In contrast to the nonlinear planner:

\textsuperscript{10} Rescaling is described in Section 2.4.5.1, which describes the scalable operator extension to the nonlinear planner.
1) the integrator is not guaranteed to find the minimum length solution

2) the integrator is incomplete, as it may fail to find a solution even if the integration problem is solvable (although it always finds a solution if the integration problem is satisfiable)

Plan integration is appropriate if there is some nontrivial investment in the original plans, and thus the only desirable changes are those that exploit synergies and avoid conflicts between the plans (by deleting redundant nodes, reordering, and rescaling nodes). In contrast, planning "from scratch" is appropriate if the best plan for the combined problem is sought, even if that requires significant modifications (e.g., substitution of nodes) to the original plans.

Finally, Section 3.5 illustrates plan integration in the collaborative construction planning scenario (first introduced in Section 2.5).

3.2. The Plan Integration Problem

This section defines the plan integration problem, and what it means for a plan integration problem to be solvable or satisfiable.

The plan integration problem is the problem of combining two plans to resolve potential conflicts and exploit potential synergies, where the two plans are solutions to their own respective planning problems.

Nonlinear plan integration is more useful in supporting collaborative planning than linear plan integration for two reasons:
1) the steps in a nonlinear plan are partially ordered (rather than totally ordered, as in a linear plan), thus allowing a natural representation of the parallelism inherent in most collaborative plans

2) as discussed in Section 2.4, nonlinear planning is more efficient than linear planning

**Definition.** A plan integration problem \(<P_A, P_B>\) is the problem of combining two plans, \(P_A\) and \(P_B\), which are minimal solutions to the planning problems \(<I_A, G_A, \{S_A\}>\) and \(<I_B, G_B, \{S_B\}>\), into an integration \(P_{int}\) that solves the problem \(<I_{int}, G_{int}, \{S_{int}\}>\), such that:

1) \(I_A\) and \(I_B\) are consistent. That is, they are either identical, or they describe different subsets of the initial state relevant to their respective planning problems. In the latter case, \(I_{int}\) will be the union of the propositions and resource quantities specified in \(I_A\) and \(I_B\).

2) \(G_{int}\) includes the aggregate goals of \(G_A\) and \(G_B\): the conjunction of the propositional goals of \(G_A\) and \(G_B\)\(^{11}\), as well as the aggregated production goals of \(G_A\) and \(G_B\). (i.e. if both \(G_A\) and \(G_B\) specify lower bounds on the production of some resource, then the corresponding lower bound in G will be the sum of the lower bounds in \(G_A\) and \(G_B\)).

3) \(\{S_{int}\} = \{S_A\} U \{S_B\}\). The steps available in the integration problem include the steps available in the individual planning problems.

**Definition.** A plan integration problem \(<P_A, P_B>\) is *solvable* if the planning problem \(<I_{int}, G_{int}, \{S_{int}\}>\) is solvable.

---

\(^{11}\) If \(G\) contains a contradiction (i.e. \(G_A\) contains the negation of a propositional goal of \(G_B\)), then the plans \(A\) and \(B\) are unintegrable, and one of the contradictory goal propositions must be removed from \(G_A\) or \(G_B\) in order to enable successful integration.
**Definition.** A plan integration problem \(<P_A, P_B>\) is *satisfiable* if planning problem \(<I_{int}, G_{int}, (\text{NODES}_{AB})>\) is solvable, where \((\text{NODES}_{AB})\) is the union of the sets of nodes contained in \(P_A\) and \(P_B\).

Thus the concept of satisfiability is more restrictive than the concept of solvability, since the choice of nodes to be added to a plan is restricted to a smaller set.

### 3.3. A Plan Checker

The plan checker detects potential conflicts and synergies between two plans. Two types of conflicts\(^\text{12}\) may arise during plan integration:

1) unsafe causal links, if a step in one plan negates a precondition of a step in the other plan

2) resource deficits, if a step in one plan competes for the same resource as a step in the other plan

For further discussion of unsafe links and resource deficits see Section 2.4.1. Synergies arise due to redundant nodes, as discussed below.

\(^{12}\) Note that open preconditions cannot be caused by interactions between two plans.
3.3.1. Redundant Nodes

**Definition.** A node i is *redundant* if:

1) There is an *alternate establisher* e for each causal link <i, p, j> that i establishes such that:
   
   a) e asserts p
   
   b) e might precede j
   
   c) for each clobberer node k, such that k is possibly before j and k negates p: either e might follow k, or k might follow j.

2) For each resource r produced by i and each consumer C of r that must follow i, it is possible to reorder or rescale nodes as follows to avoid a deficit of r at C (i.e. if i is deleted, then no additional deficits will be incurred with respect to any resources it produces):
   
   a) adding precedence constraints of the form C < C' (where C' is a competitor of C with respect to r)
   
   b) adding precedence constraints of the form S < C (where S is a producer of r)
   
   c) rescaling some supplier S of C

If a node meets the above requirements, then all of its desired effects may be achieved by other nodes in \{NODES\}. 
For example, in Figure 3.1 below, both i and k are redundant because both nodes assert p and might precede j.

![Diagram](image)

**FIGURE 3.1: Two Redundant Nodes**

Redundant nodes commonly occur when two plans each contain a setup operation that only needs to be performed once. Other common examples include any pair of nodes that are different instantiations of the same operator schema. For example, both the warehouse and school plans shown in Figures 2.11 and 2.15 contain several instances of generic construction tasks, such as roofing, lay concrete, etc. Other examples might include tool or machine setup operations in manufacturing, telescope positioning operations in astronomical observation applications (e.g., Johnston, 1990), etc.
3.3.2. The Plan Checking Algorithm

The plan checking algorithm proceeds as follows:

1) For each causal link in one plan, add an unsafe link for each of O(N) potential clobberers in the other plan. The number of causal links is O(NP), where N is the number of nodes in the plan, and P is the maximum number of propositional preconditions at any node. Thus, checking for unsafe links is O(N^2P).

2) Adjust the GRA values of each node with respect to its competitors in the other plan, noting any new resource deficits. This requires O(N^2R) GRA adjustments, where R is the maximum number of resource preconditions at any node, since there are O(N) competitors for each node.

3) Check to see if each node i is redundant, as follows (using the definition given in Section 3.3.1):
   
a) Find an "alternate establisher" e for each causal link <i, p, j> that i establishes. There are O(P) such links, O(N) potential alternate establishers, O(N) potential clobberers, so the complexity of this operation is O(N^2P).

   b) For each node C that was formerly supplied by i, check if the competitors and alternate suppliers of C can be reordered to avert a new deficit at C. There are O(R) resources supplied by i, O(N) nodes supplied by i, O(N) competitors and alternate suppliers of C, so the complexity of this operation is O(N^2R).

Thus, checking for redundant nodes is O(N^3K), where K = P + R (since steps a and b are performed for each node).

Thus the worst-case computational complexity of the plan checker is O(N^3K).

It is also possible to check or integrate two plans that are "incorrect" (i.e. do not achieve all of their own goals) or contain unexploited synergies. In this case, it may be useful to discover the inter-plan conflicts and synergies among partial plans that are under development.

Checking or integrating N plans can be accomplished by O (log N) pairwise calls to the plan checker or integrator. This is an efficient way of dividing up a complex integration problem, especially if N is large.
3.3.3. A Single-Plan Checking Algorithm

Since Synapse allows a user to manually create a plan, it is useful to allow the user to check such a plan for any internal conflicts or synergies. The single-plan checking algorithm proceeds as follows:

1) For each propositional precondition $p$ of each node $j$, check if there is a node $i$ such that $i$ asserts $p$ and $i$ might follow all potential clobberers $k$ that negate $p$. If not, add the open precondition $\langle p, j \rangle$.\(^\text{13}\) This is an $O(N^3P)$ operation.

2) Compute the GRA values of each node, noting any deficits as described in the multiple-plan checker (see Section 3.3.1).

3) The set of possibly redundant nodes is computed as in the multiple-plan checker (see Section 3.3.1).

3.4. Two Approaches to Plan Integration

This section describes two approaches to plan integration: one provably correct, and the other heuristic.

3.4.1. A Provably Correct Plan Integrator

The plan integrator is identical to the nonlinear planner described in Section 2.4, except that the only steps that may be added to a plan are those that correspond to some node in the original plans (i.e. any step that corresponds to a node in the set $\{\text{NODES}_{AB}\}$). (The nonlinear planner, in contrast, can add any step in the domain.) Recall that the completeness invariant of the nonlinear planner (stated in Section 2.4.2) guarantees that the planner always finds the globally minimal length solution. The integrator, however, is not guaranteed to find the globally minimum length solution, but will instead find the "locally" minimal length solution with respect to the steps represented in $\{\text{NODES}_{AB}\}$. If the

\(^{13}\) This contrasts with multiple-plan checking (described in Section 3.4.1), where open preconditions cannot arise due to interactions between two plans.
number of nodes in this solution is less than the number of nodes in \( \{ \text{NODES}_{AB} \} \), then one or more of the nodes in \( \{ \text{NODES}_{AB} \} \) were redundant. The integrator will miss a globally minimal solution \( P_{\text{MIN}} \) if \( P_{\text{MIN}} \) requires one or more steps that do not correspond to any node in \( \{ \text{NODES}_{AB} \} \). For example, consider the problem of integrating two plans for producing different types of car bodies, and assume that both plans build the car body out of steel. It may be the case, however, that in the globally minimal plan, the car bodies must be built from plastic, using some steps that are not present in either of the original plans. The integrator will fail to find this globally minimal plan, since it only considers steps that are represented in the original plans. In contrast, the planner described in Section 2.4 would find this global minimum, since it considers all possible steps in the domain.

The integrator is incomplete, because it may fail to find a solution even if the integration problem (i.e. the planning problem \( \langle I_{\text{int}}, G_{\text{int}}, \{ S_{\text{int}} \} \rangle \)) is solvable. This is because an integration problem may be solvable, but not satisfiable,\(^{14}\) and the integrator only finds a solution if the problem is satisfiable. We conjecture that in order for an integrator to be complete (i.e. find a solution whenever the integration problem is solvable), the integrator must embed the full capabilities of a planner (e.g., it must be able to add in any step in the domain).

Plan integration is appropriate if there is some nontrivial investment in the original plans, and thus the only desirable changes are those that exploit synergies and avoid conflicts between the plans (by deleting redundant nodes, reordering, and rescaling nodes). In contrast, planning "from scratch" is appropriate if the best plan for the combined problem is sought, even if that requires significant modifications (e.g., substitution of nodes) to the original plans.

\(^{14}\) It may be the case that the integration problem can only be solved by the addition of some node that does not correspond to any node in the original plans.
3.4.2. A Heuristic Plan Integrator

Another approach to plan integration is to start out with both of the original plans placed in parallel, $P_A$ and $P_B$, and to explore ways of deleting, reordering, or rescaling nodes until no conflicts or redundant nodes remain.

**Definition.** The *parallel combination* $P_{AB}$ of two plans $P_A$ and $P_B$, is a plan consisting of the nodes, precedence constraints, and causal links of plans $P_A$ and $P_B$.

An integrator could proceed as follows:

1) Compute $P_{MIN}$, a subplan of $P_{AB}$, by successively deleting redundant nodes\(^{15}\) from $P_{AB}$ until no redundant nodes remain

2) Invoke the planner with PARTIAL-PLANS initialized to a single plan, $P_{MIN}$, and run the planner with the restriction that no new nodes be added.

I conjecture that this approach will yield a solution if exploiting a redundancy does not affect the resolvability of a conflict.\(^{16}\) In some cases, this heuristic integrator may be more efficient than the provably correct integrator.

\(^{15}\) This choice could be informed by some heuristic that evaluates the cost of the node, and estimates the distance to the goal (e.g., how the duration will increase, if time is a priority, what the impact of any foreseeable rescales might be, etc.).

\(^{16}\) I conjecture that this condition would be satisfied in many domains, but this is an empirical question. If this condition is not satisfied, an integrator that explores all possible ways of deleting *non-essential* nodes (i.e. nodes that do not appear in every integration) from $P_{AB}$ could be used.
3.5. A Collaborative Planning Example: Checking and Integrating Plans to Construct a Warehouse and a School

This section illustrates the conflicts and redundancies that may arise when two plans are checked, and how these conflicts and synergies may be integrated. The collaborative construction planning scenario, first introduced in Section 2.5, is used here for illustrative purposes. Note that plan checking and integration may be performed centrally, by an agent with access to all of the collaborating agents' plans, or in a decentralized fashion, where agents exchange their plans as needed. The decentralized approach is more flexible, while the centralized approach can detect some conflicts that may go otherwise unnoticed (e.g., a precedence loop spanning several plans).

3.5.1. Checking the Plans

This section discusses the conflicts and redundancies between the warehouse and school construction plans, shown in Figure 3.2 below. The tasks that are specific to either the warehouse or school project are in boldface (the other tasks are generic construction tasks).
FIGURE 3.2: The Warehouse and School Plans
3.5.1.1. Conflicts

All of the conflicts are resource deficits, and fall into four categories:

1) The roofing, precast walls, and structural steel tasks in both plans compete for the 2 cranes allocated in the initial state (the aggregate demand is for 5 cranes).

2) The structural steel tasks in both plans compete for the 600 tons of steel allocated in the initial state (the aggregate demand is for 748 tons of steel).

3) The lay concrete and concrete setup tasks in both plans compete for the cement mixer allocated in the initial state (the aggregate demand is for 2 cement mixers).

4) The lay concrete tasks in both plans compete for the 10000 tons of concrete allocated in the initial state (the aggregate demand is for 11600 tons of concrete).

3.5.1.2. Redundancies

The redundancies are due to pairs of tasks that appear in both plans and may potentially be mergeable (i.e. one task in the pair may be deleted, while the other may be rescaled to achieve the desired effect of both tasks). The potentially mergeable tasks are concrete setup, roofing, fire protection, electrical, building finish, precast walls, structural steel, plumbing & hvac, site earthwork, and lay concrete.
3.5.2. Integrating the Plans

The integrated plan (computed by Synapse) is shown below in Figure 3.3. The rescaled nodes are indicated in boldface.

![Diagram of integrated plans](image)

**Figure 3.3: Integrating the Warehouse and School Plans**
The conflicts that arose in Figure 3.2 have been eliminated in the integrated plan, since there are fewer tasks competing for the reusable resources (i.e. the cranes and cement mixer).

The initial resource allocations were increased by the problem reformulation mechanism. It turns out that the two plans were unintegrable, since the steel and concrete deficits were unproducible resource deficits, which could only be resolved by increasing the initial allocations of steel and concrete from 10000 tons of concrete and 600 tons of steel to 11600 tons of concrete and 748 tons of steel.

The resource usage of the integrated plan is shown below in Figure 3.4:

![Image of resource usage table]

FIGURE 3.4: Resources Used in the Integrated Plan
These quantities are smaller than the sum of the resource usage quantities of the original plans (given in Section 2.5).
4. The Synapse Implementation

4.1. Overview

Synapse is implemented in MacIntosh Allegro Common Lisp, and runs as an Object Lens application (Lai, Malone, and Yu, 1988). Object Lens provides a sophisticated user interface, including a powerful object-oriented template editor and hypertext links between objects.

Section 4.2 describes the functionality of Synapse, and the model of tasks, plans, and related objects that is presented to the user. This includes a discussion of the rationale for object type and field names, and a description of the mechanism for instantiating task instances. Section 4.3 gives the running times for the construction example (described in Sections 2.5 and 3.5), and briefly discusses performance-improving heuristics.

4.2. The User Interface

This section describes the functionality of Synapse, and the model of tasks, plans, and related objects that is presented to the user.
4.2.1. System Functionality

This section summarizes the basic functionality of Synapse (most of which was described in detail in Chapters 2 and 3). The following new Object Lens actions are defined on plans:

1) *Plan Construction*: Users may specify initial and goal states and instruct Synapse to automatically construct a plan (as discussed in Chapter 2, and illustrated in Section 2.5). Alternatively, users may manually construct their own plan by instantiating task objects and specifying precedence constraints between tasks.

2) *Plan Checking*: Users may instruct Synapse to check a plan for internal conflicts and synergies, or to check two plans for inter-plan conflicts and synergies (as discussed in Section 3.3 and illustrated in Section 3.5). The purpose of the plan-checking facility is to check manually constructed plans (since plans generated by Synapse will be correct!).

3) *Plan Integration*: Users may instruct Synapse to resolve conflicts and exploit synergies between the two plans (as discussed in Section 3.4 and illustrated in Section 3.5).

Synapse performs standard CPM calculations (Schmenner, 1990) to obtain the following quantities for each task in a plan: early start, early finish, late start, late finish, and total slack. Thus, one can view Synapse as a sophisticated enhancement to the well-established CPM project management technique. Since people are willing to enter information about their projects to obtain the benefits of CPM, then they may be willing to enter similar, albeit more detailed, information about their projects to obtain the additional benefits of Synapse.
(e.g., Synapse can improve plans with respect to all resources, whereas CPM only deals with time).

4.2.2. User Models of Tasks

Part of the challenge in building a usable prototype plan integration system lies in presenting a model of tasks, plans, and other important objects that is clear and comprehensible to the user who is not familiar with AI planning terminology. This section discusses various issues concerning the presentation of these objects to the user.

4.2.2.1. Names of Object Types and Fields

The Synapse application required the definition of the following new Object Lens types:
<table>
<thead>
<tr>
<th>Object Type</th>
<th>Key Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td>Value (true or false)</td>
</tr>
<tr>
<td>Resource</td>
<td>Value (numerical)</td>
</tr>
<tr>
<td>Prerequisite</td>
<td>Resource or Condition Value</td>
</tr>
<tr>
<td>Result</td>
<td>Resource or Condition Value</td>
</tr>
<tr>
<td>Task</td>
<td>Prerequisites(^{17}) Results(^{18})</td>
</tr>
<tr>
<td>Plan</td>
<td>Initial State(^{19}) Goal State(^{20}) Tasks(^{21})</td>
</tr>
</tbody>
</table>

Table 4.1: New Object Lens Types Required by Synapse

We discarded the AI planning term Proposition in favor of the more common term Condition. Prerequisites and Results were used as fieldnames in tasks instead of Precondition and Effect, since the former terms are more common in everyday usage. We also decided to name the types of objects that would occupy these fields Prerequisites and Results, since although there is no structural difference\(^{22}\) between the Prerequisite and Result types, we could not find a single term (in common usage) that encompasses the meaning of both Prerequisite and Result.

4.2.2.2. The Context Mechanism

\(^{17}\) Implemented as a folder containing one or more Prerequisites.
\(^{18}\) Implemented as a folder containing one or more Results.
\(^{19}\) Implemented as a folder containing one or more Prerequisites.
\(^{20}\) Implemented as a folder containing one or more Results.
\(^{21}\) Implemented as a folder containing one or more Tasks.
\(^{22}\) There are subtle semantic differences between Prerequisites and Results: quantities are always positive in Prerequisites (indicating the amount of resource required by a task), but in Results, may be either positive (indicating the production of resources) or negative (indicating the consumption or utilization of resources).
Synapse is a "ground" planner: it does not use variables\textsuperscript{23} in task prerequisites and results, since the added complexity seems unnecessary for many real-world applications. However, there is a need for some mechanism to specify the context of a proposition. For example, consider the Lay Concrete Complete proposition that is asserted by the lay concrete tasks in both the warehouse and school construction plans. The two propositions are really asserted in different contexts, a warehouse and a school, and should not be considered the same. Thus, Synapse provides a mechanism for indicating the context of a proposition or resource store.\textsuperscript{24} When Synapse (or a user) constructs a plan, propositions asserted by a task are assigned a context as follows:

1) The default context associated with the task type is used, if such a default exists. For example, in the construction scenario, the concrete setup task always asserts the Concrete Mixer Setup proposition in the context of the same global concrete mixer.

2) The context associated with the plan containing the task is used when the task is added to the plan (see Section 4.2.2.3). For example, the lay concrete task will assert the Lay Concrete Complete proposition in the context of either the warehouse or the school.

4.2.2.3. Dynamic Task Parameterization

Some attributes of a task instance may be statically inherited from the task type, while other attributes may be dynamically supplied when the task is added to a plan. For example, in the construction scenario described earlier, the names of the resources and conditions specified in each task instance's Prerequisites and Results are statically inherited from the task type (e.g., a lay concrete task instance inherits the Prerequisite of a cement mixer from the lay concrete task type). However, the contexts of propositional Prerequisites and Results may be dynamically supplied (as described in the previous section) when a task

\textsuperscript{23} If it did, codeignation constraints (Chapman, 1987, Wilkins, 1988) would be needed to constrain variable values to bind or not to bind with specific constants.

\textsuperscript{24} This allows a resource store to be partitioned into two or more disjoint stores (e.g., separate resource stores that are used by different projects).
instance is added to a plan. Similarly, the numerical values of resource Prerequisites and Results may be dynamically computed (if consumption constraints are present) when a task instance is added to a plan. A task instance's Resource Usage Rates (the everyday term for what I called consumption constraints in Section 2.4.5.1) specify how to compute the values of resources mentioned in the Prerequisites and Results. For example, the Resource Usage Rates for the lay concrete task type are given by the following formulas:

\[
\text{Money} = \text{Size} \times 2 + 15200
\]

\[
\text{Labor Hours} = \text{Size} \times 0.045 + 900
\]

\[
\text{Concrete} = \text{Size} \times 0.029
\]

Each rate specifies that some resource (e.g., money, labor hours, and concrete) is proportional to the scaling factor called "size". Scaling factors are indirect references to the value of some field in the object that is the context of the task. In the construction example, the context will either be the Warehouse or School objects. Each of these objects has a field called "size", indicating its square footage. Thus the amount of money, labor hours, and concrete in the warehouse plan's lay concrete task will be proportional to its size (150000 square feet), and the amount of money, labor hours, and concrete in the school plan's lay concrete task will be proportional to its size (250000 square feet). Another example of the use of a scaling factor in Resource Usage Rates might be the "experience" of a laborer, measured in the number of days at a given trade (e.g., carpentry, masonry, iron work, plumbing, electrical, etc.). This experience level could affect the duration and/or cost of a task.

Allowing indirect references to the fields of the context object provides a flexible mechanism for dynamically parameterizing tasks as they are added to a plan.
4.2.2.4. Sharing vs. Copying Default Field Values

The construction example defines task types corresponding to various tasks in the construction domain. Each task type specifies a set\textsuperscript{25} of default Prerequisites, Results, and Resource Usage Rates. The question arises: should the various instances of a given task type share the same set of default objects, or should they each have their own copies of the default objects? Is the answer to this question different for Synapse tasks and Object Lens objects in general? Clearly, for the Synapse application it was desirable for each task instance to have its own set of default objects, since users ought to be free to modify the task's default Prerequisites, Results, and Resource Usage Rates without interfering with the default values of other task instances. For example, each instance might specify a different context for its propositional Prerequisites and Results. The Object Lens designers decided to support both choices by providing an option on the field menu that determines whether to copy or share the default objects.\textsuperscript{26}

4.2.2.5. The Multi-User Interface: The Need for a Common Language

To the extent that collaborator planners use a common language, Synapse can detect and integrate interactions among their plans. This is not necessarily a huge limitation of Synapse, since the development and use of a common vocabulary is probably a useful thing for potential collaborators to do anyway (it is hard to imagine plan integration, or

\textsuperscript{25} Implemented as an Object Lens folder.

\textsuperscript{26} The copy function is a one-level copy, unless the object is a folder, in which case, a two-level copy is used (both the folder and its contents are copied). Fields that should have identical values in each instance (i.e. fields that should be thought of as "class variables" in object-oriented programming (e.g., Goldberg and Robson, 1983)) should be shared, while fields that may have different values in each instance (i.e. fields that should be thought of as "instance variables" (Goldberg and Robson, 1983)) should be copied.
coordination of any sort, occurring without the collaborators taking the time to translate their idiosyncratic terms into a common vocabulary).

A common terminology folder for the collaborative construction planning scenario would contain objects (e.g., task, resource, and condition instances) that should have the same name and structure in different plans (e.g., the generic construction tasks that were merged in Section 3.5, as well as the resources and conditions these tasks manipulate). Users should also share the same basic set of Synapse type definitions (e.g., resource, condition, prerequisite, result, task, plan, etc.).
4.3. Performance

Table 4.2 below gives the running times for the planner, checker, and integrator, running on a Macintosh II with 8 Megabytes of memory, on the construction example outlined in Sections 2.5 and 3.5:27

<table>
<thead>
<tr>
<th>Problem</th>
<th>CPU Elapsed Time in Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warehouse Plan</td>
<td>11.08</td>
</tr>
<tr>
<td>School Plan</td>
<td>13.15</td>
</tr>
<tr>
<td>Checking the Warehouse and School Plans</td>
<td>2.95</td>
</tr>
<tr>
<td>Integrating the Warehouse and School Plans</td>
<td>37.75</td>
</tr>
</tbody>
</table>

Table 4.2: Running Times for the Construction Example

4.3.1. Heuristics

Synapse uses a best-first search with respect to a set of user-specified priorities, which assign weights to the various resources used in the plan. Such priorities are a kind of plan

---

27 The tasks used in the construction example are described in Appendix A.
evaluation, or objective function, which are used to rank and compare different plans. If no priorities are given, the plans are ranked by their length. If the plans to be integrated have different sets of priorities (e.g. minimizing execution-time may be of high priority to one agent, while minimizing cost may be of high priority another), it may be desirable to average the priorities, or perform the integration twice, once with each set of priorities.

Synapse also uses the following heuristic to reduce the size of the search space (although in some cases, a solution may be missed): don't add a new node (in steps 5c or 6d of the planner) if the plan can be improved by adding a precedence constraint or rescaling (i.e. increasing the production of) an existing step.

Although not implemented in Synapse, a branch and bound approach may also be used to speed up the search for the optimal solution (with respect to the user-specified priorities). The priorities are an admissible heuristic (see Winston, 1984), since they are used to weight the resources actually produced and consumed by the plan (i.e. there is no component of this heuristic that "guesses" the distance remaining to the goal).

4.3.2. The Resource Manipulation Table

To facilitate the ranking of plans with respect to resource priorities, each plan has an associated resource manipulation table, the resource-based analog to Table of Multiple Effects, or TOME, introduced by NOAH (Sacerdoti, 1975). The resource manipulation table indicates:

1) the consumers/utilizers and producers of each resource r used in the plan

2) the weakest precondition of the plan with respect to r.
5. Related Work

5.1. Overview

Section 5.2 contrasts Synapse with previous systems that supported a capability for planning with resources. Section 5.3 does a similar comparison between Synapse and systems that supported some form of plan integration. Section 5.4 contrasts the Synapse problem reformulation mechanism with related systems.

5.2. Planning With Resources

One reason that traditional planners ignored resources is that resource issues were viewed as part of the scheduling problem, which was viewed as distinct from the planning problem. For example, ISIS (Fox, 1983), a job-shop scheduling system assumes that resources are allocated and tasks scheduled only after the task network has been planned. However, there are good reasons for including resource considerations during planning:

1) it is often natural to express some planning goals (e.g., the desired quantity of some product) in terms of resources

2) resource constraints may determine whether a given planning problem is solvable

3) resource considerations may be useful in comparing and evaluating alternate plans

4) there may be conflicts (e.g., a resource's aggregate demand exceeds its aggregate supply) and synergies (e.g., mergeable and redundant steps), resulting from resource interactions between plans
Few planners have a non-trivial planning with resources capability. SIPE (Wilkins, 1988) is the best-known planner example.\textsuperscript{28} SIPE provides a powerful constraint mechanism that can represent complex numerical constraints among resources. However, SIPE does not detect consumable resource deficits. SIPE only detects one type of resource conflict: when a reusable resource is utilized by possibly simultaneous steps. Reusable resources in SIPE are viewed as \textit{binary resources}: they may either be available or unavailable for use by some step. Reusable resource conflicts in SIPF are always handled by step reordering, whereas in Synapse, an additional conflict resolution strategy is to increase the supply of the resource.

Several other planning systems are able to resolve simple binary resource conflicts, similar to those resolved by SIPE (e.g., Corkill, 1979, Georgeff, 1983, Lee and Chung, 1989, Lee and Chung, 1989). Callisto (Sathi, Morton, and Roth, 1986), a project management system, resolves reusable resource conflicts (but ignored synergies or consumable resource interactions). von Martial (von Martial, 1990) also resolves reusable resource scheduling conflicts, using a framework based on Allen's (Allen, 1984) temporal logic.

5.3. Plan Integration

When hierarchical planners, such as NOAH (Sacerdoti, 1975), NONLIN (Tate, 1977), DEVISER (Vere, 1983, Vere 1985), SIPE (Wilkins, 1988), etc., expand a task into a set of subtasks, there is a need to "integrate" the subtasks with the already constructed plan. NOAH and NONLIN detected and resolved conflicts (essentially open preconditions and unsafe causal links); NOAH and NONLIN also detected and exploited synergies (although these capabilities were not described in detail). SIPE extended these planners to detect

\textsuperscript{28} SIPE is one of the few planners to be successfully applied to practical problems of a non-trivial size (Wilkins, 1989).
binary reusable resource conflicts, as described in Section 5.2, and to prevent redundant steps from being added to a plan.

Artificial intelligence researchers working in multi-agent planning generally assume that a centrally constructed plan is to be executed in a distributed fashion (Georgeff, 1987, Rosenschein, 1982). One system that integrated separately constructed plans was (Corkill, 1979), a distributed version of NOAH, which could detect and resolve the same types of conflicts that NOAH handled. And the partial global planning framework (Durfee and Lesser, 1987) allows planners communicating in real-time, in a dynamically changing environment, to exploit redundant steps and balance the processing loads of the collaborating planners.

Some recent work has been done on a portion of the plan integration problem (without considering resources). Machinist’s (Hayes, 1989) operator overlap, which occurs when two operators have common (propositional) preconditions, is a special case of my redundant step concept (described in Section 3.3.2). Yang and his colleagues (Yang, Nau, and Hendler, 1990, Nau, Yang, and Hendler, 1989, and Foulser, Li, and Yang, 1990) combine plans that solve individual goals, deleting any redundant steps, and merging any mergeable steps. (However, their notion of resource is limited to simple integer costs.) They also show that if certain restrictions on interactions between plans hold (e.g., all possible merges are independent), the integration problem can be solved in polynomial time.

None of the systems mentioned in this section provides a general mechanism for reasoning about interactions involving reusable and consumable resources, as the notion of resource interaction is restricted to binary resource conflicts. Synapse is the first planner to provide
a general and provably correct mechanism for detecting and integrating interactions involving both reusable and consumable resources.

5.4. Problem Reformulation

Problem reformulation in response to unsolvable planning problems is a novel contribution of Synapse. Although many planners re-order goals to avoid conflicting interactions and reduce search (e.g., Dawson and Siklossy, 1977, Tate, 1977, Vere, 1985, Hayes, 1986, Wilkins, 1988, Drummond and Currie, 1989, Irani and Cheng, 1989, Cheng and Irani, 1989), they all fail when a solution cannot be found. When the planning framework includes quantities of consumable, reusable, and producible resources, incremental changes in production goals or initial resource allocations may yield a solvable problem.

The problem reformulation algorithm given in Section 2.4.5.2 will fail if not all propositional goals can be achieved (i.e. there are propositional goal conflicts). If propositions are scored, then the lowest priority goal could be discarded until a solvable problem is obtained. (Luria, 1987) discusses ways to speed up the search for conflicting goals. ISIS (Fox, 1983), a job-shop scheduling system, provides an expressive constraint representation and constraint-relaxation techniques that are useful in resolving conflicting goals.
6. Conclusions and Future Research

6.1. Summary

This thesis details a domain-independent, theoretically rigorous planning framework that incorporates plan integration and planning with resources.

Chapter 1 motivated the plan integration problem and its importance in collaborative planning. It also introduced the collaborative construction planning scenario used throughout this dissertation. Plan integration will probably be useful in any domain in which multiple agents separately develop interacting plans (e.g., multi-project planning, process or production planning, construction planning).

Chapter 2 described the details of a planner that can reason about resources. Although most previous planners either ignored resources, or included only a limited capability for reasoning about resources, there are several advantages to including resource considerations when planning: 1) it is often natural to express some planning goals in terms of resources; 2) resource constraints may determine whether a given planning problem is solvable; 3) resource considerations may be useful in comparing and evaluating alternate plans; 4) resource interactions between plans allow potential synergies between plans to be exploited. Chapter 2 also described the details of the mechanism for reformulating unsolvable planning problems, a novel contribution of Synapse. When the planning framework includes quantities of consumable, reusable, and producible resources, incremental changes in initial resource allocations or production goals may yield a solvable problem.
Chapter 3 described the plan integrator, which is identical to the nonlinear planner, except for the restriction that only the nodes present in the original plans may be added during the planning process. In contrast to the nonlinear planner, the integrator is not guaranteed to find a minimum length solution.

Plan integration is appropriate if there is some nontrivial investment in the original plans, and thus the only desirable changes are those that exploit synergies and avoid conflicts between the plans (by deleting redundant nodes, reordering, and rescaling nodes). In contrast, planning "from scratch" is appropriate if the best plan for the combined problem is sought, even if that requires significant modifications (e.g., substitution of nodes) to the original plans.

Synapse is the first planner to provide a general mechanism for detecting and integrating conflicting and synergistic interactions involving both reusable and consumable resources. Past approaches had been restricted to a limited model of reusable resource conflict.

Chapter 4 discussed the Synapse user interface, system functionality, and implementation issues. Chapter 5 compared Synapse to previous planners. Most previous planners either ignored resources, or were only able to detect a limited type of resource conflict: when a binary reusable resource (i.e. a resource that is either available or unavailable) is utilized by possibly simultaneous steps.
Finally, Section 6.2 discusses several representational and algorithmic extensions to the Synapse planning framework that are likely to be useful in practice, the most important of which are:

1) an extended temporal reasoning capability, as time is likely to be a crucial factor in many real-world planning problems

2) a replanning capability, to adapt plans to an environment that may change during plan execution

Section 6.2 also briefly discusses applications of plan integration, including process planning and simultaneous engineering.
6.2. Future Extensions to Synapse

This section briefly discusses several possible extensions to Synapse that may be useful in practice, as well as possible future applications.

6.2.1. Planning Extensions

This section briefly discusses extensions involving the basic planning mechanism.

6.2.1.1. Temporal Reasoning

Various temporal reasoning and scheduling capabilities could be added to Synapse (e.g., Vere, 1985, Miller, Firby, and Dean, 1987). For example, Synapse currently assumes that a task needs to reserve each utilized resource for its entire duration. However, it may be the case that a task can relinquish its reservation before the task completes, or acquire its reservation after the task has begun. The task representation could be extended to allow the specification of a reservation interval, whose size could be related to the levels of other resources by consumption constraints.

6.2.1.2. Replanning

A practical planning system would need some sort of replanning capability to adapt a plan to a changing environment. For example, if the current state or goal is modified, some tasks may be redundant, since their effects are no longer required, while new tasks may be required to achieve new preconditions or goals. Synapse could be extended to incorporate techniques used in PRIAR (Kambhampati, 1989), which adapts an existing plan to a new
problem, as well as in SIPE (Wilkins, 1988), where a plan is adapted to changes in the execution environment.

6.2.1.3. Resource Capacities

Synapse implicitly assumes a capacity of 1: only one task may utilize a resource at a given instant. In reality, the capacity of the resource may allow simultaneous access by a number of tasks (e.g., a time-sharing computer system).

6.2.1.4. Disjunctive Goals and Preconditions

Disjunctive goals and preconditions may be a more natural way for users to model some domains. The task representation could be extended to allow this, and Synapse could ignore any open preconditions or unsafe causal links pertaining to a disjunctive precondition once its corresponding precondition has been satisfied (this is the approach taken by NOAH (Sacerdoti, 1975)).

6.2.1.5. Hierarchical Planning

Organizations can be said to plan in a hierarchy of abstraction spaces (Sacerdoti, 1974), where progressively finer levels of detail are introduced at progressively lower levels of the organizational hierarchy. A task at one level of abstraction may be viewed as a planning problem at a more detailed level of abstraction, which may be solved by a plan consisting of a set of subtasks. For example, a car production task may be decomposed into a plan consisting of subtasks to produce and assemble various car components (e.g., engine, chassis, transmission). Conversely, a planning problem may be collapsed into a task
whose prerequisites are specified in the initial state, and whose results are specified in the goal state.

A major challenge in such a scenario is to support *hierarchical integration*, by maintaining a consistent task and plan hierarchy in the presence of change. Tasks will inevitably be modified, causing the following *hierarchical conflicts*:

1) A plan containing a modified subtask no longer solves its planning problem

2) A subplan no longer solves a modified planning problem (due to a modified task)

Changing a single task may cause conflicts to propagate up and down the task/plan hierarchy. Similarly, the following *hierarchical synergies* are possible:

1) A modified task now has a redundant subtask.

2) A modified task now has a subtask that produces excess quantities of resources

It should be relatively straightforward to extend Synapse to detect and integrate such hierarchical conflicts and synergies. Kambhampati's plan reuse framework (Kambhampati, 1989), where an existing plan is adapted to a new problem, and SIPE's replanning capability (Wilkins, 1988), where a plan is adapted to changes in the execution environment, would also be relevant to hierarchical integration in the presence of changing tasks.
6.2.1.6. Resource Surpluses

Resource surpluses may occur during problem reformulation (e.g., if a node's consumption of some resource is reduced). Surpluses may be exploited by reducing the production of one or more suppliers, if doing so will not cause a deficit. Reducing production, in turn, may cause other surpluses, etc. Alternatively, the consumption of consumers of the resource can be increased to use up the surplus. This may be useful if the increase in consumption allows a production goal to also be increased. For example, it may be desirable to increase the consumption of steel if it allows us to build more cars.

6.2.2. Problem Reformulation Extensions

This section briefly discusses extensions involving the problem reformulation mechanism.

6.2.2.1. Handling Deadline Violations

Deadline violations should really be treated as another type of conflict. During the planning process, if a user-specified deadline is violated, the plan should be suspended. When reformulating, the two ways of addressing deadline violations are to either:

1) relax the deadline

2) decrease the duration of the plan (if the durations of one or more tasks on the critical path are affected by a consumption constraint)
6.2.2.2. Production Ceilings

One possible representational extension is to allow a goal to specify upper, as well as lower, bounds on production (e.g., of some toxic by-product). Unsolvable problems due to production goal conflicts may now arise if satisfying a lower bound on the production of one resource prevents the satisfaction of an upper bound on the production of some other resource, or vice versa. One of the conflicting production goals must be modified in order for the planning problem to have a solution, using problem reformulation techniques analogous to those described in Section 2.4.5.2. This is similar in spirit to constraint relaxation in ISIS (Fox, 1983), a job-shop scheduling system, where knowledge about conflicting constraints is used to decide how to modify constraints to eliminate the conflict.

6.2.2.3. Reformulating Both the Initial and Goal States

Currently, when Synapse encounters an unproducible resource deficit, it searches for reformulations in which either the initial resource allocation are increased, or the consumption of one or more steps is decreased, but not both. A more complete treatment would allow reformulations in which both the increase initial resource allocation strategy and the decrease consumption strategy are applied with respect to a single resource deficit. (However, it is non-trivial to determine the amount by which to reallocate or decrease consumption if both strategies are to be used.)
6.2.2.4. Propositional Reformulation

Unsolvable problems may also occur if every plan on the queue contains an unassertable open precondition \( p \), such that no step in the domain asserts \( p \), or an undeclobberable unsafe causal link, such that neither promoting or demoting the clobberer is possible.

The planner could be extended to perform propositional reformulations by suspending plans that contain either unassertable open preconditions or undeclobberable unsafe causal links, and searching for ways of eliminating the open precondition or unsafe link. For an unassertable open precondition \( p \) at node \( i \), such possibilities include:

1) Adding \( p \) to the initial state.

2) Delete node \( i \) (if \( i \) is the goal node, then remove \( p \) from the set of goal propositions).

For an undeclobberable unsafe causal link \( \langle i, p, j \rangle \) with clobberer \( k \), such possibilities include:

1) Delete node \( j \)

2) Delete node \( k \)

Note that deleting steps may cause other conflicting and synergistic interactions which need to be integrated (or may require reformulating the initial or goal states). For example, establishers of the deleted step now have an open precondition, steps formerly supplied by the deleted step may now incur resource deficits, and steps that supplied resources or established propositions required by the deleted step may now be redundant.
6.2.3. Performance Enhancements

6.2.3.1. Search Enhancement

An iterative-deepening-A* search control approach could be used instead of best-first-search. Korf (Korf, 1985) has shown that an iterative-deepening-A* search is significantly more space efficient (it uses linear space) than breadth-first-search (which uses exponential space).

6.2.3.2. Incremental Interaction Detection

Incremental updating of the set of conflicts and synergies will have the same worst-case computational complexity as the plan checking algorithm, but on average, the number of nodes examined will be smaller, since the analysis can be focused on the nodes that have been changed (added, deleted or reordered) and the nodes that depend on them. Similarly, incremental computation of plan scores (with respect to the priorities) and critical path method quantities (Nasrallah, 1989) in response to plan modifications, could also be done, as a minor performance enhancement.

6.2.4. Other Extensions

6.2.4.1. Synapse Agents

Object Lens agents (Lai, Malone, and Yu, 1988) that invoked the plan checker or integrator could be triggered by certain events. One possibility is that when a user obtains a new plan (e.g., upon receipt of an electronic mail message from a colleague containing a hypertext
link to a new plan) the plan checker could check the incoming plan with respect to a pre-specified existing plan. (For example, the mail message might describe the warehouse construction plan, and the recipient of the message might be the developer of the school construction plan.) Synapse agents could also be triggered by changes to a plan, or changes to a task contained by a plan.

6.2.4.2. Living Without a Common Language

Collaborators may use different languages (e.g., different names for resource and condition instances, and task types) if some mechanism for translating terms into their equivalents is available. Lee and Malone (Lee and Malone, 1988) discuss the issues involved in translating objects originating from different object hierarchies.

6.2.5. Future Applications

This section briefly discusses two possible application of plan integration: simultaneous engineering, and a generalization of the plan integration problem: the system integration problem.

6.2.5.1. Simultaneous Engineering

Simultaneous or concurrent engineering is the parallel development of product designs and the process plans to produce those products (Nevins and Whitney, 1989). Plan integration techniques could be useful in supporting simultaneous engineering since a design defines a planning problem to be solved by the plan that produces the designed object. For example, the goal of a car production plan is to produce and assemble the car's components, as specified in the design, given current resource levels. Changing a design would therefore
be equivalent to changing a goal, perhaps resulting in the need to modify the process plan by adding new tasks (to establish new features), or deleting obsolete tasks (that establish obsolete features). In addition, the problem reformulation algorithm could detect when sets of design features are collectively unproducible given available resources, and suggest how to reformulate the problem, either by redesign or by reallocation of resources.

Process planning for machined solids may be a promising domain for the application of plan integration techniques, since it is a well-understood domain, and the types of interactions are known (and thus can relatively easily formulated in the artificial intelligence planning framework). Systems exist to generate process plans for separate features (Shea, 1986), but not how to integrate these plans. Plan integration could be used to improve the performance of a set of related process plans (e.g., plans for producing various components of the products in a company's product line). Synergies might include sharing a common setup or tool configuration (thereby minimizing setup time or the number of tool changes).

6.2.5.2. System Integration

One can think of a system as a set of interacting components that performs some function. A plan is therefore a system: its components are tasks and its function is to achieve some goal. Other examples of systems are: hardware (mechanical, electrical, etc.), software, and even written documents (e.g., sections of a document interact and have functional purposes29). One intriguing research question is: how does the plan integration problem relate to the more general system integration problem?

29 See (Neuwirth, Kaufer, Chandhok, and Morris, 1990) for a discussion of the use of plans during the writing process.
If we model the function of a system component with a set of preconditions and effects, then we can transform the system integration problem into a plan integration problem. In the car example, the effects of the engine may be to consume some amount of fuel, and produce some amount of energy. A conflict would arise if the amount of energy produced by the engine is insufficient to meet the aggregate demand by other car components. A redundancy synergy would occur if two components had the same effect (e.g., to support some other component). In the case of written documents, the effect of a section in a manual may be to document a certain software feature. Redundancies could arise if several sections had the same effect, such as redundantly documenting the same feature, a frequent occurrence in technical support organizations.
Appendix A: Tasks in the Construction Domain

This section shows the type definitions of the construction tasks used in the warehouse and school example. Figure A.1 below illustrates the warehouse construction plan taken from (Barrie and Paulson, 1984).  

---

**FIGURE A.1: The Warehouse Construction Plan**

Note that the start-to-start restraint duration (or SS) values shown in Figure A.1 really represent information at the subtask level (e.g., which subtasks establish preconditions of which other subtasks). However, it's difficult for an artificial intelligence planner to make use of this information in the absence of any information about the decomposition of the task into subtasks.
First the generic construction tasks are described, followed by the tasks specific to warehouse and school construction, respectively. Each task type has a set of Prerequisites, Results, and Resource Usage Rates, as well as a Duration.

Generic Construction Tasks

*Lay Concrete*

Prerequisites

- site earthwork complete
- concrete setup complete
- 1 cement mixer
- K tons of concrete\(^{31}\)

Results

- lay concrete complete

Resource Usage Rates

- Money = Size \(*\) 2 + 15200
- Labor Hours = Size \(*\) .045 + 900
- Concrete = Size \(*\) .029

Duration: 157 days

*Building Finish*

Prerequisites

- precast walls complete

Results

- building finish complete

Resource Usage Rates

- Money = Size \(*\) 2.05 + 9300
- Carpenter Hours = Size \(*\) .05 + 970

Duration: 119 days

*Electricity*

\(^{31}\) The variable K is used to indicate that the exact quantity is to be determined by the appropriate Resource Usage Rate.
Prerequisites
structural steel complete

Results
electricity complete

Resource Usage Rates

Money = Size * 1.3 + 3000
Electrician Hours = Size * .02 + 470

Duration: 110 days

Fire Protection

Prerequisites
plumbing & hvac complete

Results
fire protection complete

Resource Usage Rates

Money = Size * 1.08 + 4600
Electrician Hours = Size * .018 + 520

Duration: 150 days

Plumbing & Hvac

Prerequisites
site earthwork complete

Results
plumbing & hvac complete

Resource Usage Rates

Money = Size * 1.63 + 2600
Plumber Hours = Size * .025 + 500

Duration: 170 days

Precast Walls

Prerequisites

structural steel complete
1 crane

Results

precast walls complete

Resource Usage Rates

\[ \text{Money} = \text{Size} \times 1.7 + 23500 \]
\[ \text{Iron Worker Hours} = \text{Size} \times 0.015 + 390 \]

Duration: 70 days

Roofing

Prerequisites

structural steel complete

Results

roofing complete

Resource Usage Rates

\[ \text{Money} = \text{Size} \times 0.7 + 17000 \]
\[ \text{Carpenter Hours} = \text{Size} \times 0.018 + 450 \]

Duration: 60 days

Concrete Setup

Prerequisites

1 cement mixer

Results

concrete setup complete
80 Labor Hours
$10000

Duration: 3 days

Site Earthwork

Results

site earthwork complete

Resource Usage Rates

\[ \text{Money} = \text{Size} \times 0.73 + 900 \]
\[ \text{Labor Hours} = \text{Size} \times 0.009 + 120 \]
Duration: 90 days

*Structural Steel*

Prerequisites

- lay concrete complete
- special projects begun
- 1 crane
- K tons of steel

Results

structural steel complete

Resource Usage Rates

\[
\text{Money} = \text{Size} \times 2.85 + 19900
\]

\[
\text{Iron Worker Hours} = \text{Size} \times 0.02 + 770
\]

\[
\text{Steel} = \text{Size} \times 0.00187
\]

Duration: 75 days

Tasks Specific to Warehouse Construction

*Start Special Floors*

Prerequisites

- site earthwork complete

Results

- special floors begun
- special projects begun

Resource Usage Rates

\[
\text{Money} = \text{Size} \times 0.88 + 2000
\]

\[
\text{Labor Hours} = \text{Size} \times 0.02 + 665
\]

Duration: 40 days

*Complete Special Floors*

Prerequisites

- special floors begun
- precast walls complete

Results

special floors complete
Resource Usage Rates

Money = Size * .88 + 2000
Labor Hours = Size * .02 + 665

Duration: 45 days

Tasks Specific to School Construction

Brickwork

Prerequisites

precast walls complete

Results

brickwork complete

Resource Usage Rates

Money = Size * .8 + 3000
Mason Hours = Size * .03 + 500

Duration: 80 days

Start Special Rooms

Prerequisites

site earthwork complete

Results

special rooms begun
special projects begun

Resource Usage Rates

Money = Size * .88 + 2000
Labor Hours = Size * .02 + 665

Duration: 60 days

Complete Special Rooms

Prerequisites

special rooms begun
precast walls complete

Results
special rooms complete

Resource Usage Rates

Money = Size \times 0.88 + 2000
Labor Hours = Size \times 0.02 + 665

Duration: 70 days
Appendix B: Correctness Proofs

This Appendix proves that the linear planner is sound and complete, and that the nonlinear planner is complete.\(^{32}\)

Linear Planning

This section proves that the linear planner is sound (it will never return an incorrect answer), and complete (if a solution exists, the planner will find it).

Soundness

I first restate the definition of weakest precondition (introduced in Section 2.3):

**Definition.** \(V\) is the *weakest precondition* for a plan \(h\) to achieve \(G\) if there is no other sufficient precondition for \(W\) for \(h\) to achieve \(G\), such that \(W\) is a *weaker precondition* than \(V\).

**Definition.** \(V\) is *weaker* than \(W\) if the following are true:

1) the propositions contained in \(V\) are a subset of the propositions contained in \(W\)

2) each resource requirement in \(V\) is less than or equal to the corresponding resource requirement in \(W\)

3) \(V \preceq W\)

---

\(^{32}\) Recall that the soundness lemma for nonlinear plans (discussed in Section 2.4.2) removes the need for a soundness invariant on the search process.
I restate the soundness invariant (introduced in Section 2.3.1), which captures the notions of soundness for linear resource-manipulating plans.

**Soundness Invariant.** If \(<G', h>\) is an element of PARTIAL-PLANS then \(G'\) is the weakest precondition for \(h\) to achieve \(G\).

**Soundness Theorem.** *The linear planner preserves the soundness invariant.*

**Proof.** By induction. The basis for the induction is established by the initial entry of PARTIAL-PLANS, \(<G, \text{the null plan}>\), which trivially satisfies the soundness invariant. Assume that the soundness invariant is satisfied for a given entry \(<G', h>\). This implies that \(G'\) is the weakest precondition for \(h\) to achieve \(G\). I now show that the algorithm always constructs an extension \(<G'', S;h>\) of the entry \(<G', h>\) that also satisfies the soundness invariant (i.e. \(G''\) is the weakest precondition for \(S;h\) to achieve \(G\)).

The algorithm obtains \(G''\) by modifying \(G'\) to account for:

1) the propositional preconditions and effects of the new step \(S\)

2) the resource preconditions and effects of the new step \(S\)

I analyze the propositional and resource\(^{33}\) components of \(G''\) in the two sections below.

---

\(^{33}\) To simplify the proofs, I will not explicitly consider reusable resources, and will assume that they are modeled as consumable resources that are both consumed and produced in equal amounts by any step that uses them (as mentioned in Section 2.2.2).
Propositions

Step 4ai forces the set of propositions in $G''$ to be those contained in $G'$, minus the propositions asserted by $S$, plus the propositional preconditions of $S$. Clearly $G''$ is a sufficient precondition for $S; h$ to achieve $G$, since:

1) $G''$ includes all of the propositional preconditions of $S$

2) $G''$ includes all of the propositions in $G'$ (except those asserted by $S$), and $G'$ is a sufficient precondition for $h$ to achieve $G$

I now show that there is no $G'''$, such that $G'''$ is a weaker precondition than $G''$ for $S; h$ to achieve $G$.

Assume that $G'''$ is a weaker precondition than $G''$ for $S; h$ to achieve $G$. Then there must be some proposition $p$ in $G''$ that is not in $G'''$ (i.e. $G'''$ must contain at least one fewer proposition than $G''$). There are two possibilities:

1) $p$ is a precondition of $S$

2) $p$ is contained in $G'$ (i.e. $p$ is a precondition of some step $T$ in $h$), such that $p$ is not asserted by $S$

Clearly $p$ cannot be a precondition of $S$, because then $G'''$ would not be sufficient for $S; h$ to achieve $G$ (since a precondition of $S$ would not be satisfied). Similarly, $p$ cannot be in $G'$ (if $p$ is not asserted by $S$) because again, $G'''$ would not be sufficient (since a precondition of some step $T$ in $h$ would not be satisfied).
Therefore there is no such G'', and G'' is the weakest precondition for $S;h$ to achieve $G$ (with respect to propositions).

Thus step 4ai guarantees the preservation of the soundness invariant (with respect to propositions).

Resources

Steps 4a(ii) and 4a(iii) force the resource requirements of $G''$ to be the resource requirements of $G'$, minus the amounts that $S$ produces, plus the amounts that $S$ consumes. Clearly $G''$ is a sufficient precondition for $S;h$ to achieve $G$, since:

1) for each resource $r$ consumed by $S$ in quantity $k$, the resource requirement for $r$ specified in $G''$ exceeds the corresponding resource requirement in $G'$ by $k$

2) $G''$ includes the resource requirements in $G'$ (reduced by the amounts produced by $S$), and $G'$ is a sufficient precondition for $h$ to achieve $G$

I now show that there is no $G'''$, such that $G'''$ is a weaker precondition than $G''$ for $S;h$ to achieve $G$.

Assume that $G'''$ is a weaker precondition than $G''$ for $S;h$ to achieve $G$. Then there must be some resource $r$ such that the amount specified in $G'''$, is less than the amount specified in $G''$. There are two possibilities:\footnote{If $r$ were not consumed by any step, then there would be no need for $r$ to appear in the weakest precondition.}
1) \( r \) is consumed by \( S \)

2) \( r \) is consumed by some step \( C \) in \( h \)

Let \( K \) be the resource requirement for \( r \) specified in \( G' \). Since \( G' \) is the weakest precondition for \( h \) to achieve \( G \), \( K \) must be the minimum amount of \( r \) required for \( h \) to achieve \( G \). If \( r \) is consumed by \( S \) in quantity \( k \), the requirement for \( r \) specified in \( G'' \) will be \( K + k \). If \( G'' \) specifies a requirement less than \( K + k \), then there will clearly be a deficit (since \( h \) requires \( K \) units of \( r \) and \( S \) requires \( k \) units of \( r \)), and \( G'' \) would not be sufficient. Therefore \( r \) cannot be a resource consumed by \( S \). Now assume that \( r \) is consumed by some step \( C \) in \( h \). Recall that the aggregate consumption of \( r \) by \( h \) (specified in \( G' \)) is \( K \). If \( r \) is produced by \( S \) by in quantity \( k \), the requirement for \( r \) specified in \( G'' \) will be \( K - k \). If \( G'' \) specifies a requirement less than \( K - k \), then there will clearly be a deficit (since \( h \) requires \( K \) units of \( r \) and \( S \) produces only \( k \) units of \( r \)), and \( G'' \) would not be sufficient. Therefore \( r \) cannot be a resource consumed by a step in \( h \).

Therefore there is no such \( G'' \), and \( G'' \) is the weakest precondition for \( S;h \) to achieve \( G \) (with respect to resources).

Thus steps 4aii and 4aiii guarantee the preservation of the soundness invariant (with respect to resources). And since step 4ai guarantees the preservation of the soundness invariant (with respect to propositions), \( G'' \) is the weakest precondition for \( S;h \) to achieve \( G \), preserving the soundness invariant.

**Linear Soundness Lemma.** *The linear planning with resources algorithm is sound (i.e. will never return an incorrect answer).*
**Proof.** Since the linear planning with resources algorithm preserves the soundness invariant (by the previous theorem), then for any entry \(<G', h>\) in PARTIAL-PLANS, \(G'\) is the weakest precondition for \(h\) to achieve \(G\). Since step 3 of the linear planner only returns a plan \(h\) if \(I\) (the initial state) satisfies the weakest precondition for \(h\) to achieve \(G\), the algorithm is sound./

**Completeness**

I first restate the completeness invariant (introduced in Section 2.3.1), which captures the notions of completeness for linear resource-manipulating plans.

**Completeness Invariant.** If \(f\) is a finite minimum length solution (i.e. no other solution has fewer steps than \(f\)) to the planning problem then there exists some entry \(<G', h>\) in PARTIAL-PLANS such that \(h\) is a suffix of \(f\) (e.g. if \(f\) were the linear plan \(a;b;c\), then \(h\) could be the null plan, the plan \(c\), the plan \(b;c\), or the plan \(a;b;c\)).

**Linear Completeness Theorem.** *The linear planner preserves the completeness invariant.*

**Proof.** By induction. The basis for the induction is established by the initial entry of PARTIAL-PLANS, \(\langle G, \text{the null plan}\rangle\), which trivially satisfies the completeness invariant (since the null plan is a suffix of all plans). Assume that the completeness invariant is satisfied for a given entry \(<G', h>\). This implies that there exists a minimum length solution \(f\), such that \(f = g;h\) (i.e. \(h\) is a suffix of \(f\)). We now need to show that if the algorithm removes \(<G', h>\) from PARTIAL-PLANS, it always adds back to PARTIAL-PLANS some extension \(<G'', S;h>\) of \(<G', h>\) that also satisfies the completeness invariant (i.e. \(S;h\) is a suffix of the minimal length solution \(f = g;h\)).
Step 4 constructs extensions $S;h$ of a plan $h$ for all steps $S$ such that:

1) $S$ does not delete any proposition in $G'$

2) $S$ either:

   a) produces a resource for which a deficit exists (i.e. a positive resource requirement in the weakest precondition $G'$)

   b) asserts a proposition in $G'$

I now show that one of these extensions $S;h$ is a suffix of the minimal solution $f = g;h$. Let $S$ be the last step in the sequence $g$, such that $g = g';S$ (if there is no such step $S$, then $g$ is the null plan, and $h$ is already the minimal length solution). If $S$ does not assert a proposition or produce a resource mentioned in $G'$, then $f$ is not a minimal length solution, violating the completeness invariant, since there can be a shorter solution that omits the step $S$. Therefore $S$ must either assert a proposition or produce a resource mentioned in $G'$, as required by step 4. Also, if $S$ deletes a proposition $p$ in $G'$, then the weakest precondition for $h$ to achieve $G$ will be violated, and it will be impossible to extend the plan $S;h$ into a solution. Therefore $S$ must not delete a proposition $p$ in $G'$, also required by step 4. Since step 4 will try all possible steps $S$ fulfilling these restrictions, one of these steps must be the step $S$ in the minimal solution. Thus, some plan $S;h$ will be a suffix of the minimal length solution, and step 4 preserves the completeness invariant./

**Linear Completeness Lemma.** The linear planner is complete (i.e. it will find a solution, provided one exists).
Proof. By the previous theorem, the linear planner always preserves the completeness invariant (that there is a plan h in PARTIAL-PLANS that is a suffix of a minimal solution f). Since there are only a finite number of steps to add at any given point in the search, the planner must eventually find some solution plan f such that f solves the problem <I, G, (S)> (provided such a finite minimal length solution exists), and the linear planning algorithm is complete.
Nonlinear Planning

This section proves that the nonlinear planner is complete (if a solution exists, the planner will find it).

Completeness

Before proving the completeness of the algorithm, I restate the completeness invariant (introduced in Section 2.4.2), which captures the notion of completeness for nonlinear resource-manipulating plans.

Completeness Invariant. If \( f \) is a minimal finite sequence of steps that solves the planning problem, then there exists some nonlinear plan \( P \) in \textsc{Partial-Plans} such that \( f \) is a \textit{linear interpretation} of \( P \).

Definition. A sequence of operations \( f = a_1; \ldots; a_n \) is a \textit{linear interpretation} of a nonlinear plan \( P \) if there are at least as many steps in \( f \) as nodes in \( P \) and there exists an assignment of integers to the nodes of \( P \) such that:

1) node 0 has number zero, node \( G \) has number \( n + 1 \) (where \( n \) is the length of the sequence \( f \)), and every other node is assigned a number between 1 and \( n \) inclusive.

2) the step \( a_i \) in the sequence \( f \) is associated with node \( i \) in \( P \).

3) the partial order of \( P \) is respected (and no two nodes have the same number). Thus, if \( P \) contains the precedence constraint \( i < j \), then \( a_i \) must precede \( a_j \) in \( f \).

4) every causal link \( <i, p, j> \) in \( P \) has a corresponding causal link \( <a_i, p, a_j> \) in \( f \). That is, if \( i \) is an establisher of \( j \), then \( a_i \) is an establisher of \( a_j \).

5) if \( i \) produces some amount \( K \) of a resource \( r \), then \( a_i \) must produce \( K' \) such that \( K' \geq K \). That is, the resource quantities produced by nodes in \( P \) are lower bounds on the resource quantities manipulated by steps in \( f \). (The resource quantities produced by a node will increase if it is rescaled.)
Nonlinear Completeness Theorem. The nonlinear planning with resources algorithm preserves the completeness invariant.

Proof. We need to show that after each iteration of the algorithm, there is a plan $P$ in PARTIAL-PLANS such that some minimal solution $f$ is a linear interpretation of $P$. I prove this by induction. The basis is established when PARTIAL-PLANS contains only the initial plan, $P_0$, which consists of nodes $I$ and $G$. The completeness invariant is satisfied, since any solution $f$ must be a linear interpretation of this skeletal plan. Assume that PARTIAL-PLANS satisfies the completeness invariant at some point in the search. Now assume that the problem is solvable, and there exists a minimal length solution $f = a_1; \ldots; a_n$, such that $f$ is a linear interpretation of some nonlinear plan $P$ in PARTIAL-PLANS. We now need to show that if the nonlinear planning algorithm removes $P$ from PARTIAL-PLANS, such that a solution $f$ is a linear interpretation of $P$, it always adds back to PARTIAL-PLANS some plan $P'$, an extension of $P$, such that $f$ is also a linear interpretation of $P'$.

If $P$ is not a solution, at least one of the following types of conflicts must be present:

1) $P$ has one or more unsafe causal links, which are made safe in step 4
2) $P$ has one or more open preconditions, which are satisfied in step 5
3) $P$ has one or more resource deficits, which are reduced in step 6

The next 3 sections show that no matter which type of conflict the planner handles, the completeness invariant will be satisfied.
Case 1: Unsafe Causal Links

Step 4 selects an unsafe causal link \(<i, p, j>\) and clobberer C in P, and:

1) if C might precede i, it constructs an extension \(P'\) containing the additional precedence constraint \(C < i\) (step 4b)

2) if C might follow j, it constructs an extension \(P''\) containing the additional precedence constraint \(j < C\) (step 4c)

where nodes i, j, and C in P correspond to the steps \(a_i, a_j,\) and \(a_C\) in f.

Since f is a linear interpretation of P, it must contain a causal link \(<a_i, p, a_j>\) corresponding to causal link \(<i, p, j>\) in P. Since the semantics of causal links in linear plans requires that there cannot be a step \(a_k\) in f, between \(a_i\) and \(a_j\), such that \(a_k\) negates p, one of the following must be true:

1) \(a_C\) precedes \(a_i\), and f must be a linear interpretation of \(P'\)

2) \(a_j\) precedes \(a_C\), and f must be a linear interpretation of \(P''\)

Thus step 4 guarantees that f is a linear interpretation of an extension of P.
Case 2: Open Preconditions

Step 5 selects an open precondition \(<p, j>\) in P, and constructs all possible extensions of P containing the causal link \(<i, p, j>\) such that:

1) i is some node already in P that asserts p (step 5b)
2) i is a new node corresponding to a step \(a_i\) that asserts p (step 5c)

Let \(h\) be the suffix of \(f\) beginning with \(a_j\), the step in \(f\) corresponding to the node j in P.
Since \(f\) is a solution, the weakest precondition for \(h\) to achieve G must be satisfied.
Therefore there must be some step \(a_i\) before \(a_j\) that asserts p, and one of the following must be true:

1) \(a_i\) corresponds to a node i already in P, and \(f\) must be a linear interpretation of some plan constructed in step 5b

2) \(a_i\) does not correspond to a node already in P, and \(f\) must be a linear interpretation of some plan constructed in step 5c

Thus step 5 guarantees that \(f\) is a linear interpretation of an extension of P.
Case 3: Resource Deficits

Step 6 selects a deficit at node $i$ with respect to resource $r$, such that $\text{GRA}(i, r) < \text{PRECONDITION}(i, r)$, and constructs all possible extensions of $P$:

1) containing the new precedence constraint $i < C$ for each competitor $C$ of $i$ in $P$ (step 6b)

2) containing the new precedence constraint $S < i$, where $S$ is either:
   a) some node already in $P$ that produces $r$ (step 6c)
   b) a new node corresponding to a step as that produces $r$ (step 6d)

3) containing an existing supplier $S$ of $i$ whose production of $r$ has been increased (step 6e, described in Section 2.4.5.1.1)

Let $h$ be the suffix of $f$ beginning with $a_i$, the step in $f$ corresponding to the node $i$ in $P$. Since $f$ is a solution, the weakest precondition for $h$ (or any suffix of $f$) to achieve $G$ must be satisfied. Therefore, $\text{GRA}(a_i, r) \geq \text{PRECONDITION}(a_i, r)$, which implies that $\text{GRA}(i, r) < \text{GRA}(a_i, r)$, which implies that at least one of the following must be true:
1) There is some competitor $aC$ in f corresponding to a node C in P, such that $aC$ follows $a_i$, but C might precede i in P (i.e. a competitor has been removed from competition in f, but not in P). In this case, f will be a linear interpretation of some extension constructed by step 6b that adds $i < C$.

2) There is some producer $aS$ in f, corresponding to a node S in P, such that $aS$ precedes $a_i$ in f but S might follow i in P (i.e. a supplier exists in f, but is not guaranteed in P). In this case, f will be a linear interpretation of some extension constructed by step 6c that adds $S < i$.

3) There is some producer $aS$ in f that does not correspond to a node in P, such that $aS$ precedes $a_i$. In this case, f will be a linear interpretation of the extension constructed by step 6d that adds the new node S (which corresponds to $aS$) and $S < i$.

4) There is some supplier $aS$ in f, corresponding to a node S in P, such that the amount of $r$ produced by $aS$ is greater than the amount of $r$ produced by S. In this case, f will be a linear interpretation of the extension constructed by step 6e that increases the production of S by the minimum amount allowed by the production constraints of S.

Thus step 6 guarantees that f is a linear interpretation of an extension of P.

Therefore steps 4, 5, and 6 of the planning with resources algorithm always construct some plan $P'$, an extension of P, such that f is also a linear interpretation of $P'$. 
Thus the nonlinear planning algorithm preserves the completeness invariant.

**Nonlinear Completeness Lemma.** *The nonlinear planning with resources algorithm is complete (i.e. it will find a solution, provided one exists).*

**Proof.** By the previous theorem, there is always some plan $P$ in PARTIAL-PLANS such that a minimal solution $f$ is a linear interpretation of $P$ (if a solution exists). The number of possible plan modifications at any given point in the search is always finite, since there are only a finite number of nodes to add, precedence constraints to add, or nodes to enlarge. Furthermore:

1) the maximum number of nodes in $P$ is bounded by the number of steps in $f$

2) the number of precedence constraints in $P$ is also finite ($O(N^2)$ in the worst case, where $N$ is the number of nodes in $P$)

3) the number of rescale operations is bounded by the resource quantities (which are constrained to be integral) produced in $f$

Therefore, the planner must eventually find a solution plan $P_f$ such that $P_f$ solves the problem $<I, G, \{S\}>$ (provided such a solution exists), and the nonlinear planning algorithm is complete.
Bibliography


McAllester, D.A., Systematic nonlinear planning (submitted to *AAAI-91*).


