**Distributed Coordination of Autonomous Agents by Communicating on a Need-To-Know Basis**

by

Judy Y. Chen

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

October 31, 2003

Copyright 2003 by Judy Y. Chen. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and distribute publicly paper and electronic copies of this thesis and to grant others the right to do so.

Author_____
Department of Electrical Engineering and Computer Science
October 31, 2003

Certified by_____
Brian C. Williams
Associate Professor
Thesis Supervisor

Accepted by_____
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Distributed Coordination of Autonomous Agents by Communicating on a Need-To-Know Basis

by

Judy Y. Chen

Submitted to the
Department of Electrical Engineering and Computer Science

September 2003

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

# ABSTRACT

Intelligent autonomous agents working cooperatively accomplish tasks more efficiently than single agents, as well as tasks that were infeasible for the single agent. For example, a single agent transporting a ton of blocks will need to take multiple trips, possibly even more trips than it can make, and will take longer than several agents transporting the blocks in parallel. Unexpected events can result in the need for agents to change their plans in order to adapt. During replanning, the expense of communication can hinder the amount of information passed. In this thesis, agents reduce the communication load while adapting to environmental events by examining what changes will be made to teammate plans and by passing information only on a *need-to-know basis*.

More specifically, in this thesis, we describe a method whereby cooperating agents use identical planners and knowledge of the other agents' capabilities in order to pass information about the environmental conditions they observe that are necessary for their teammates to infer the correct actions to take. The agent must also pass conditions only to the teammates who are affected by the changes. Given that not all agents will have the same information about environmental conditions, the plans inferred by each agent may be different from their teammates. The Pertinent Information Planning (PIP) algorithm provided seeks to allow each agent to have an incorrect plan with respect to other agents but gives each agent a correct plan with respect to the actions that they perform. In order to determine the conditions that agents communicate, agents must repair their local plan and figure out which sets of actions have changed in the repaired plan. This thesis provides the procedures for repairing the local plans and for determining the minimal set of information to be communicated between agents and the minimal set of agents who need to know this information. This thesis also explains how the agents who need to know update their local plans. Finally, the thesis presents the results of a computer simulation for three test cases of a pedagogical cooperative space exploration example.

Thesis Supervisor: Brian C. Williams
Title: Associate Professor

## Acknowledgments

This thesis could not have been completed without the help of many people. First and foremost, I would like to thank Professor Brian Williams for supervising and for giving technical guidance throughout the entire process. The members of the Model-based Robotic and Embedded Systems Group at MIT, in particular, Jonathan Kennell, John Stedl, Raj Krishnan, I-Hsiang Shu, Tazeen Mahtab, and Margaret Yoon have also provided invaluable help and support. Finally, I would like to thank my friends and family for their love and support throughout this process.

This thesis also could not have been completed without the sponsorship of the Air Force Research Laboratory under contract F33615-01-C-1850.

## Table of Contents

## List of Figures

# Chapter 1

# Introduction

## 1.1 Motivation

Autonomous intelligent agents are invaluable in situations where sending humans is expensive. For example, in the case of space exploration, the cost of sustaining a mission over several years is exorbitant due to the size of the payload involved. Furthermore, many unexplored environments are hostile and potentially fatal to humans, or other carbon-based life forms, with a very high probability of death. Incurring such human risk and cost is unwarranted while less expensive alternatives, more specifically, robotic agents, could be used. Robotic agents can vary greatly in terms of their independence, from robots that are manually controlled by an operator, to fully autonomous robots. Robotic agents that are manually controlled act purely based on the will of their human operator. However, vast distances that separate the human operator from its robotic agent can add unwanted communication delays, while a hostile environment can create a complete impasse to communication between the operator and the agent. In these cases, an agent must have the ability to develop a plan of action and be able to carry out this plan with little to no human intervention. In other words, the agent must have complete autonomy.

In many domains, it is not always feasible for an agent to work alone. By working jointly and in parallel, using specialized skills, diverse sets of resources, and localized knowledge, agents can achieve goals that would be either inefficient or impossible for a single agent to accomplish. A cooperating team of agents can be organized in several ways. On one extreme, a centralized approach might consist of having a single leader coordinate all of the activities amongst the group. One major disadvantage of this approach is that the leader would then become a weak link. When the leader fails, the other agents would not know what to do, causing the mission to fail. An alternative and more robust approach is to distribute coordination of activities amongst the different agents.

In order to coordinate their activities, it is necessary for agents to exchange information with each other. It is impractical to assume that communication channels are unlimited or that they have a very large bandwidth at all times. Multiple agents using the same channel will quickly clog up the communication lines. Even just a few agents sending large packets of data can cause congestion. Costs associated with communication can also place heavy restrictions on the way in which information is exchanged. For instance, long delays due to large distances may not be desirable and are, consequently, associated with larger costs. Another restriction is the times during which channels of communication are opened. For example, communication between Earth and Mars is blocked daily by the Earth and Mars shadows. As another example, line of sight communication on Mars is blocked by rides and valley.

In all the above cases, where communication is costly or intermittent, total communication costs can be reduced by decreasing the throughput of messages passed between agents. Congestion on communication channels can be minimized by choosing the minimum, relevant information to be passed along. With a distributed system, it is costly to communicate complete plans, observations, and states as they change the plans. A plan consists of a series of actions where each action is enabled by some preceding action. For instance, the action “turn on a computer” must precede the action “type a thesis” since typing a thesis requires that the computer be in the on state. Because a change in an activity will affect other activities, states, and in a cooperative plan, other agents, the communication costs can quickly increase with a small change that affects several agents. It would be more efficient to find a way to communicate only observed changes in the environment and to only communicate those observations that are relevant. This thesis addresses the task of having agents agree on shared plans by passing only relevant observed changes.

The key idea underlying this thesis is to reduce communication by passing observations about environmental changes on a *need-to-know basis*. Agents observe changes to their environment, and then use their knowledge of the team’s plans to identify those agents that are influenced by the changes. These changes are then communicated to the influenced agents, who use these changes to generate new plans. The net effect is to communicate the minimum number of observables needed to ensure that the agents agree upon those parts of their shared plan that influence each other.

## 1.2 A Tale of Two Rovers

### 1.2.1 Scenario

Consider the following, somewhat futuristic scenario in which two rovers are transported to an unexplored planet onboard a space probe and arrive at a planet that they must explore. The two rovers are deployed to the surface, while the space probe remains orbiting, providing a communication relay and reconnaissance. Both rovers are equipped with a narrow-band communication device and an arm to gather samples. Rover 1 is further equipped with an onboard specimen analyzer and carries a deployable wide-band communication device that needs to be set up on the planet. Because of the delicate specimen analyzer on board and the deployable communication device, Rover 1 is the larger of the two rovers and has greater difficulty traversing rough terrain. Rover 2, the smaller rover, is equipped with a rock chiseler, a camera, and is able to more easily traverse rough terrain.

Prior to the commencement of the mission, the two rovers are given a plan that is pre-generated by the orbiter. To generate the plan, the orbiter takes and analyzes visual photographs of the planet prior to the beginning of the rover mission and determines three locations from which the rovers should retrieve and analyze samples. Based on image-analysis of the terrain, the three locations have been assigned to the two rovers to explore. Two locations have smooth terrain, and one location has rough terrain. The two rovers

must set up a wide-band communication device on the planet and send data back to the orbiter where the data will be stored and then transmitted back to Earth. Now, consider an example plan in which each of the rovers travels to its designated location(s) and picks up samples. Rover 1 must set up the wide-band communication device. The two rovers must then traverse back to a common location where all of the specimens are given to Rover 1, which analyzes all of them and then sends the analysis data back up to the orbiter.

Upon arrival on the planet surface, Rover 1 discovers that one of the locations it was assigned to explore is rough terrain where it was previously categorized as smooth. Based on the approach developed in this thesis, Rover 1 generates a new plan that takes into account this new observation. This new plan specifies that Rover 2 should explore the location instead. To communicate this new plan succinctly, Rover 1 simply tells Rover 2 about the rough terrain at this location. Rover 2 then uses its own planning ability to arrive at the same conclusion. Now both rovers are aware of and agree upon a new common plan without having to communicate the plan explicitly.

Upon arrival on the planet surface, Rover 2 observes that a location contains no loose specimens. Rover 2 generates a new plan in which it uses its rock chiseler to retrieve a specimen. Rover 2 analyzes this new plan and determines that this observation does not influence the actions of either Rover 1 or the orbiter. Hence Rover 2 does not need to communicate its observation.

### 1.2.3 Scenario characteristics

The scenario described above offers several intuitions as to how the algorithm will be presented. The scenario exemplifies the types of missions the algorithm is expected to perform. In this scenario, tasks like specimen analysis involve multiple rovers. Since Rover 1 has the specimen analyzer and Rover 2 has the sample, the two rovers must synchronize their activities by meeting at a specific place at a specific time, in order to transfer the specimen for analysis. Other tasks need not include both rovers and hence can be performed independently. Rover 1 need not be aware that Rover 2 has changed its plan to include rock chiseling, that is, its view of the complete mission does not need to be updated. Rover 2's view of the complete mission has changed to include rock chiseling, hence the two rovers and the orbiter maintain different world views. However, despite the differences in the rovers' two plans, the rovers are able to perform the mission as required.

## 1.3 Problem Statement

In distributed cooperative planning, there is a general tradeoff between communication and computation. The more information an agent is given, the less reasoning that needs to be done. For instance, a rover that is given only the task - pick up a rock 10 meters away - needs to be able to determine a finite number of actions to accomplish this goal. In contrast, a rover that is told 1) turn towards the rock, 2) travel 10

meters forward, and 3) pick up the rock, has been given more pieces of information than the previous rover; however, this rover makes fewer computations in order to perform the task. Each action moves the rover towards a new state in which some facts are true that were not true before. For example, the second action moves the rover to the state in which the rover is at the location of the rock. A fact may either be communicated explicitly or it may be inferred from other facts that have been communicated and facts that are already known. Hence, the number of facts communicated may be reduced by increasing the amount of computation.

This project seeks to determine a system where autonomous agents working cooperatively can reduce communication by inferring changes to shared plans and personal plans from observed facts that have been passed by other agents. Furthermore, this thesis seeks to minimize the information communicated to only that which affects the personal plan of the agent receiving the communication.

This thesis presents the Pertinent Information Distributed Planning (PIP) algorithm. The algorithm has several distributed elements in the planning process: distribution of control amongst the agents and distribution of activities during execution. For distribution of control, each agent has the ability to observe changes in the system and make plans that influence other agents. For distribution of activities, agents allocate tasks to other agents in the team. After analysis of the new plans, the planning agents then pass information in the form of conditions that are pertinent to the actions other agents must perform.

## 1.4 PIP Algorithm Overview

The PIP algorithm developed operates on plans generated from a set of initial conditions, a set of goals, and a set of plan operators represented by an extended STRIPS language called PDDL [13]. When an initial condition changes in the environment, the PIP algorithm repairs the plan, examines the plan for changed activities to determine which agent needs to know, and communicates the changed condition to those agents who also need to update their plans.

### 1.4.1 Initializing Team Plans

The algorithm begins with a complete plan generated by a planner for a team of cooperative agents to accomplish a set of goals from a set of initial conditions using a set of plan operators that the agents can execute. As previously stated, the plan operators are represented by an extended STRIPS planning approach using the first generation PDDL language. The PIP algorithm is limited by the PDDL language, more specifically, actions are limited to non-conditional atomic actions. The complete list of plan operators for this thesis is shown in Appendix A. For the rover example, the initial conditions and goals are shown in Figure 1.1

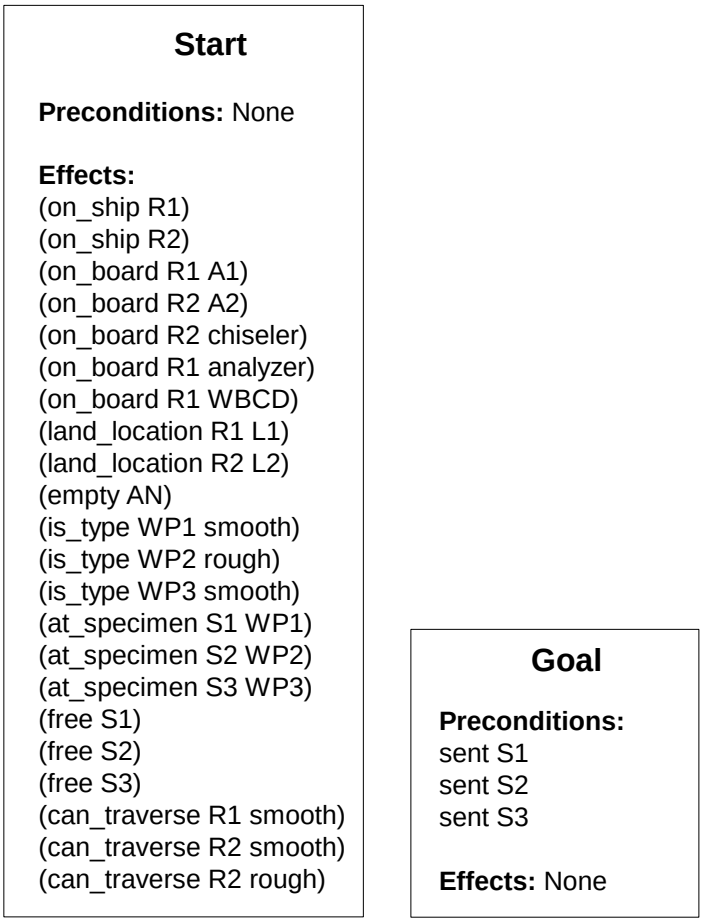


Figure 1.1. The initial conditions and goals encoded into PDDL operators as the start and goal operators.

Initially, all agents have the same complete plans since all agents are given the same problem description. In particular, we assume that all agents are given the same initial conditions, goals, and plan operators, and hence, generate the same plan. Figure 1.2 is the graphical depiction of the plan for the rover example in Section 1.2. We use a variation of the causal link plan representation [16]. This type of plan description can be extracted from a plan generated by any state of the art PDDL atomic action planner or any PDDL+ temporal planner. In this plan R1 and R2 are the two rovers respectively, L1 and L2 are the two landing locations, and WP1, WP2, and WP3 are the three locations

that the specimens are to be extracted from. S1, S2, and S3 are the three specimens, and AN is the specimen analyzer while WBCD is the wide band communication device.

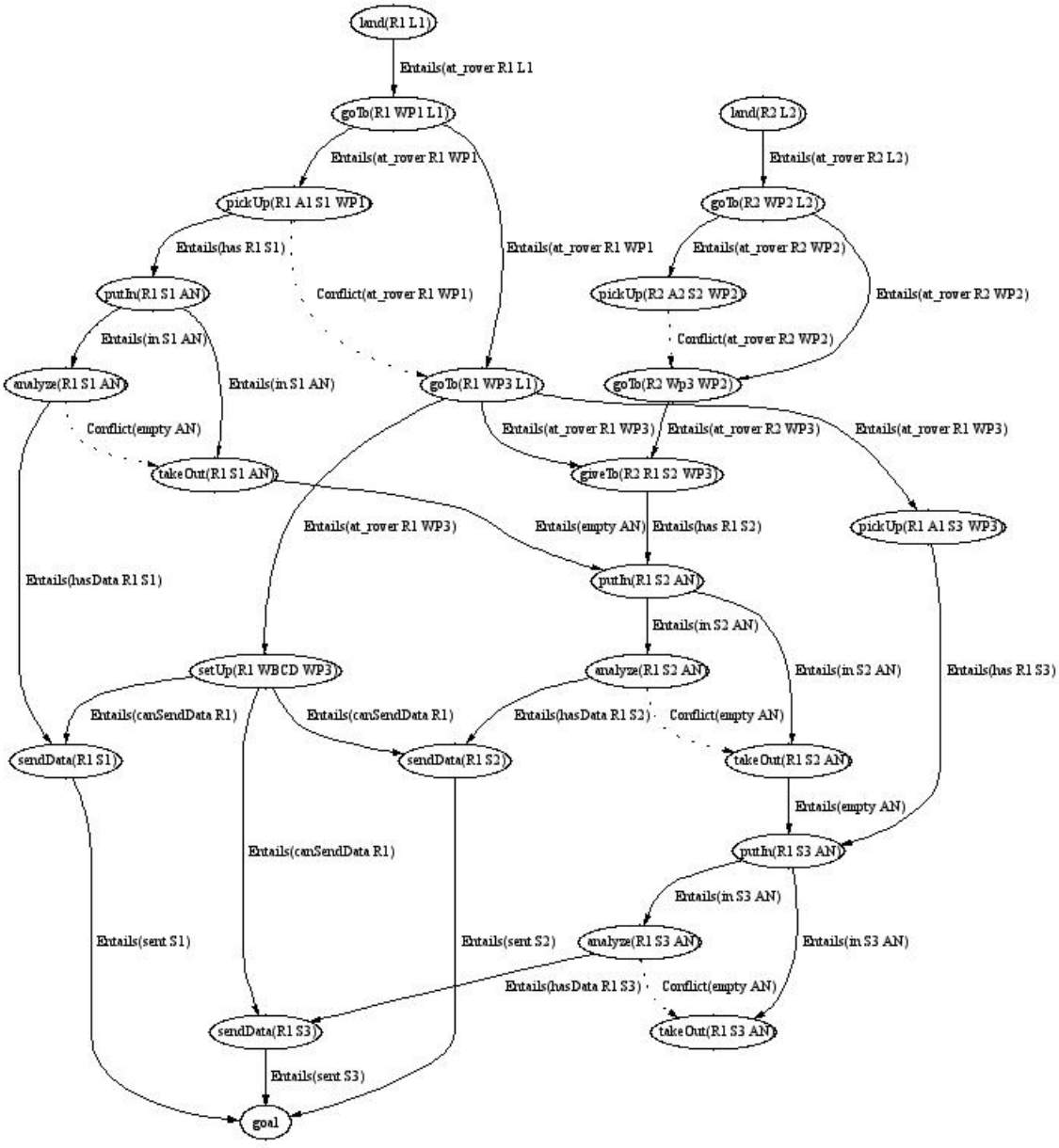


Figure 1.2. The nodes in the graph represent activities in the plan. The activities are in the form activityname(parameters). The arcs start from the activity that must precede another activity due to causal links or conflict resolutions. The proposition in the causal link produced by the starting activity and consumed by the ending activity is written on the arc. The proposition that requires conflict resolution is written on the conflict arcs.

### 1.4.2 Monitoring Changes to Initial Conditions

Due to changing environments that affect the initial conditions, the plan is not guaranteed to remain correct throughout its execution. Therefore, the agents must monitor the environment for any changes that will affect the plan.

In this thesis, we assume that all agents are in charge of monitoring all changes to the initial conditions that they observe, while specific agents are in charge of evaluating the effects of specific changes to these conditions. This approach is more flexible than the alternative approach of having each condition in the initial conditions being monitored and evaluated by a single assigned agent. The approach allows for changes in the plans that may affect an agent's ability to monitor a condition.

For each fact in the set of initial conditions, an agent is in charge of determining the changes to the plan for that condition, and the agent is said to own the condition. All the agents in the team keep a database called the relational knowledge base, which keeps track of the assignment of conditions to agents called owner agents. Once an agent discovers a change in the environment, it looks up the owner agent in the relational knowledge base and sends the change to the owner agent if the owner agent is not itself.

### 1.4.3 Plan Repair

Once the owner agent receives the changed conditions, it needs to determine what other agents also need to know about the changes. Changes to initial conditions invalidate activities supported by the initial conditions. Goals that were achieved by those activities are now unreachable. In order to reach these goals, new activities must be performed by the agents, and, in a need-to-know basis approach, only those agents performing the activities need to know that they are performing new activities. The owner agent needs to determine first which new activities must be performed before it can determine which agents need to know about the new activities. In order to find out what new activities need to be performed, the owner agent needs to generate a new plan that is consistent with current world conditions.

The first step in generating a new plan that is consistent with the current world conditions is to find the activities invalidated by the changed conditions. When an owner agent receives a set of changes, either from its own observations of the environment or from another agent, it finds a broken subplan that is comprised of activities that are no longer supported by following the causal links in its plan. In Figure 1.3, the shaded nodes are the activities that are no longer supported if Rover 1 finds out that the terrain at waypoint 1 is rough rather than smooth.

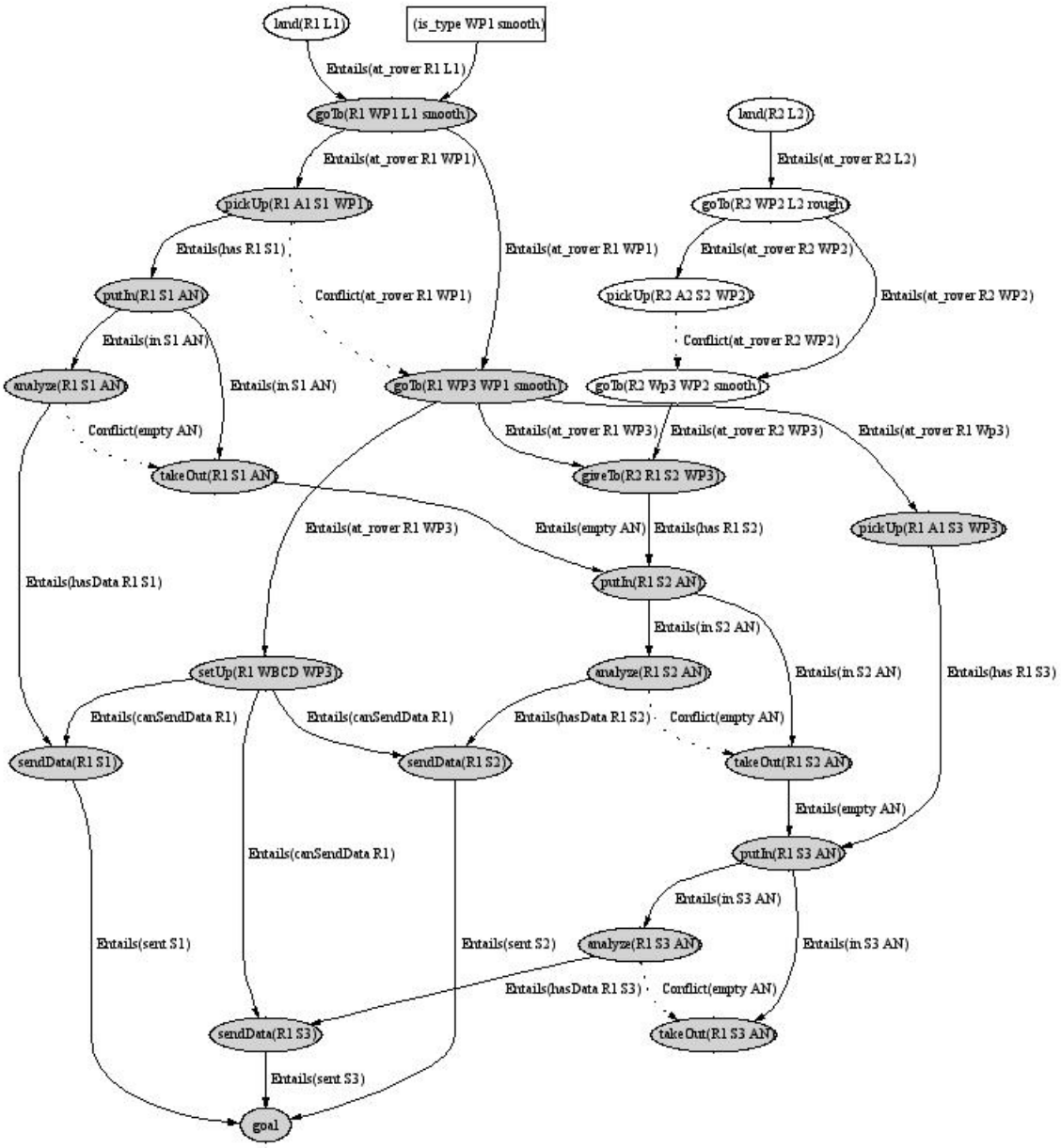


Figure 1.3. The changed initial condition “Waypoint 1 is not smooth” affects the activity “goTo(R1 WP1 L1 smooth)” since it is a precondition of that activity. The broken subplan, represented by the shaded nodes, are found by following the causal links from that activity.

Recall from the example where Rover 2 finds no loose specimen at Waypoint 2, that two agents can still agree on shared plans despite having two different complete plans. Because agents can have different plans with respect to actions that they perform,

an owner agent should only change the part of the plan that it is guaranteed to have the correct activities and casual links. Since observations are only communicated on a need-to-know basis, the owner agent is only guaranteed to have the correct activities for the activities that it supports and activities that it performs. In this thesis, only the activities that are invalidated by the changed condition are allowed to change whereas the rest of the plan is held constant. This restriction will limit the planner's ability to generate a complete plan, but it also prevents agents from modifying outdated parts of the plan.

For our approach, we make the simplification that the broken subplan is the only part of the plan that was guaranteed to be correct before the changed condition. The unshaded nodes, those that are not invalidated by the changed condition, and thus, not part of the broken subplan, must be guaranteed to be included in any plan generated by the planner. These unshaded activities must be coded in the formulation of the replanning problem in such a way as to ensure that they are included in the planner output. In the second step of plan repair, the PIP algorithm translates the unshaded nodes into PDDL operators called *strongly linked activities*.

Figure 1.4 shows the PDDL operator for the strongly linked activity representing the node "land(R2 L2)." Three dummy conditions are added for planning purposes, but do not represent any state of the world: Flag 1, Flag 2, and GoalLandR2L2. Conditions representing goals are added to ensure that the activity is included in the plan; GoalLandR2L2 is added to the goal operator. Conditions called flags are added to ensure that causal links are maintained; Flag1 is added to the start operator, the activity that

precedes Land(R2 L2), and Flag 2 is added to the goTo(R2 WP2 L2 rough) operator, the activity that follows Land(R2 L2).

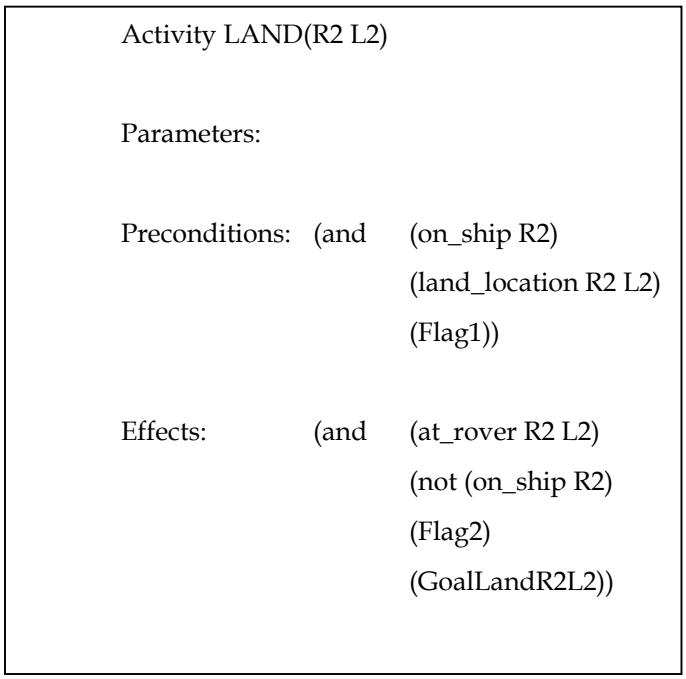


Figure 1.4. The strongly linked activity for Land(R2 L2) shows the addition of 3 dummy conditions: Flag 1, Flag 2, and GoalLandR2L2. These dummy conditions are needed to ensure that the output plan includes the activity.

The new problem formulation is given to the planner to generate a new plan. In the new plan of the rover example, Rover 2 traverses to Waypoint 1, picks up Specimen 1, and gives it to Rover 1 at Waypoint 3. Figure 1.5 shows a graphical representation of the new plan.

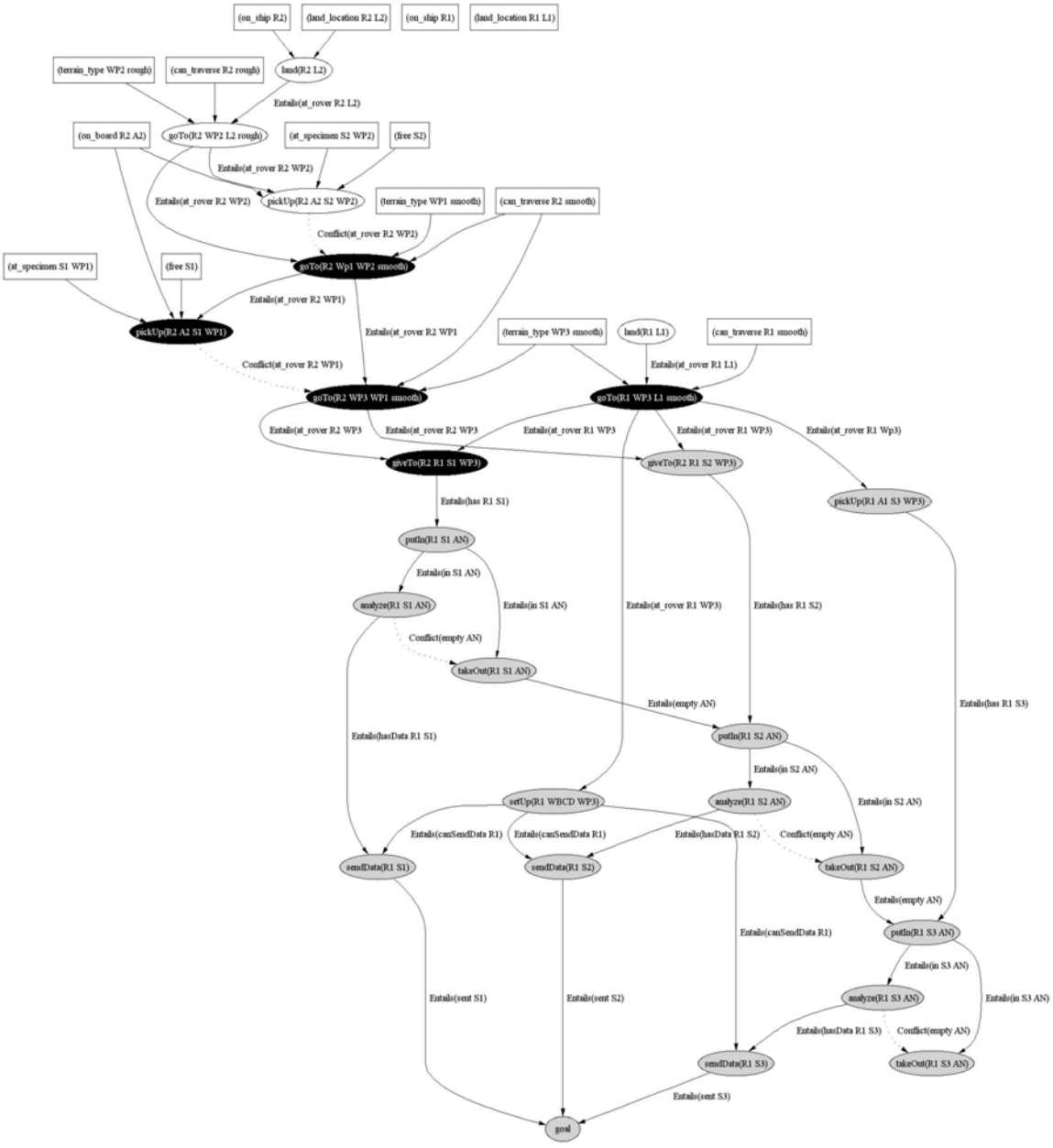


Figure 1.5. The new plan for the rover example. The gray nodes are new activities that are the same in both the new plan and the old plan. The black nodes are new activities that are not in the old plan but are in the new plan.

### 1.4.4 Plan Verification

Because the plan has changed, new dependencies upon initial conditions that the owner agent has not been tracking have been introduced. The owner agent needs to verify that these dependencies are the most current values and have not changed. The new dependencies come from the new activities in the plan. Therefore, the PIP algorithm first differentiates between the new plan and the old plan to find the new activities. The new activities for the rover example are shown in Figure 1.6. Then, the initial conditions that support the new activities are found. The owner agents of these initial conditions are looked up in the relational knowledge base, and asked if the value of the condition they own is the most current value. If the value is correct, then the algorithm moves onto the next step. If not, the algorithm begins plan repair again with the new conditions.

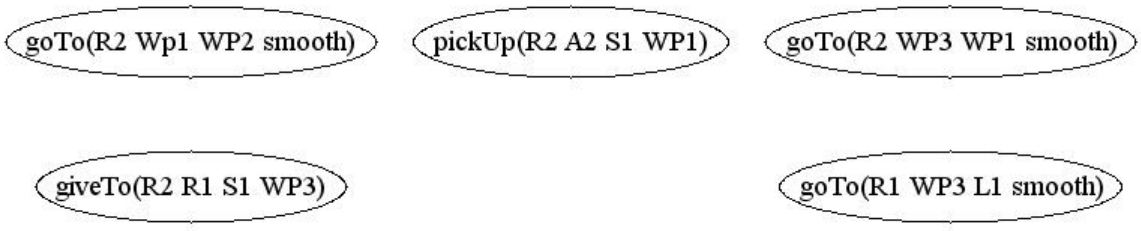


Figure 1.6. The set of new activities that need to be performed are found by differentiating between the new plan shown in Figure 1.5 and the old plan shown in Figure 1.3.

### 1.4.5 Finding Communication Agents

Once the owner agent finds the new plan, it must determine with whom to communicate the changed conditions. The owner agent needs to communicate with all agents who are either now performing a new activity, used to perform an activity, or

owns the initial condition supporting a new activity. In order to determine with whom to communicate, the owner agent differentiates between the new plan with the old plan to find the activities that are in one but not the other. The activities that are in the old plan, but not in the new plan are actions that no longer need to be performed as, shown in Figure 1.7 for the rover example. The activities that are in the new plan, but not in the old plan, are new activities that need to be performed, as shown in Figure 1.6 for the rover example.

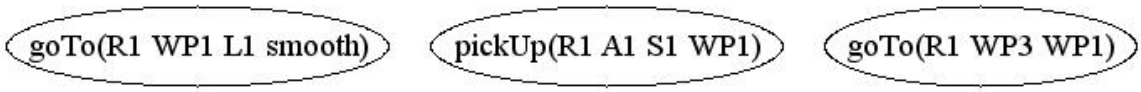


Figure 1.7. The set of activities that no longer need to be performed are found by differentiating between the old plan shown in Figure 1.3 and the new plan shown in Figure 1.5.

As can be seen in the example, Rover 1 and Rover 2 both perform activities in both sets. Thus, Rover 1 communicates with Rover 2 the changed conditions “Waypoint 1 is rough terrain” and “Waypoint 1 is not smooth terrain”.

Because Rover 2 and Rover 1 both started with the same plan and the changed conditions are the same, Rover 2 repairs its plan in the same manner as Rover 1 and ends up with the same plan.

## 1.5 Thesis Layout

Chapter 2 provides background information on previous work that relates to this thesis. Chapter 3 presents the strategies used in the algorithm. Chapter 4 introduces the representations used by the planning approach presented in this thesis. The algorithm for PIP is presented in Chapter 5. More in depth exploration of the algorithm through examples is presented in Chapter 6. The thesis is concluded in Chapter 7 with results and future work.

# Chapter 2

# Background

## 2.1 Overview

The algorithm presented in this thesis builds upon two areas of research: Classical Planning and Distributed Planning.

## 2.2 Classical Planning

In planning, an agent must determine a sequence of actions that moves the agent(s) from an initial state to one that achieves the problem goals. The agent is given a set of operators that describe those actions that can be performed to change the state of the world. The agent is also given the initial state of the world in which it belongs. The agent uses a planner to determine how the set of operators can be composed into an action sequence that allows it to arrive at the state in which its goals have been accomplished.

### 2.2.1 Condition Representation

The state of the world is described by a conjunction of conditions, where each condition is a proposition that describes the state of some object in the world, such as “the sky is blue” or “the terrain is not blue”. Each proposition is assigned true or false. A literal is a proposition, p, or its negation, not p. P is called a positive literal, while not p is called a negative literal. The statement “the sky is blue” is a positive literal describing the state of the sky as blue. The second condition “the terrain is not blue” is a negative literal where the proposition describes the state of the terrain being blue.

### 2.2.2 The PDDL Activity Representation

This thesis needs a representation for the activities that an agent can perform. The PDDL, an extended STRIPS activity language, is used.

The most common operator representation used to describe activities performed by an agent is STRIPS. A STRIPS operator consists of a conjunction of preconditions that must be satisfied before the start of the operation, and a conjunction of effects that are the end result of the activity. The preconditions are positive literals, while the effects may be positive or negative literals. The PDDL representation extends the STRIPS representation to include a set of variable parameters. The parameters are variables that are bound to terms denoting objects during planning. Figure 2.1 shows an example of a

PDDL operator that defines the pick-up operator of the example scenario presented in Section 1.2.1.

```
action PICKUP

parameters:    (and      (?r rover)
                         (?a arm)
                         (?s specimen)
                         (?l location))

preconditions: (and      (on-board ?r ?a)
                         (at_specimen ?s ?l)
                         (at_rover ?r ?l)
                         (not(have ?s ?r)) )

effects:           (and  (have ?s ?r)
                         (not (at ?s ?l)))
```

Figure 2.1 The activity “pick-up” in PDDL notation.

The preconditions tell us that to pick up a specimen, the agent must have an arm on board, it must be at the same location as the specimen, and it must not already have the specimen. If any of these things are not true, then the agent cannot pick up a specimen. The effects tell us that the agent will have the specimen at the end of the operation. Propositions that are not mentioned in the effects are left unchanged.

Rather than representing initial conditions, goals, and activities as three different data types, the initial conditions and the goals can also be modeled as restricted PDDL operators to simplify the planning algorithm. However, unlike activities, initial

conditions consist of only effects, while goals consist of only preconditions. Figure 1.1 shows an example of the initial condition activity and the goal activity of the rover example

### 2.2.3 Plan Characteristics

A plan describes sequences of actions that will lead the agent(s) from the initial state to a state that satisfies the problem goals. A plan is comprised of a set of actions, causal links, and orderings. The set of actions include the initial and goal actions. For the rover example, these are the actions in Figure 1.1.

An action is a PDDL operator that has been instantiated. To be instantiated, the variables in the operator are bound to objects in the object domain. In the rover example, rover 1 needs to use arm 1 to pick up specimen 1 at waypoint 1. The corresponding instantiated action is PICKUP(R1 A1 S1 WP1) as represented in Figure 2.2.

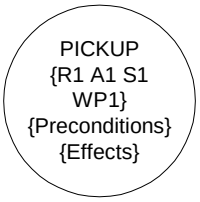


Figure 2.2. The instantiated PDDL operator for the action pickup. For aesthetics, the actual preconditions and effects have been left out. The preconditions are (on_board R1 A1), (at_specimen S1 WP1), (at_rover R1 WP1), and (not (have (S1 R1)). The effects are (have S1 R1) and (not (at S1 WP1)).

A causal link is a relationship between two actions specifying that the effect of one action is required to satisfy the precondition of another action [16]. A causal link specifies what actions supply needed effects that enable other actions to be performed. The causal link has a proposition P, a producer action $A_{Pr}$ that has P as an effect, and a consumer action $A_C$ that has P as a precondition. For example, a precondition of "analyze(R1 S1 AN)" is "Rover 1 has Specimen 1". In order for Rover 1 to have Specimen 1, Rover 1 needs to pick up Specimen 1 at Waypoint 1. Therefore, we say that the action "pickup(R1 A1 S1 WP1)" entails "analyze(R1 S1 AN)" with the reason being the fact "Rover 1 has Specimen 1".

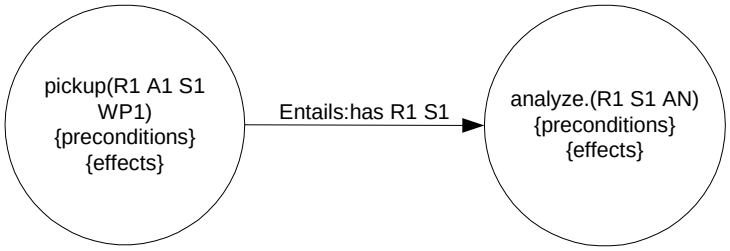


Figure 2.3. The top set of nodes is a general case of a causal link. The bottom set shows the specific example of picking up and analyzing a specimen. R1 = rover 1, WP1 = waypoint 1, S1 = specimen 1, AN = analyzer. Once again, the preconditions and effects have been left out to prevent clutter in the figure. The preconditions and effects of pickup are in Figure 2.2. The preconditions of analyze are (has R1 S1), (on_board R1 AN), (in S1 AN). The effects are (have_data S1).

A causal link imposes the constraint that no other action can occur during the period between the start and end of the causal link that would negate P. If an action $A_I$ exists that has the effect (not P), then the causal link is said to be "threatened" by $A_I$.

The third component of a plan is a set of orderings that are introduced to resolve conflicts to preconditions and effects in a plan. These conflicts, sometimes called mutex, result when there is a threat to a causal link, that is, when two actions produce effects that negate each other or when two actions consume preconditions that negate each other. In order to resolve threats to causal links, a plan specifies orderings between events. When an action $A_I$ threatens a causal link, the plan will either "demote", order the action after the link it threatens, or "promote", order the action before the link it threatens, $A_I$ to resolve the conflict.

Plans are either partial order or total order. Partial order plans are also known as least commitment plans since they make the minimal set of commitments to orderings and operators. Total order plans return a set of actions that have an absolute ordering. This thesis operates on partial order plans since total order plans introduce constraints that do not need to be communicated and are not relevant to the PIP algorithm.

A plan is *complete* when every precondition has been satisfied by the effect of some other action that has occurred before it. This occurs if for each precondition P for some action $A_C$, there exists a causal link in the plan with proposition P and producer $A_{Pr}$. For a cooperative team of agents, a complete plan will include all the actions of all the agents.

A plan is *consistent* if no contradictions occur in the plan, either in the ordering or the causal links. If an action $A_1$ is constrained to begin before an action $A_2$, and action $A_2$ is constrained to begin before action $A_3$, then action $A_3$ cannot begin before action $A_1$. In addition, the ordering of the plan is consistent if no action is threatened.

The algorithm in this thesis uses causal links and mutex to determine which activities are affected by changed conditions. It then generates a new complete and consistent plan that replaces activities that are no longer supported with the changed conditions.

### 2.2.4 Planner Characteristics

Planners can be either deterministic or non-deterministic. If there are several valid solutions to the planning problem, a deterministic planner will always choose the same one. Graphplan is an example of a deterministic planner [2]. In successive trials, a non-deterministic planner is not guaranteed to return the same solution every time if more than one solution exists. Walksat is an example of a non-deterministic planner[17]. WalkSat uses a random seed to randomly traverse the space of possible plans until it discovers one that is valid. A non-deterministic planner can be made into a deterministic planner by making the random seed deterministic. One way of making a deterministic random seed is to use a pseudo random generator.

The algorithm presented by this thesis uses a deterministic planner. It is important to use a deterministic planner in this thesis because our approach relies on the fact that two agents, given the same fact, must generate the same plan. If a non-deterministic planner was used, the agents would not be guaranteed to generate the same plan, and will thus not be able to agree on a shared plan.

## 2.3 Distributed Planning

As stated in the introduction, distributed systems offer several advantages over centralized systems. Distributed systems are more robust because other agents can take on the responsibilities of an agent who has failed. Agents working in parallel can also sometimes provide a speed advantage in problem solving and plan execution. Agents whose activities interact with each other must communicate to ensure that their actions do not result in any conflicts. In a centralized approach, conflicts are resolved by sending all the information to one agent. The central agent determines a set of actions that do not conflict and sends the resulting actions to the other agents. Communication can be decreased by having agents transmit information only to nearby nodes that need to know the information without having to go through an intermediary.

In most distributed systems, data and control are decentralized. Each agent has incomplete information about the world. The issue in distributed problem solving is how an agent updates its own plan of actions in a way that makes it consistent with the plans of other agents in the team. In most of the planning research performed to date [5][6], the

agents work simultaneously on a subpart of the planning problem. They communicate to find and resolve conflicts that arise between their sub-solutions through a series of strategies.

One strategy is called *task sharing*. In task sharing, the team works to divide the larger problem into smaller subproblems. Each of the subproblems is then assigned to an agent to resolve. The Contract-Net Protocol framework is an example of a task-sharing system[5]. Another strategy is called *result sharing*. Agents work simultaneously on finding a solution to a small portion of the plan. They then share their results to get a better view of the problem. If conflicts arise in their solutions, the agents iteratively exchange their tentative solutions until they resolve the conflicts. Distributed Constraint Heuristic Search is a framework that utilizes result sharing[6].

In the previous two approaches, agents explicitly communicate parts of plans to coordinate actions. In the task sharing approach, agents communicate subtasks that need to be performed. If a large number of assignments are made, large numbers of messages are passed amongst the agents, since for each subtask assigned, there is at least one message. In the result sharing approach, agents communicate their tentative results to a subproblem. The larger the solution to the subproblem, the more information needs to be passed. Furthermore, if conflicts arise, then the iterative resolution process will require more information to be passed.

This thesis presents a method whereby agents do not pass explicit information about parts of a plan but rather, passes information that can be used to implicitly infer parts of a plan. Also, unlike the previous approaches, agents are able to avoid having to negotiate conflict resolutions because they have a global view of the problem. Both of these characteristics of the thesis algorithm contribute to reducing communication.

# Chapter 3

# Pertinent Information Distributed Planning Algorithm Strategy

In the introduction, we briefly went over the algorithm and what steps were taken. In this chapter, we will go into more detail about the steps and the approaches taken for the algorithm and the reasons behind these steps and approaches. Agents working in cooperation must be able to effectively adapt plans to a changing environment while reducing communication by minimizing what needs to be transmitted between agents. The key in our approach to reducing communication is to communicate information on a need-to-know basis. In this approach, agents monitor the world to determine what changes have occurred (Section 3.2). Agents who observe changes to their environment make updates to their plans in order to compensate for their observations. They then analyze the updated plans to determine who needs to know about the changes (Section 3.3). During this analysis, they must identify those agents that are influenced by the changes and communicate the minimum number of observable changes necessary to ensure that the agents agree upon those parts of their shared plans that influence each other's actions (Section 3.4). The influenced agents then use these communicated observable changes to generate new plans (Section 3.5).

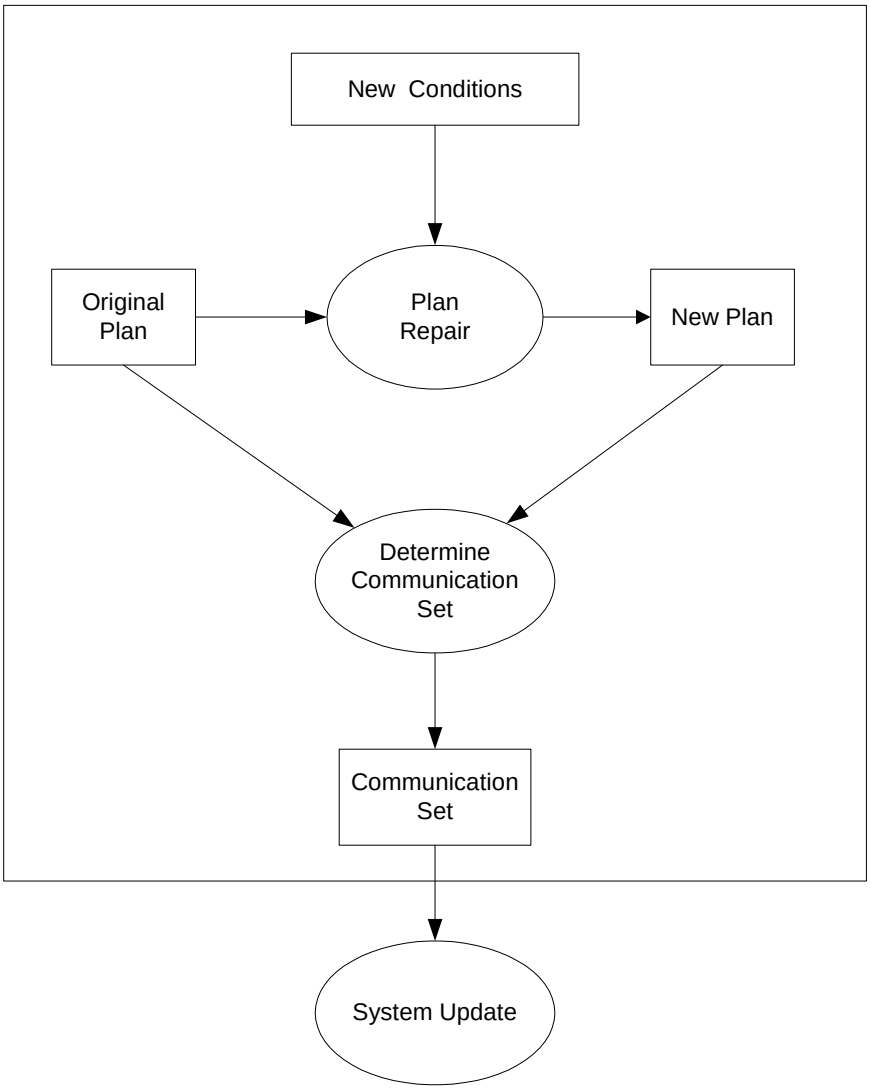


Figure 3.1. A flowchart of the algorithm where an individual agent's view is in the rectangular box. The new conditions are changes that are discovered in the world. These changes result in parts of the original plan that are invalid. The original plan (with the invalid parts) and the new conditions are given to a method that performs plan repair, in order to find a new plan. Both plans are used to determine who needs to know about the changes represented by the rectangle labeled "communication set". The changes are sent to the agents in the communication set who then also update their plan by performing re-planning on the communicated knowledge.

## 3.1 Describing the Problem

We consider in this thesis the effects of modifications under changing conditions to a multi-agent plan that has been developed for accomplishing a mission comprised of a set of goals. The agents become aware of dependencies with other agents through the multi-agent plan. For instance, in the rover example, the multi-agent plan requires that Rover 1 and Rover 2 must work together to finish the mission. Given the plan, Rover 1 and Rover 2 are both aware that they must cooperate with each other in order to accomplish the goals. Each of the agents is initially given a copy of the complete plan. We will refer to this as the original plan.

Since a plan is a set of logical steps, the conditions that are true in the initial state are called *initial conditions.* The initial conditions of the original plan are not guaranteed to be correct at the time of execution. For instance, sometimes a plan cannot be completed without knowledge of unspecified facts. In order to generate the plan, the planner must make assumptions about these facts. In the rover example, the plan cannot be complete unless rovers are assigned to visit particular waypoints. The assignments will depend upon their ability to traverse rough or smooth terrain. However, from space, it is difficult to determine if the terrain at Waypoint 1 is rough or smooth. The predetermined plan in the example makes the default assumption that the terrain is smooth. Assumptions, by definition, are not guaranteed to be correct. In another case, conditions that were correct when the plan was developed change over time. By the time the plan is ready to be executed, the conditions may be incorrect. For instance, at the time of plan

generation, Waypoint 3 could be within communication range of the space probe. The original plan, based on this assumption, would have the rovers set up the wide band communication device at Waypoint 3. However, by the time the plan is to be executed, Waypoint 3 has been moved out of communication range due to the rotation of the planet. In both cases presented, the initial condition that supports the plan is no longer valid. Usually, the original plan will fail when the initial conditions are not correct. However, the agents in the team can respond by discovering precisely the correct value of the conditions and by making alterations to the plan.

Once an initial condition has been discovered to be incorrect, the goals achieved by the plan that relied on this condition are no longer valid. For example, one of the goals of the original plan was to analyze a specimen from Waypoint 1. However, if Rover 1 cannot explore Waypoint 1 because the terrain is rough, then this goal cannot be accomplished. If there are alternative plans that achieve their goals, the agents must find and agree on one. In the case that Waypoint 1 is a rough terrain, an alternative plan that achieves the goal of analyzing a specimen from Waypoint 1 would be to have Rover 2 explore the waypoint instead.

In the scope of this thesis, we assume the agents always correctly observe the truth about all changing conditions. We do not address what happens if an agent observes faulty information from erroneous sensor readings or other impasses to gathering information. Because the information is incorrect, the plans generated by an agent are not guaranteed to work.

## 3.2 Initial Condition Monitoring

As previously stated, initial conditions in the original plan may be incorrect at the time of execution due to false assumptions or changing environmental conditions. In this thesis, we assume agents have the ability to correctly identify any condition that is different from an initial condition in the plan. Throughout the rest of this thesis, a condition that is different (or changed) from an initial condition from the plan due to either incorrect assumptions or changing environmental conditions is called a changed condition. All agents are given the task of monitoring all the environmental conditions. An alternative would be to assign each condition a monitoring agent. This approach is not flexible in a world in which the difference between the plan's initial condition and the real world condition may affect one agent's ability to monitor a condition. For example, in the rover example, assume Rover 1 was assigned to monitor the availability of a specimen at waypoint 1. During execution, Waypoint 1 turns out to be rough terrain, making it impossible for Rover 1 to traverse the terrain and monitor the availability of the specimen.

Even though all agents are monitoring for changes in the environment, only one agent needs to make the computation in order to determine who needs to know about the changes. One approach is to have the agent who discovers the change also perform the computations. However, since an agent's plan is guaranteed only to be correct with respect to the actions it performs or influences and the changed condition might affect a

part of the plan that the agent does not influence, then the agent could be making changes to an outdated plan. The alternative approach used in this thesis is to assign the task of responding to a change in each initial condition to a unique agent called the conditions' "owner". The agent keeps track of the changes to the parts of the plan that are supported by the initial condition. Thus, each agent's plan is guaranteed to be correct with respect to the parts that it influences and at the parts that are supported by the condition it owns. In order to keep track of the effects of the initial condition, the agent who owns a condition is responsible for generating an alternative plan when changes to the initial condition are discovered. The owner agent tells only the other agents it believes are affected by the change so that they could update their plans accordingly.

The distribution of initial conditions to agents affects the amount of communication required. For example, if Rover 1 were to own the fact "Specimen 2 is free", then when Rover 2 discovers the fact, it will need to tell Rover 1. Then Rover 1 will repair the plan and discover that Rover 2 is the only rover affected by the fact and send the information back to Rover 2. As can be seen, a randomized distribution is inefficient, since agents can own a condition that will never affect an activity they need to perform. Hence, they will always need to tell someone else. Instead, a more intelligent strategy involves distributing ownership based on characteristics such as relevance, location, skill or any other number of groupings that affect an activity an agent performs. We will discuss in the following paragraph how to assign ownership by relevance, location, and skill.

By relevance, we mean that if a condition describes the state of an agent, then that agent should be the owner, since agents usually perform actions that have a precondition describing its state. For example, if the condition said that "Agent 1 is available" then agent 1 ought to own that initial condition, since it describes Agent 1's availability for performing an action. Location refers to the geographical distribution of agents. The agent who is assigned to traverse a geographical location should be the owner of any initial conditions about the geographical location. This is because activities performed by an agent at that location are affected by the conditions of the location. For example, if Rover 1 should be the owner of the fact, "Waypoint 1 is smooth terrain" since it is the agent currently assigned to visit Waypoint 1. Rover 1's ability to traverse Waypoint 1 is affected by a condition about Waypoint 1. Skills are the capabilities of an agent. The capabilities of an agent affect the validity of an action the agent performs. If the fact was "the specimen analyzer is in use," the agent who can analyze specimens, Rover 1, should be the owner of the fact. In our problem domain, we assume that all agents know who owns which fact. In Section 4.3, the representation that keeps track of the relationships between agents and conditions, the *relational knowledge base*, will be introduced.

## 3.3 Repairing the plan

### 3.3.1 Basic Plan Repair

The section of the plan invalidated by the changed condition is referred to as the *broken subplan*. For the changed condition, "Waypoint 1 is rough," all the actions where

Rover 1 is at Waypoint 1 are now part of the broken subplan. In addition, other actions such as analyze Specimen 1 and send data Specimen 1, are all now part of the broken subplan. The owner agent needs to determine an alternative set of actions to achieve the affected sub-goals. The set of alternative actions make up a new subplan. The goals of this new plan are the goals previously achieved by the broken subplan. For example, the goal affected by the change “Waypoint 1 is rough” is “sent Specimen 1”.

Recall from the introduction that the owner agent finds an alternative plan to achieve the invalidated goals by repairing only the section invalidated by the broken subplan. Because only the broken subplan needs to be repaired, it makes sense to save work by building upon the currently existing plan to produce a new plan, that is, the owner agent will incrementally produce a new plan. The planning agent incrementally produces a new plan using a deterministic planner. We specify that the planner be deterministic because we want different agents to agree or infer the same plan given the same set of initial conditions.

Incremental planning is the addition of activities to a plan that achieves consistency with respect to existing activities already in the plan. We utilize a deterministic planner to achieve the same results as incremental planning, that is, it returns a plan that includes activities in the unbroken plan and new activities that are added to reach the subgoals of the broken subplan. We accomplish the same results as incremental planning by modifying the planning problem so that any plan the planner outputs always includes the existing activities.  This is accomplished through the use of a

specific set of operators called *strongly linked activities* which will be described in Section 4.2. Strongly linked activities are operators that are guaranteed to be included in the plan generated by the planner and connected by the same causal links as in the existing plan. The result of the planner is a complete plan that includes the original plan's unbroken activities and new activities that achieve the goals of the broken subplan. The portion of the output that replaces the broken subplan is called the *new subplan*. In the rover example, the new subplan is comprised of actions where Rover 2 is now going to Waypoint 2 rather than Rover 1.

Sometimes, it is the case that the new subplan contains many nodes that are similar to the broken subplan. Since the planning agent is only interested in what has changed, a method called *differentiate* is used to find the activities that are in one plan but not the other. The subplan that is part of the new subplan but not in the broken subplan is called the *differentiated new subplan*. The subplan that is part of the broken subplan that is not in the new subplan is called the *differentiated broken subplan.*

We guarantee that the activities in the differentiated new subplan produced by the planner are consistent with the current state of the world by checking the values of the initial conditions that support the new subplan. A plan output by the planner that is consistent with the current state of the world is called a *correct plan*. A correct plan is one that follows from a set of initial conditions that are confirmed to be correct. The planning agent must validate the initial conditions of the differentiated new subplan with the owner agents. For instance, one of the initial conditions in the rover example is that

Rover 2 can traverse rough terrain. Before Rover 1 assumes that Rover 2 can take over, it must check with Rover 2 that it can traverse rough terrain. If any of the initial conditions are discovered to be inconsistent with what the agent believes, then the agent must start from the beginning of plan repair.

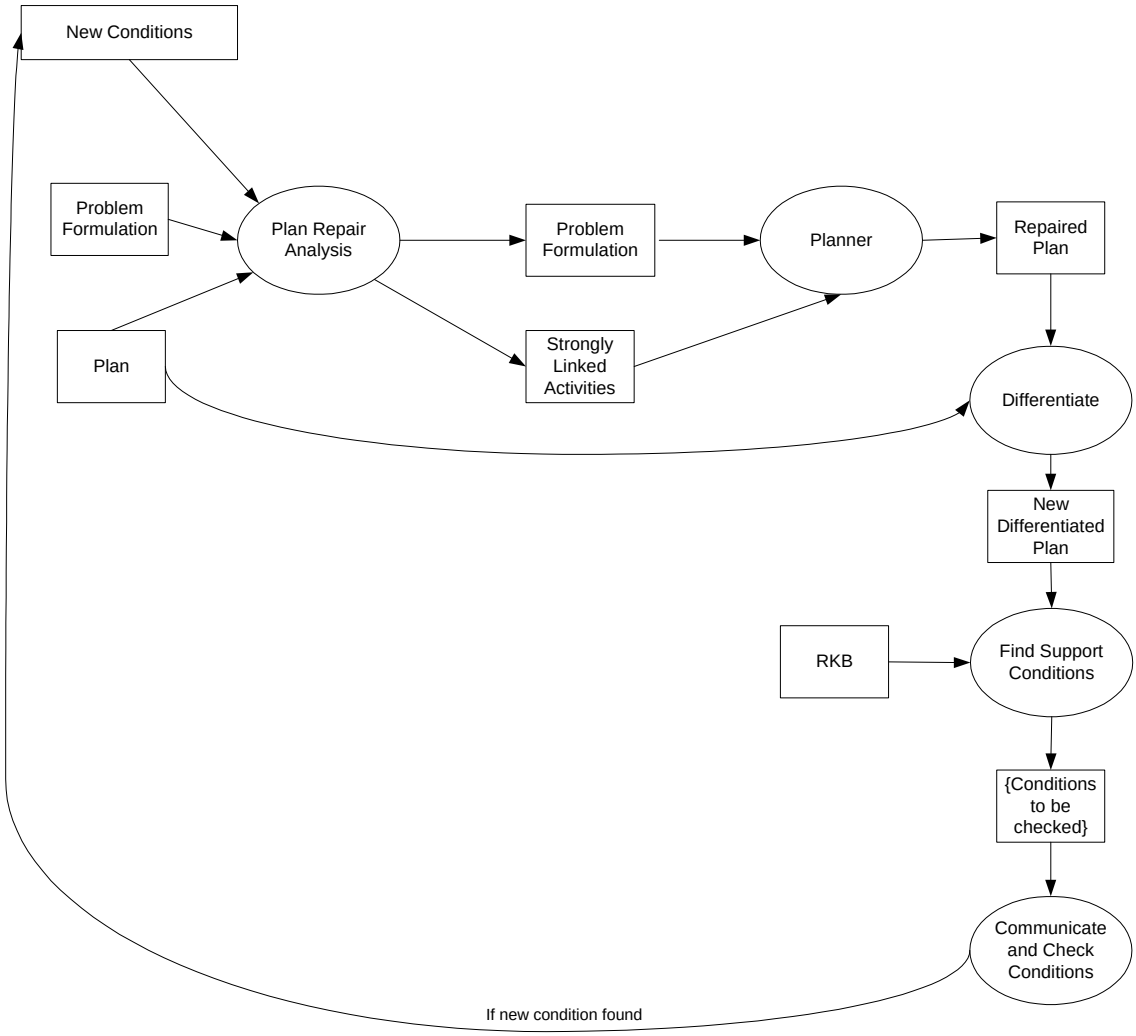


Figure 3.2. A flowchart for the plan repair as presented in Section 3.3.1. The plan repair analysis determines the broken subplan and translates the remaining plan to strongly linked activities. RKB stands for the relational knowledge base, as mentioned at the end of Section 3.2.

### 3.3.2 Handling Planner Failure

What happens if the planning agent fails to find a new plan? An agent will fail to find a new plan if 1) no consistent new plan is possible, 2) no new plan can be generated by the agent, given what the agent knows, or 3) a new plan exists but requires changes to the unbroken subplan. In Case 1, the mission fails because there is no possible solution. In Case 2, a plan might be possible but the agent does not have enough information to complete the plan. For instance, suppose the solution to the plan requires the activity goto(Rover 2, location Foo). Goto requires the preconditions “Rover 2 is available” and “the location Foo is traversable”. The broken subplan did not include goto because the initial conditions were originally “Rover 2 is not available” and “the location Foo is not traversable”. Now suppose Rover 2 finds out that “Rover 2 is available,” Rover 1 discovers that “the location Foo is traversable,” and the only way to achieve the goals is to include goto(Rover 2, location Foo) in the new subplan. Without both pieces of knowledge, neither rover can determine a new plan. One agent must pass the changed condition it has observed to the other agent. One strategy is to pass along the condition and the role of plan repair to a randomly selected agent within the broken subplan. This is because an agent within the broken subplan is most likely to need to change actions that it performs and so would need to know about the changed condition. If this agent cannot find a solution, then the changed condition and the role are passed on to another agent within the broken subplan until either a solution is found or none can find a solution. If

none of the agents in the broken subplan can find a new plan, then the problem is passed on to other agents in the team until either a solution is found or no solution is found. For this case, Rover 1 will determine that it is the only rover in the broken subplan. This is not useful since Rover 1 already cannot determine a plan. Therefore, Rover 1 will randomly choose an agent in the team, which is Rover 2. It will pass the condition "location foo traversable" to Rover 2 along with the message "I cannot solve this." Rover 2 now has enough information to perform plan repair and finds a solution.

## 3.4 Agent Communication

Our strategy for limiting communication is to transmit only changed conditions, and to communicate them only on a need to know basis. There are two categories of agents who need to know.

Type 1) Agents who need to change their actions as a result of a changed initial condition including:

a. Agents who should no longer perform an activity contained in the broken subplan,

b. Agents who now need to perform an activity in the new subplan but did not previously in the broken subplan.

Type 2) Owner agents who own initial conditions that support the parts of the plan that have changed:

a. Agents who own one of the initial conditions of the broken subplan,

b. Agents who own one of the initial conditions in the new subplan.

Type 1 agents need to be informed of the new initial conditions to plan the correct activities that it needs to perform or to determine that it no longer needs to perform an activity. For our example, Rover 2 needs to know some information since it is the rover who explores Waypoint 1 and perform the new actions associated with exploring Waypoint 1.

Type 2 agents are agents who might repair the part of the new subplan in the future and need to be sure that they are repairing the correct version of the plan. For our rover example, suppose the space orbiter owned the initial condition not(dangerous WP1). Now if Type 2 agents were not informed of the changed conditions “Waypoint 1 is rough terrain” and “Waypoint 1 is not smooth terrain”, that is, if the orbiter was not informed of the previous changes that Rover 2 and Rover 1 made to the plan, it will update the plan incorrectly when not(dangerous WP1) changes. When the probe finds out that WP1 is in fact dangerous, it thinks that Rover 1, being the one who used to traverse WP1 needs to change its actions. However, the broken subplan does not belong to the most current set of plans. The most current set of plans say that Rover 2 is traversing

WP1. The owner agent, the orbiter, ends up informing the wrong set of agents about the changes. However, if the orbiter was informed of the changed conditions that Rover 1 observes, then it knows that Rover 2 is now traversing WP1. When it finds out that WP1 is dangerous, it will send the information to the correct agent.

The set of agents that a planning agent must communicate with is called the *relevant agent set*. Once the planning agent determines the relevant agent set, it sends to each agent in the set all initial conditions it updated in the planning stage. This includes the changed condition that caused it to start repairing the plan and all initial conditions it updated during the validation of the plan. In our rover example, the only fact that has changed is the type of terrain. Therefore, Rover 1 would send Rover 2 the fact "WP1 is rough terrain" and "WP1 is not smooth terrain". This set of initial conditions is called the *relevant fact set*. The planning agent needs to communicate only the initial conditions necessary for the other agents to infer the same new subplan. At the start of the planning phase, all agents were given the same set of initial conditions. Therefore, the agents need to know only conditions that have changed from what was initially believed. Rover 2 does not need to know the values of the other initial conditions because it was initialized to the same values as Rover 1.

## 3.5 System Update

### 3.5.1 Updating Plans for Relevant Agent Set

The agents in the relevant agent set repair their plans using the information received in the relevant fact set. The relevant agent set need not validate the initial conditions in their new subplan since the planning agent has sent them the latest set of initial conditions.

There is one more task that the agents in the relevant agent set must do. A new subplan created by the planning agent might produce a conflict with another part of the complete plan. If an agent in the relevant agent set finds a conflict arc that arises from a part of its plan that has been previously changed, it determines the initial conditions that resulted in the change and tells the planning the initial condition. The relevant agents can detect changed activities due to a marking process that occurs at the end of the planning iteration as will be discussed in the next paragraph. If the planning agent had a different value for the condition, it begins plan repair over again. Otherwise, nothing is done with the information.

If no more changes are made to the plan with regards to the current changed condition, the new activities in the plan are marked with the changed condition to keep track of the changes made to the plan. The marked activities are used to detect conflicts as discussed above.

### 3.5.2 Resolving Concurrency Conflicts

Suppose that several conditions that are each owned by two different agents change within a small time window such that the delays in communication may result in two agents repairing the same section of the plan concurrently with two different sets of initial conditions. Let us see what happens in our rover world if two rovers plan concurrently. Suppose Rover 1 finds out that the waypoint WP2 is in smooth terrain that it previously thought was rough. Rover 1 will repair its plan and determine that to achieve the goal of analyzing specimen 2, it can go to WP2 rather than Rover 2 and retrieve a specimen. Rover 2, at the same time, discovers that it has a specimen analyzer. Rover 2's new plan is to analyze the specimen without the help of Rover 1. Both rovers have changed their plans in such a way that the other rover must also change its actions. To make sure the other rover knows of the changes, the rovers will tell each other their changed conditions. When Rover 1 receives the fact that Rover 2 has a specimen analyzer, it will disregard the fact because it assumes that Rover 2 no longer needs to analyze the specimen at WP2. When Rover 2 finds out that the terrain is smooth, it will disregard the fact because Rover 2 will have already determined that it can analyze its own specimen without the help of Rover 1. Now both rovers will redundantly examine the specimen at WP2.

One way to resolve concurrency issues is for the agents to label each initial condition with the time that it was discovered. Agents repair and confirm plans in the order that initial conditions are discovered. Sometimes, an agent will need to backtrack

and repair a plan it had already repaired. In order to do this, each agent saves its old plan within memory, and recalls the correct plan when needed. In the previous example, suppose Rover 1 finds out its changed condition first. Therefore, it sends Rover 2 the fact "the terrain is smooth" and a time stamp. Rover 2 recognizes that the discovery of "the terrain is smooth" occurred before the discovery of "Rover 2 has a specimen analyzer." It recalls the plan it had before it made the changes to "Rover 2 has a specimen analyzer." Rover 2 generates a plan that also says Rover 1 will be the one to explore WP2 and retrieve the specimen. Then it will try to repair its plan with the fact "Rover 2 has a specimen analyzer." Like Rover 1, it does nothing with this fact since the fact is irrelevant.

# Chapter 4

# Representing Plans and Initial Conditions

This chapter introduces an enhancement to the representation of plans, in order to enable tracking of conflicts. In addition, it introduces strongly linked activities to enable incremental planning-like behavior, and the relational knowledge base to keep track of relationships between initial conditions, agents, and activities.

## 4.1 Augmentations to the plan representation

When a planner generates a plan, activities are ordered to resolve threats. In this thesis, we want to keep track of these conflict resolutions, because agents who generate a new plan need to be aware of when they resolve conflicts with a part of the plan that is used by another agent.

In causal link representations of plans, conflict resolutions are not labeled. For this thesis, we add labels to those arcs in a graphical representation of a plan that represent conflict resolutions. If an activity $A_C$ produces a proposition (not P) and an activity $A_P$ that consumes a proposition P, and the two activities are not connected through causal links, then the arc is labeled as a conflict arc with proposition P as the reason for the conflict. For example, the activity “goTo(R1 WP3 WP1)” produces the

effect “not (at_rover R1 WP1)”. However, the activity “pickUp(R1 A1 S1 WP1)” requires the precondition “(at_rover R1 WP1)”. Since “goTo(R1 WP3 WP1)” occurs after “pickUp(R1 A1 S1 WP1)” and produces a proposition that negates the precondition of pickUp, a conflict arc is added between the two activities that with the label “Conflict(at_rover R1 WP1)”.

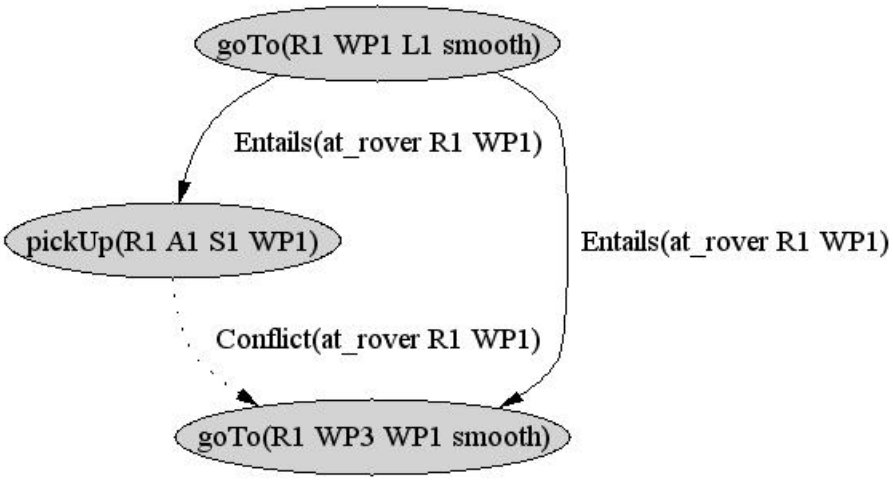


Figure 4.1. The proposition (at_rover R1 WP1) is needed by pickUp(R1 A1 S1 WP1) but the proposition not(at_rover R1 WP1) is produced by goTo(R1 WP3 WP1 smooth). The conflict arc shows how the planner has demoted goTo(R1 WP3 WP1) in order to resolve the conflict.

## 4.2 Strongly Linked Activities

This thesis uses a generative planner to generate plans. However, when the planner repairs the plan, incremental planning-like behavior is required. Incremental planning builds upon activities already in the plan. Thus, we want to represent activities

that must remain unchanged in the plan in a form that the planner is guaranteed to choose.

Planners generally take in as inputs, a set of initial conditions, a set of goals, and an action domain. The initial conditions are positive literals that are true in the initial state of the world. The goals are the literals that are desired to be true in the final state. The action domain is a set of PDDL operators that the planner can use for planning.

Strongly linked activities are operators that are required to be included in any plan generated by the repair planner. Strongly linked activities have unique preconditions and effects that can be satisfied by a causal link with only one other strongly linked activity. All strongly linked activities are included in a plan because they have an effect that is required by the goal state. Each activity is transformed into a strongly linked activity operator with the following steps:

1) A new STRIPS operator is created for the activity represented by each node. Unlike a normal operator, the new operator has no parameters. Instead, the preconditions and effects are instantiated to the values of the parameters of the activity in the unbroken subplan and contain no variables.
2) A new dummy fact called $G_{ActivityName}$ is created. It is added as an effect to the new STRIPS operator created in Step 1 and as a precondition to the goal operator of the plan.

3) If the activity is a producer of a condition that produces a causal link, then a new dummy fact called $Flag_i$ is created. It is added as an effect to the operator and as a precondition to the operator represented by the activity that consumed the condition in a causal link.

A planner must choose an action if it is the only way to achieve a goal. Therefore, every strongly linked activity must have a goal as an effect. To do this, a unique goal is added to the set of the plan goals and correspondingly added as an effect to each action.

A planner must also choose an action if it is the only way to satisfy a precondition of another action already in the plan. In a plan, preconditions are satisfied by causal links. For each set of activities in a causal link, a flag is added to the precondition of the tail activity with the same flag added as an effect to the head activity.

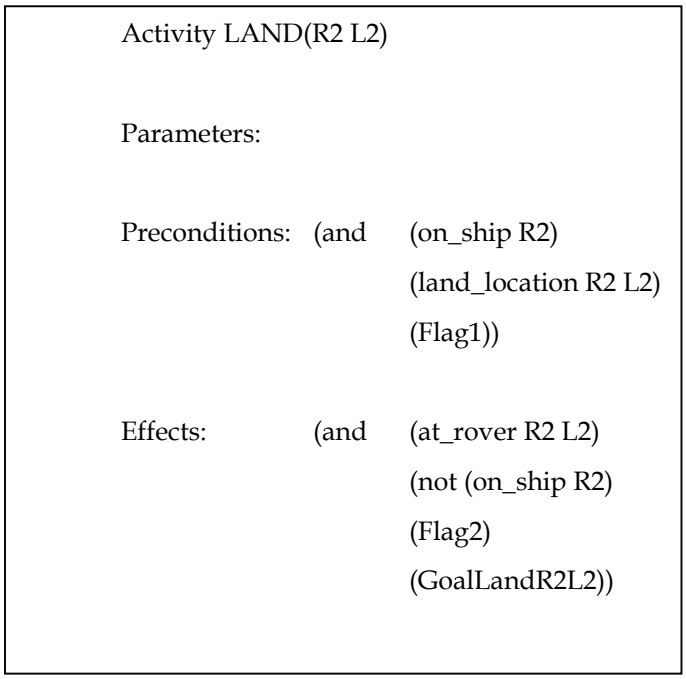


Figure 4.2.The strongly linked activity as a PDDL operator for the activity “land(R2 L2)” shows the addition of dummy variables. The dummy variable Flag 1 is added to the preconditions while the dummy variables Flag 2 and GoalLandR2L2 are added to the effects.

## 4.3 Relational Knowledge Base

In artificial intelligence, the set of conditions that an agent believes is called a knowledge base. For this thesis, an agent needs to be able to keep track of not only what conditions it knows, but must also be able to keep track of the relationships between these conditions, the agents in the team, and the activities in the plan. For example, the agent needs to keep track of owner agents and the conditions they own and the conditions that support an activity. A relational knowledge base is a structure that keeps track of these relationships. Each agent has its own RKB.

In Section 3.2, we discussed the concept of initial condition monitoring and ownership. Each initial condition has an agent that is responsible for keeping the most current version of the condition in its knowledge base. That agent is the owner agent. Part of the relational knowledge base is a database that associates each condition with the owner agent of that condition. Figure 4.3 shows the part of the relational knowledge base that keeps track of owner agents for the rover example.

The relational knowledge base allows an agent to differentiate between the conditions it knows and those that it believes. The conditions that an agent knows are the ones owned by that agent. The conditions that an agent believes are owned by other agents.

Conditions stored in the RKB are of two types: initial or dependent. Initial conditions are in the initial state of the world. Dependent conditions are those conditions that become true through changes made by activities. The RKB has an associative structure that keeps track of the relationship between dependent conditions and initial conditions. An initial condition supports a dependent condition if there exists a sequence of activities $\{A_1 \ldots A_n\}$ such that $A_1$ has as its precondition the initial condition and $A_n$ has as its effect the dependent condition and there is a set of causal links that connect $A_1$ to $A_2$ and so on to $A_n$.

The relational knowledge base allows an agent to look up the owner agent of an initial condition. It also allows an agent to look up the initial conditions that support a dependent condition. Those initial conditions are called *supports*.

| **Initial condition** | **Owner Agent** |
|---|---|
| On_ship R1 | Rover 1 |
| On_ship R2 | Rover 2 |
| On_board R1 A1 | Rover 1 |
| On_board R2 A2 | Rover 2 |
| On_board R1 AN | Rover 1 |
| On_board R2 chiseler | Rover 2 |
| On_board R1 WBCD | Rover 1 |
| Land_location R1 L1 | Rover 1 |
| Land_location R2 L2 | Rover 2 |
| Empty AN | Rover 1 |
| Is_type WP1 smooth | Rover 1 |
| Is_type WP2 rough | Rover 2 |
| Is_type WP3 smooth | Rover 1 |
| At_specimen S1 WP1 | Rover 1 |
| At_specimen S2 WP2 | Rover 2 |
| At_specimen S3 WP3 | Rover 1 |
| Free S1 | Rover 1 |
| Free S2 | Rover 2 |
| Free S3 | Rover 1 |
| Can_traverse R1 smooth | Rover 1 |
| Can_traverse R2 rough | Rover 2 |
| Can_traverse R2 smooth | Rover 2 |

Figure 4.3. The left column contains initial conditions that are owned by the agent in the right column.

## 4.4 Summary

In Chapter 3, we went over the algorithm at a high level. However, some of the finer details were left out in Chapter 3, because some of the representations had not yet been presented. The three representations presented in this chapter enable us to present a more detailed description of the specific methods underlying PIP in Chapter 5.

# Chapter 5
# Cooperative Distributed Planning Algorithm

In Chapter 3, we went did not say specifically how the algorithm operates on the plan and planner inputs, but just what needed to be done. In this chapter, we provide more details about the methods that operate on plans and planer inputs.

## 5.1 Overview

The PIP algorithm described in this thesis extracts a minimal set of changing initial conditions to be passed amongst agents. This allows the agents to generate a plan that is always correct *with respect to the tasks that it is to perform* although it may be inconsistent with respect to tasks that other agents perform. The inconsistency is a result of the fact that an agent is not informed of changed conditions that do not support any action that the agent performs. The benefit of allowing these inconsistencies to exist is that it reduces the number of conditions communicated. These inconsistencies are acceptable because they are guaranteed not to influence the performance of any action. Our planning algorithm is divided into three phases. The first phase consists of the agent that "owns" certain initial conditions repairing its plan with respect to the changed conditions. The second phase consists of the agent determining which other agents it must send the changed conditions and other conditions relevant to the change. In the third stage, the "informed" agents, ones who receive the changed conditions update their plans

in order to make the same changes to the plan as the agent who sent them the condition and inform the owner agent if any inconsistencies arise.

## 5.2 Plan Repair

### 5.2.1 Finding a new plan

Recall that plan repair is accomplished by replacing the broken subplan with a new subplan. Given a set of changed conditions, the broken subplan can be found by tracing the causal links in a plan. The pseudocode for finding a broken subplan is given below.

Method: Find Broken Subplan

Input: A plan and a set of changed conditions

Output: a set of activities in the plan

$S_V$ = set of visited activities.

$Q_V$ = queue of activities to be expanded.

Both $S_V$ and $Q_V$ are initially empty

Initially, for each changed condition, add to $S_V$ the set of activities that have the changed condition as a precondition.

For each activity affected by the changed condition, add the activity to $Q_V$ if the activity is not already in $S_V$.

while ($Q_V$ is not empty)

Let $A_P$ be an element of the queue $Q_V$

For each activity $A_C$ that is supported by $A_P$ through a causal link, if $A_C$ is not already in $S_V$, add $A_C$ to $S_V$ and to $Q_V$.

Take $A_P$ off of the queue $Q_V$

end while

The resulting set of activities, $S_V$, is the set of activities in the broken subplan.

Figures 5.1-5.3 shows how the broken subplan is found for the rover example in which Rover 1 discovers the changed conditions “Waypoint 1 is rough terrain” and “Waypoint 1 is not smooth terrain.” Figure 5.1 shows the activity “goTo(R1 WP1 L1 smooth)” that has the changed condition as a precondition. Figure 5.2 shows the activities that are connected by a causal link to “goTo(R1 WP1 L1 smooth).” Figure 5.3 shows the resulting broken subplan.

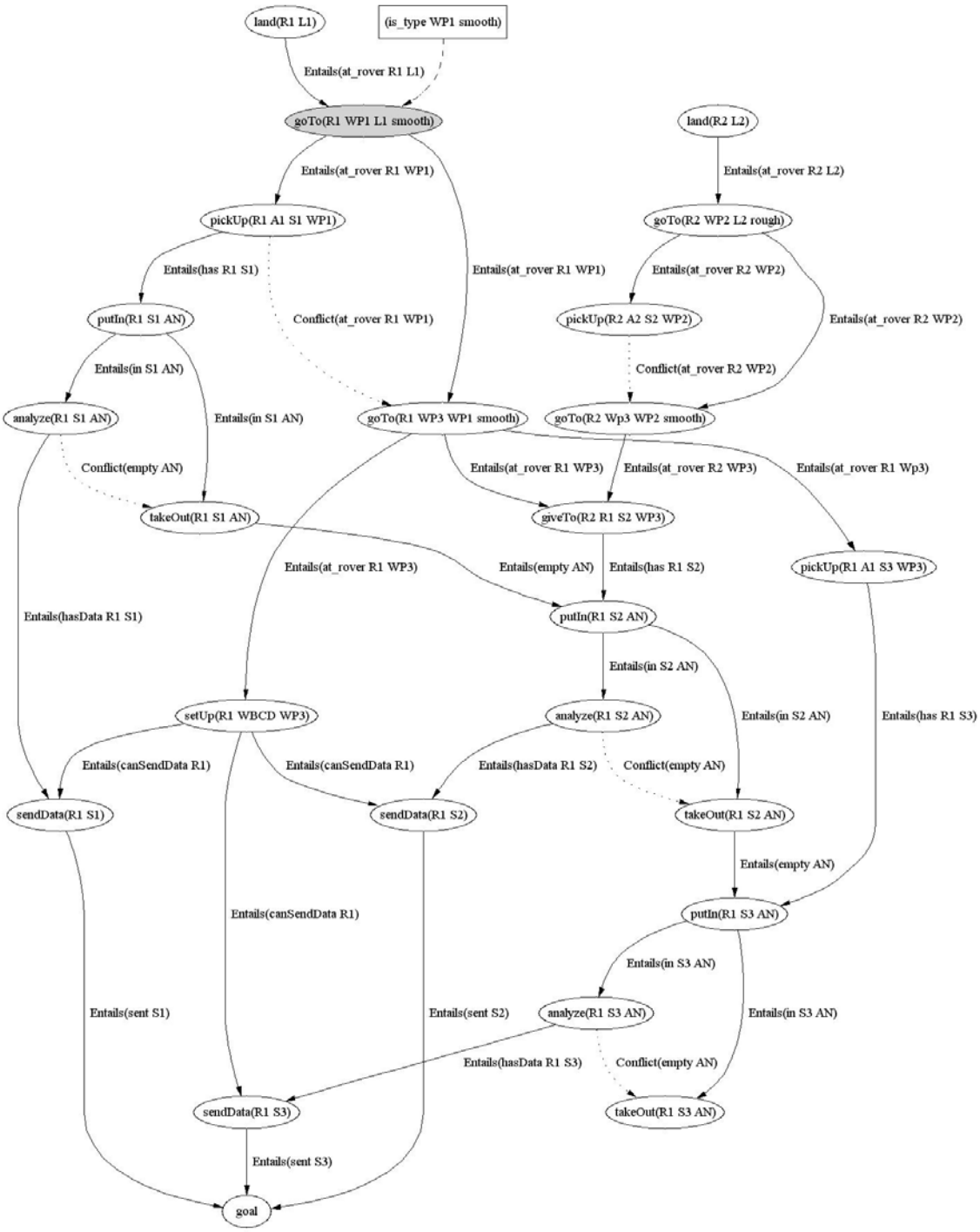


Figure 5.1. Rover 1 observes the changed conditions “Waypoint 1 is rough terrain” and “Waypoint 1 is smooth terrain” which influences Rover 1’s action “goTo(R1 WP1 L1)”.

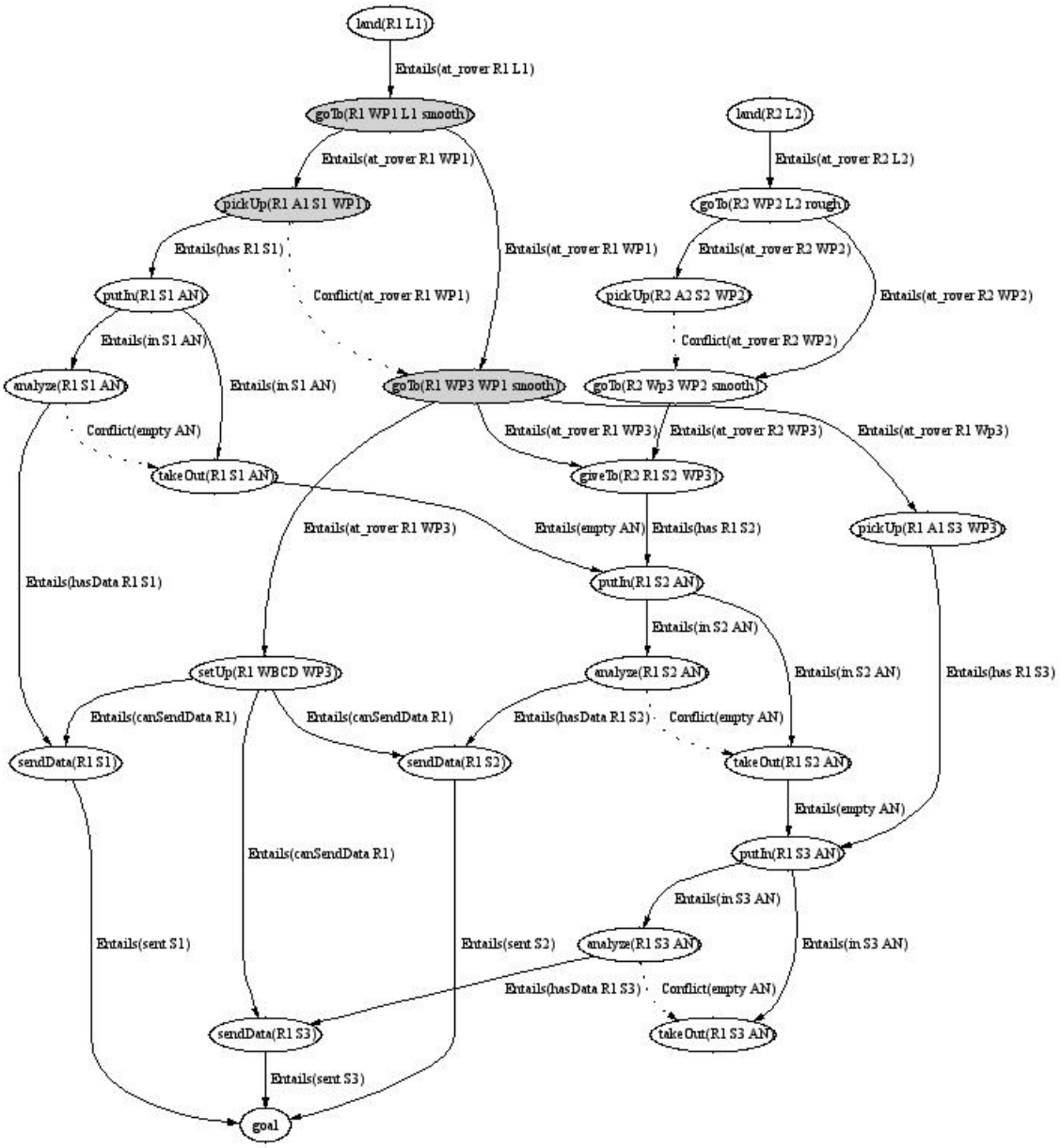


Figure 5.2. The activities “pickUp(R1 A1 S1 WP1)” and “goTo(R1 WP3 WP1)” are also affected by the changed condition through causal links from the activity “goTo(R1 WP1 L)”.

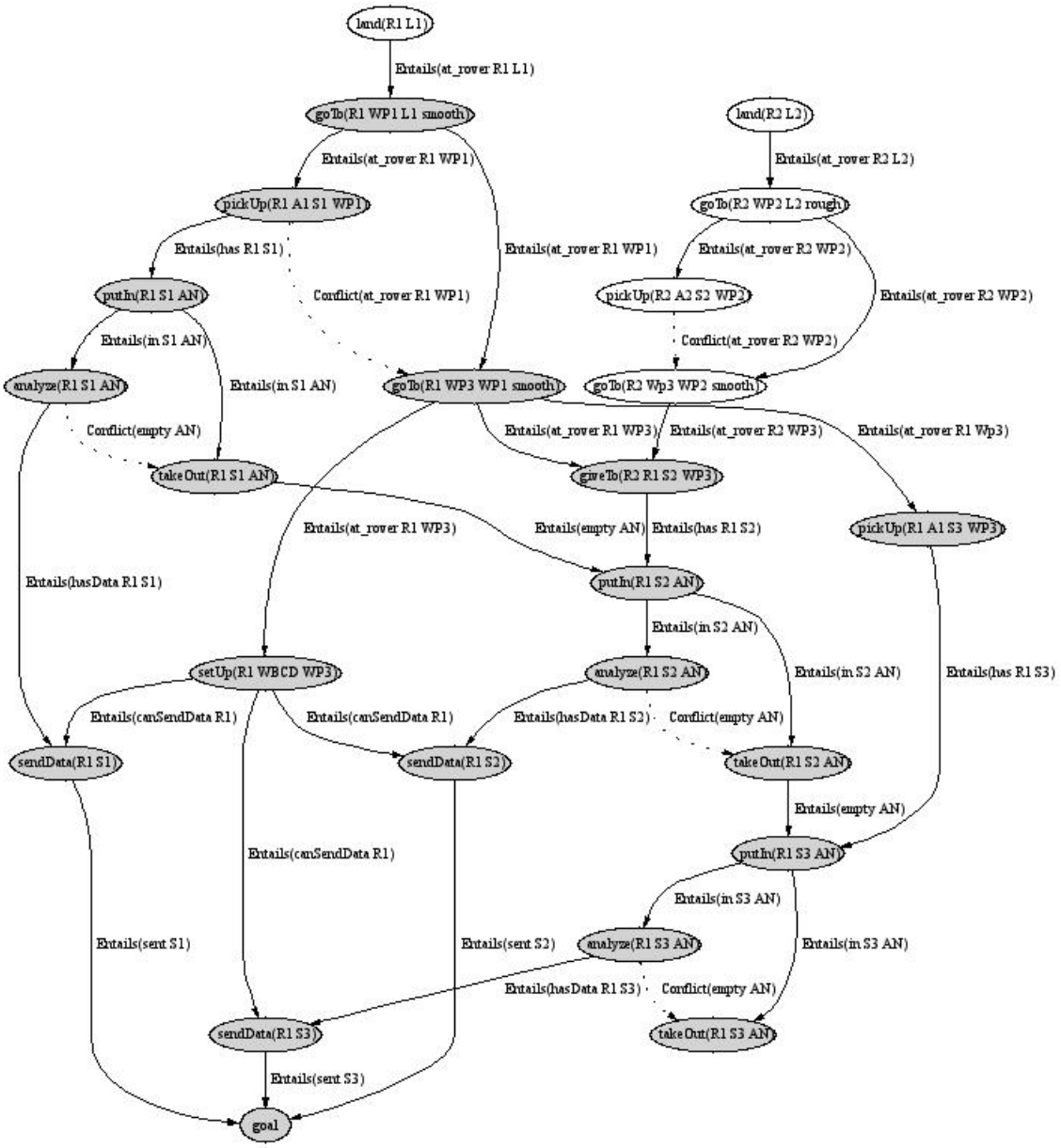


Figure 5.3. The shaded nodes represent the broken subplan as a result of the changed conditions “Waypoint 1 is rough terrain” and “Waypoint 1 is not smooth terrain”.

Recall that to repair the plan, we add to the original planning problem the constraint that the uninfluenced activities must appear in the new plan. The unshaded activities are activities that are not part of the broken subplan. Having identified the

activities in the broken subplan, the next step involves reformulating the plan repair problem by transforming each uninfluenced activity into a “strongly linked activity” PDDL operator.

Figure 5.4 shows the strongly linked activity PDDL operator representing the activity “land(R2 L2)”. Recall in Section 4.2 the process that creates this PDDL operator. The activity “land(R2 L2)” is made into a PDDL operator with instantiated conditions and no parameters. “GoalLandR2L2” is also added to the goal operator. “Flag1” is added to the start operator and “Flag 2” is added to the “goTo(R2 WP2 L2)” operator.

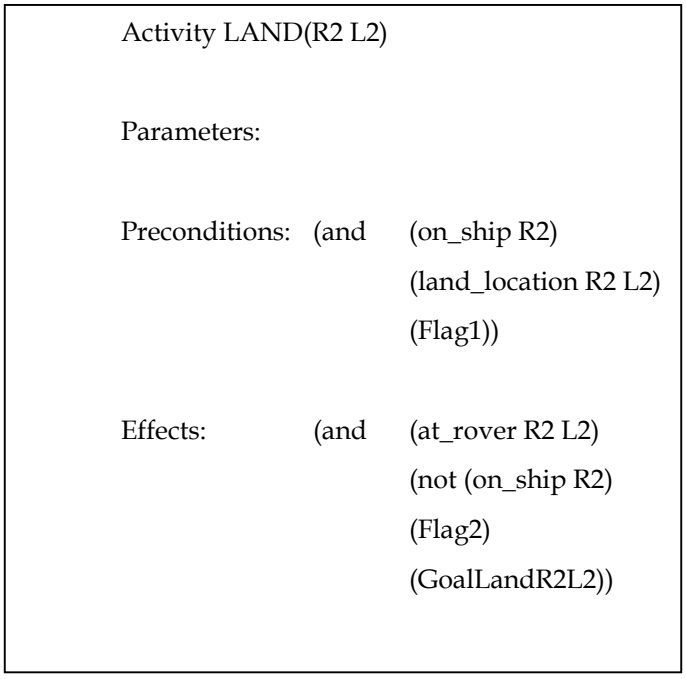


Figure 5.4. The strongly linked activity PDDL operator is added to the action domain. The dummy condition “Flag 1” is added both to the preconditions and to the start operator. The dummy condition “Flag 2” is added to the effects and to the activity connected by a causal link from this activity. The dummy condition “GoalLandR2L2” is added to the effects and to the goal operator.

The strongly linked activities are all added to the action domain, as are the modified initial condition and goal operators. These three inputs are given to the planner in order to generate a new plan. A PDDL planner is then invoked on the reformulated planning problem, resulting in a new plan. The new plan for the rover example is shown in Figure 5.5.

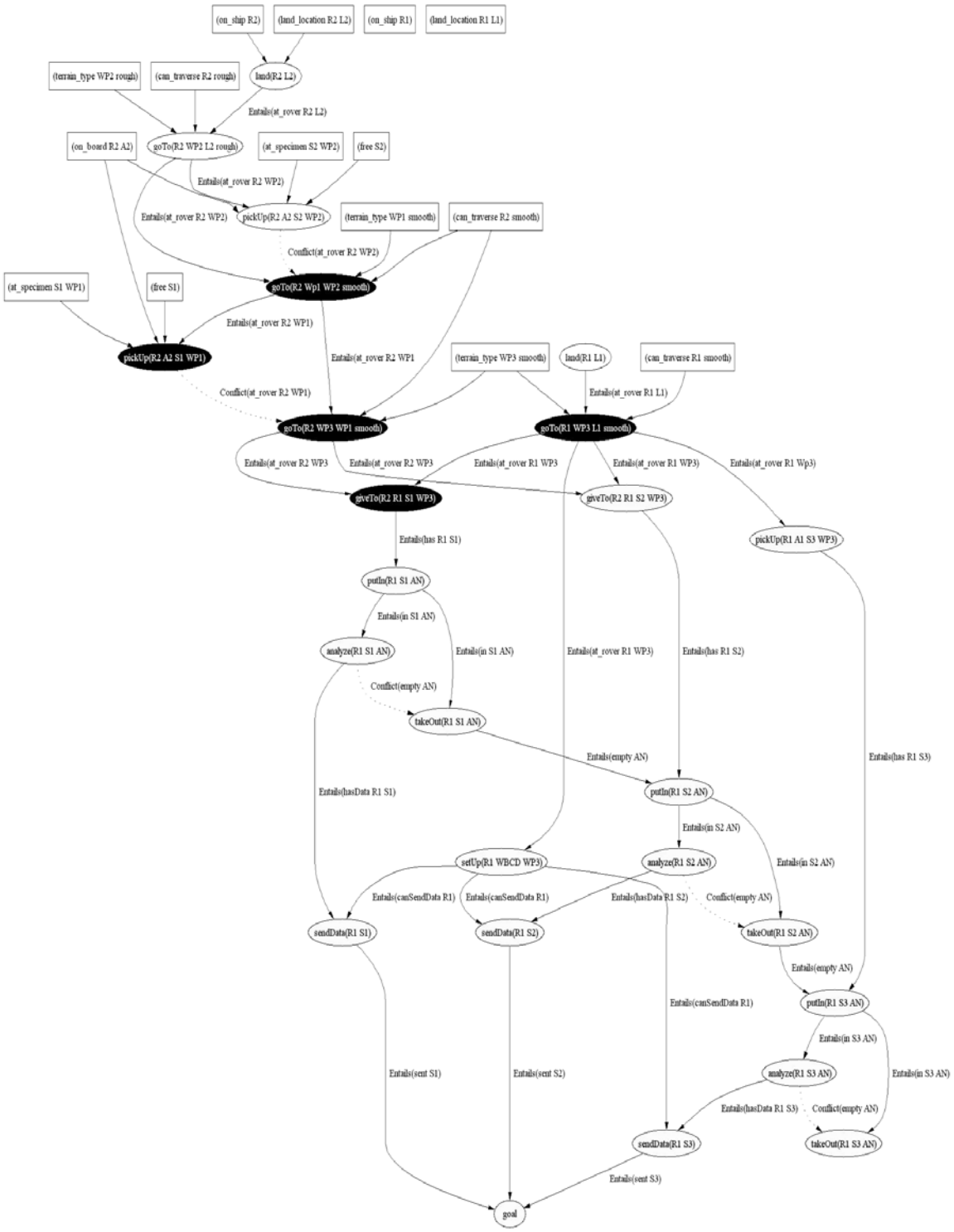


Figure 5.5. The new plan contains strongly linked activities in the unshaded nodes, activities that have not been changed by the plan repair in the grey nodes, and new activities that must now be performed by the black nodes.

Once the new plan is found, the agent doing the planning completes the repair phase by checking to make sure that the new plan generated is a valid plan with respect to the current conditions in the world. For an activity in the plan to be valid, the initial conditions supporting it must be correct.

### 5.2.2 New Plan Verification

The planner is guaranteed to have been updated with respect to any changes of conditions supporting the broken subplan. However, by introducing new actions, the new plan may introduce dependencies on new conditions that the agent has not been tracking. Rather than checking all the initial conditions that support the new plan, the agent checks only the initial conditions that support the activities of the new plan that are different from the old plan. Checking this subset of the initial conditions significantly reduces the amount of communication if the set of activities that have changed between the old plan and the new plan is small.

To check the subset of initial conditions involves two steps: 1) differentiating activities between the two plans, and 2) verifying the initial conditions that support the new activities.

For Step 1, a method called *differentiate* is used to perform a set differentiation on the activities in the two plans. Differentiate identifies the new activities introduced in the new plan.

Differentiate is a method that takes as an input two plans and returns as an output the set of activities that are in the first plan, but are not in the second plan. Differentiate is basically a set difference operator on the two sets of activities in the two plans. The pseudocode is given below.

Method: Differentiate

Input: Plan1 and Plan 2

Output: A set of activities

$S_A$ = set of activities to be returned. Initially empty

$S_{A1}$ = set of activities in plan1

$S_{A2}$ = set of activities in plan2

For each activity in $S_{A1}$

    If the activity is not in $S_{A2}$, add the activity to $S_A$

End for

Return $S_A$

In order to determine whether an activity is in another set, we need a notion of activity equality. Two activities are equal if and only if 1) their names are the same and 2) their parameters are bound to the same objects.

Figure 5.6 shows the set of activities that result from using differentiate on the new plan and the original plan for the rover example. The resulting activities are

activities that are in the new plan but not in the old plan and are part of the differentiated new subplan. These are activities that now need to be performed to achieve the goal state.

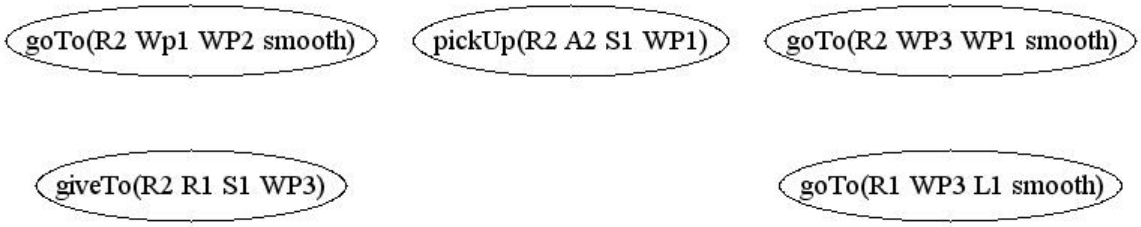


Figure 5.6. Invoking differentiate on the new plan (Figure 5.5) and the original plan (Figure 5.3) results in the new activities depicted here.

Once differentiate returns the new activities from the repaired plan, the planning agent checks the value of the initial conditions that support these new activities to make sure the new activities will safely execute. The pseudocode for initial condition checking is shown below.

Method: Initial condition Checking

Input: A set of activities corresponding to a plan and the set of changed initial conditions

Output: None

$S_P$ = set of initial conditions to be checked. Initially empty.

For each activity A in the input

    For each precondition of A

        Look up the initial conditions that supports the precondition in the RKB

        Add the supports to $S_P$ if the support is not in the set of changed initial conditions

    End For

End For

For each initial condition in $S_P$

    Look up the owner agent in the RKB

    Ask the owner agent if the value of the initial condition is correct

End for

If any of the initial conditions are incorrect, restart plan repair again with the new initial conditions.

For each new activity, the supports of the preconditions are looked up in the RKB. These supports together form the initial conditions that need to be checked. The agent will look up the owner agent of each of the initial conditions. It then asks each of the

owner agents what the correct value of the condition is. If all the conditions are correct, then the agent will continue. Otherwise, it will start at the beginning of plan repair using the new information.

For our example, the initial conditions to be checked are:

1 (on_ship R1)
2 (on_ship R2)
3 (on_board R2 A2)
4 (land_location R1 L1)
5 (land_location R2 L2)
6 (terrain_type WP2 rough)
7 (terrain_type WP3 smooth)
8 (at_specimen S1 WP1)
9 (at_specimen S2 WP2)
10 (free S1)
11 (free S2)
12 (can_traverse R1 smooth)
13 (can_traverse R2 smooth)
14 (can_traverse R2 rough)

The RKB associating these initial conditions with agents is shown in Figure 4.3

Of the initial conditions to be checked, only initial conditions 2, 3, 5, 6, 9, 11, 13, and 14 belong to Rover 2. Therefore, Rover 1 asks Rover 2 if the values it has for those initial conditions are correct. Rover 2 looks up the value of each of the initial condition that Rover 1 has sent in its relational knowledge base. Since nothing else in the world has changed yet, Rover 2 sends Rover 1 a message saying no values have changed. Rover 1 finds that all initial conditions are correct as is.

At the end of the repair phase, the owner agent has generated a plan modification that corrects for the changed condition it has observed and that it has confirmed to be correct with respect to current conditions. The number of messages communicated so far is 9. There were 8 initial conditions that needed to be verified and 1 message from Rover 2 confirming that none of the initial conditions are wrong.

## 5.3 Communicating Changed Initial conditions

Once the planning agent has found a new subplan that is supported by correct initial conditions, it needs to succinctly communicate the new subplan to the relevant agents. To accomplish this, it needs to determine what other agents also need to know about the changed initial conditions in order to re-derive the subplan modifications. In Section 3.4, we discussed two types of agents who need to know about the changes. In this section, we will discuss how each type of agent is found.

The first type consists of agents who need to change the actions they perform as a result of the changed conditions, either by performing new activities, or deleting the execution of old activities. The agents who should no longer perform an activity are the agents who performed activities in the set of activities found by invoking the method differentiate on the old plan and the new plan. The agents who now need to perform an activity in the new plan but did not previously in the old plan are found by invoking the method differentiate on the new plan and the old plan.

Figure 5.6 from above showed the activities that result from using differentiate on the new plan and the original plan, and are activities that now need to be performed. Figures 5.7 below shows the activities that result from using differentiate on the original plan and the new plan for the rover example, and are activities that were in the original plan but not in the new plan. These are activities that no longer need to be performed and are part of the differentiated broken subplan.

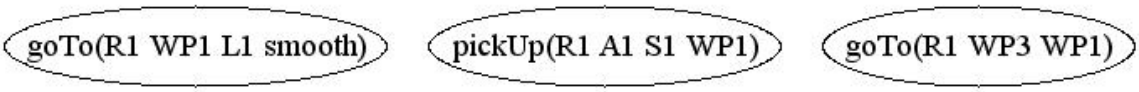


Figure 5.7. Invoking differentiate on the original plan (Figure 5.3) and the new plan (Figure 5.5) results in these activities that no longer need to be performed.

The second type is comprised of agents who must update part of the plan when initial conditions change. As stated in Section 3.4, there are two cases: agents who own one of the initial conditions of the differentiated broken subplan and agents who own one of the initial conditions in the differentiated new subplan. The pseudocode to find both types of agents is shown below.

Method: Find Relevant Agent Set

Input: Differentiated Broken Subplan and Differentiated New Subplan

Output: A set containing the names of agents

$S_A$ = set of agents who need to know about the changed initial conditions

For each activity in the differentiated broken subplan, add the name of the agent who performs the activity to $S_A$.

For each activity in the differentiated new subplan, add the name of the agent who performs the activity to $S_A$.

$S_A$ now contains the agents from finding the first type of agents.

For each activity in the differentiated broken subplan and the differentiated new subplan

    For each precondition $P_i$ of the activity

        Find the initial conditions that support $P_i$ in the RKB

        For each of the support initial conditions

            Look up the owner agent in the RKB and add the agent to the $S_A$

        End for

    End for

End for

The planning agent sends the set of changed initial conditions to the agents in $S_A$. For our example, Rover 1 finds that Rover 2 is in the relevant agent set.

At the end of this phase, the owner agent sends the changed conditions and any other new conditions it discovered during initial condition validation to the agents in the relevant agent set. For our example, Rover 1 sends Rover 2 the two conditions “Waypoint 1 is rough terrain” and “Waypoint 1 is not smooth terrain”.

## 5.4 System update

At the end of the previous phase, the agents in the relevant agent set are sent the conditions that have changed. For this phase, when each agent receives an update consisting of the set of changed conditions, it must perform the same repair step as the planning agent, so it is informed of the changes to its plan that the planning agent has identified. At the end of the previous phase, Rover 1 sends Rover 2 the two new conditions it discovered, more precisely, it sends Rover 2 “Waypoint 1 is rough terrain” and “Waypoint 1 is not smooth terrain.” Now the number of messages communicated has increased by 2 from 9 to 11.

Since the rovers had identical plans at the beginning of our example and receive identical changed conditions, Rover 2 goes through the same process as Rover 1 in Figures 5.1-5.5.

Each agent in the relevant agent set also checks the conflict arcs in its plan. If any of the conflict arcs connects a node that is part of a previously modified subplan to a node in the new subplan, the agent tells the planning agent the supports of the modified node.

The pseudocode for checking conflicts is shown below where the planning agent is the agent that sent the changed condition that caused the conflict. In our example, the owner agent is Rover 1.

```
Method: Check Conflicts

Input: Plan, Planning Agent

Output: C_C Set of Changed Conditions

A_Cf = Conflict arcs in a plan

For each arc A in A_CF,
        If one activity connected by A is marked
        and the other activity is in the differentiated new subplan
                Add reason for marking to C_C
        End If
End For

If C_C is not empty
        Send C_C to the owner agent
```

Given the new fact, the planning agent will replan to acknowledge the conflict. The Check Conflicts method allows the planning agent to avoid accidentally ignoring potential threats to its new plan. Since Rover 2 has not previously modified any part of its plan, it does not tell the Rover 1 anything. Rover 1 makes a note on the new activities in its new plan that they are new conditions as a result of the changed conditions “Waypoint 1 is rough terrain” and “Waypoint 1 is smooth terrain.” In Section 6.2, we will see an example where Rover 2 does tell Rover 1 about a conflict

At this point, both Rover 1 and Rover 2 have the same plan, and so are in agreement at the activities where they interact with each other. By exchanging only part of the knowledge bases, the two rovers are able to adapt to a change in the environment.

# Chapter 6

# Exploring More Examples

Two more examples will be shown in the rover world to help explore the subtleties of the PIP algorithm. One shows when the agent does not need to communicate and the other where the agent does need to communicate.

## 6.1 No Communication

Rover 2 finds out upon reaching Waypoint 2 that no loose specimens are available. The literal for representing a loose specimen is "(free S2)." In the original plan, the literal was positive but Rover 2 has discovered the initial condition is actually a negative literal. Rover 2 is now the planning agent. It notices the changed condition affects the activity "pickup(R2 S2 WP2)" since a precondition of pickup is "free(S2)," meaning that Specimen 2 must be free. The first step is to find the broken subplan. Figure 6.1 shows the broken subplan for this example.


Figure 6.1. The shaded activities represent the broken subplan as a result of the changed condition "not(free S2)" which is represented by a rectangular node in the top right hand corner.

The unshaded activities are translated into strongly linked activities. The new action domain, the new initial conditions, and the new goals are given as input into the planner. The planner returns an output plan as shown in Figure 6.2.

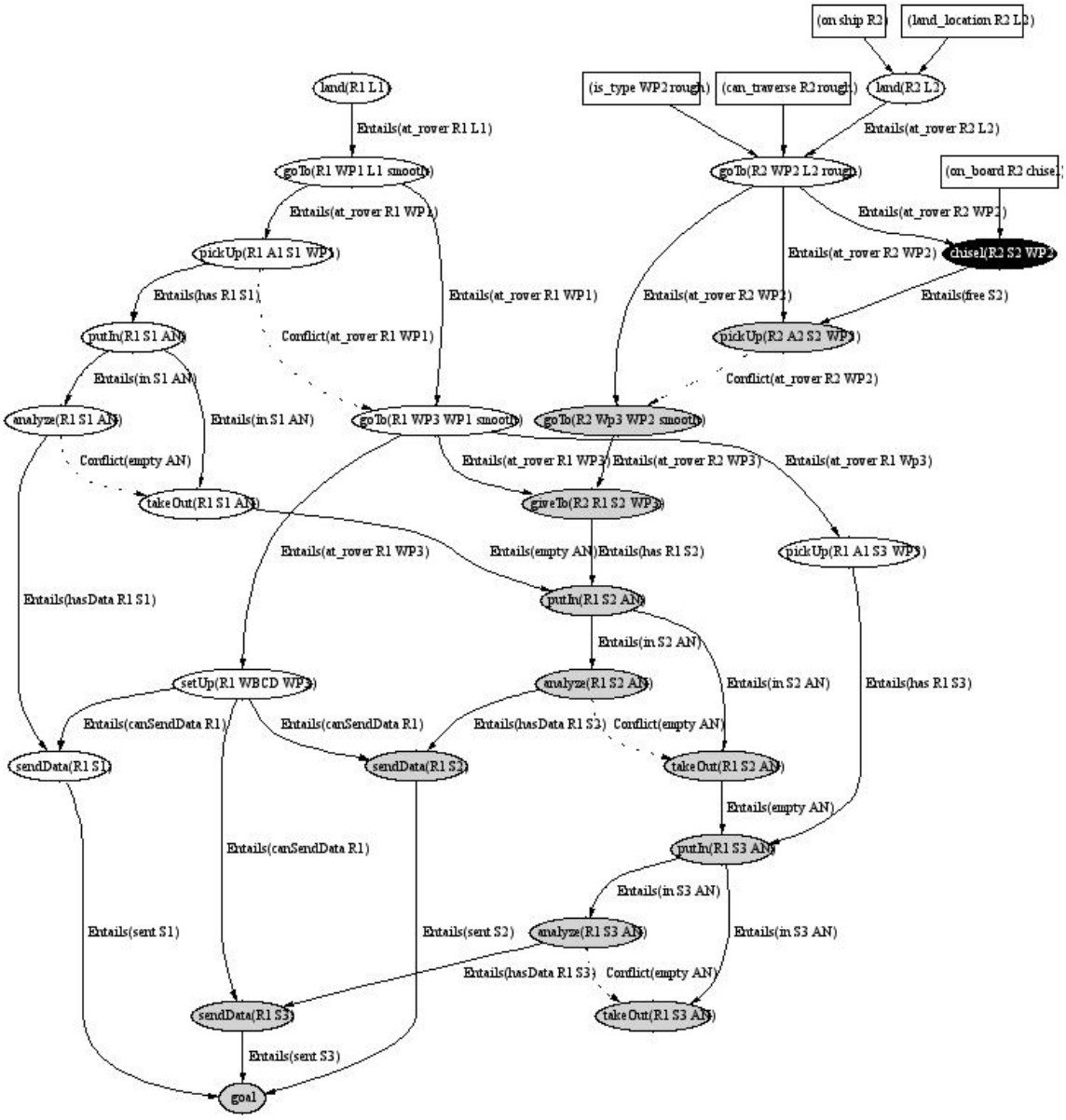


Figure 6.2. The new plan returned by the planner. The shaded nodes are part of the new subplan. The black node is the only activity that is in the new plan that is not in the old plan. The initial conditions that support the plan up to the differentiated new subplan are shown as rectangular nodes.

The new plan is differentiated from the old plan. The differentiated activities are shown in Figure 6.3.

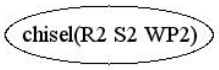


Figure 6.3. For this example, “chisel(R2 S2 WP2)” is the only differentiated activity.

Rover 2 looks up the initial conditions that support each of the activities. The initial conditions that support the new differentiated plan are:

(is_type WP2 rough)

(can_traverse R2 rough)

(on ship R2)

(land_location R2 L2)

(on_board R2 chisel)

Rover 2 looks up the owner agent of the initial conditions in the relational knowledge base. The relational knowledge base is shown in Figure 4.3. The owner agent of all the initial conditions is Rover 2. Because Rover 2 owns all the initial conditions, it knows that are values of the conditions are correct, and therefore, the plan is valid.

Rover 2 invokes differentiate on the old plan and the new plan to find out which activities need to be dropped. This set is empty since no activities are dropped.

Rover 2 goes through each activity in both differentiated sets to find the agent(s) who perform(s) the activity. The only activity is “chisel(R2 S2 WP2)” which is performed by Rover 2. Rover 2 then looks up in the RKB the owner of the initial conditions that support the activities in both differentiated plans. Rover 2 is the only owner agent of all the support initial conditions. Thus, the communication set is {Rover 2}. Hence, no communication is required. Rover 2 makes a note on the activity “chisel(R2 S2 WP2)” that it is a new activity as a result of “not(free S2).”

At the end of this process, Rover 1’s plan has not changed and is represented by the plan in Figure 6.1 while Rover 2’s plan has changed and is represented by the plan in Figure 6.2. However, the two rovers can still act together since the only time Rover 1 and Rover 2 interact is when Rover 2 gives Rover 1 Specimen 2. In Rover 2’s plans, the activities that affect Rover 1, more specifically, giveTo, has not changed.

## 6.2 Communication Required – 2 way – Example 3

In the previous two examples, Rover 2 has not needed to communicate with Rover 1 at the end of Phase 3. Rover 1 never needed to know that something has changed in Rover 2’s world. Therefore, Rover 2 never needed to communicate with Rover 1. This example will show when Rover 2 needs to inform Rover 1 about a change, due to a conflict.

We make a small change to the rover example involving the chiseler. The chiseler is now carried on a separate small rover that can traverse both types of terrain. It is initially on board Rover 2 but can be remotely controlled by Rover 1 and dispatched to a different location if Rover 1 needs to use it. The operators “unboard” and “board” are introduced to represent the actions Rover 1 can call on the chiseler. These operators are added to the action domain to reflect these changes.

Let us assume that the rovers have just completed the planning process in the “no communication” example. Suppose Rover 1 finds out that there are no loose specimens at Waypoint 1. Rover 1 first finds the broken subplan for the changed condition “not(free S1).” The shaded activities in Figure 6.4 are part of the broken subplan.

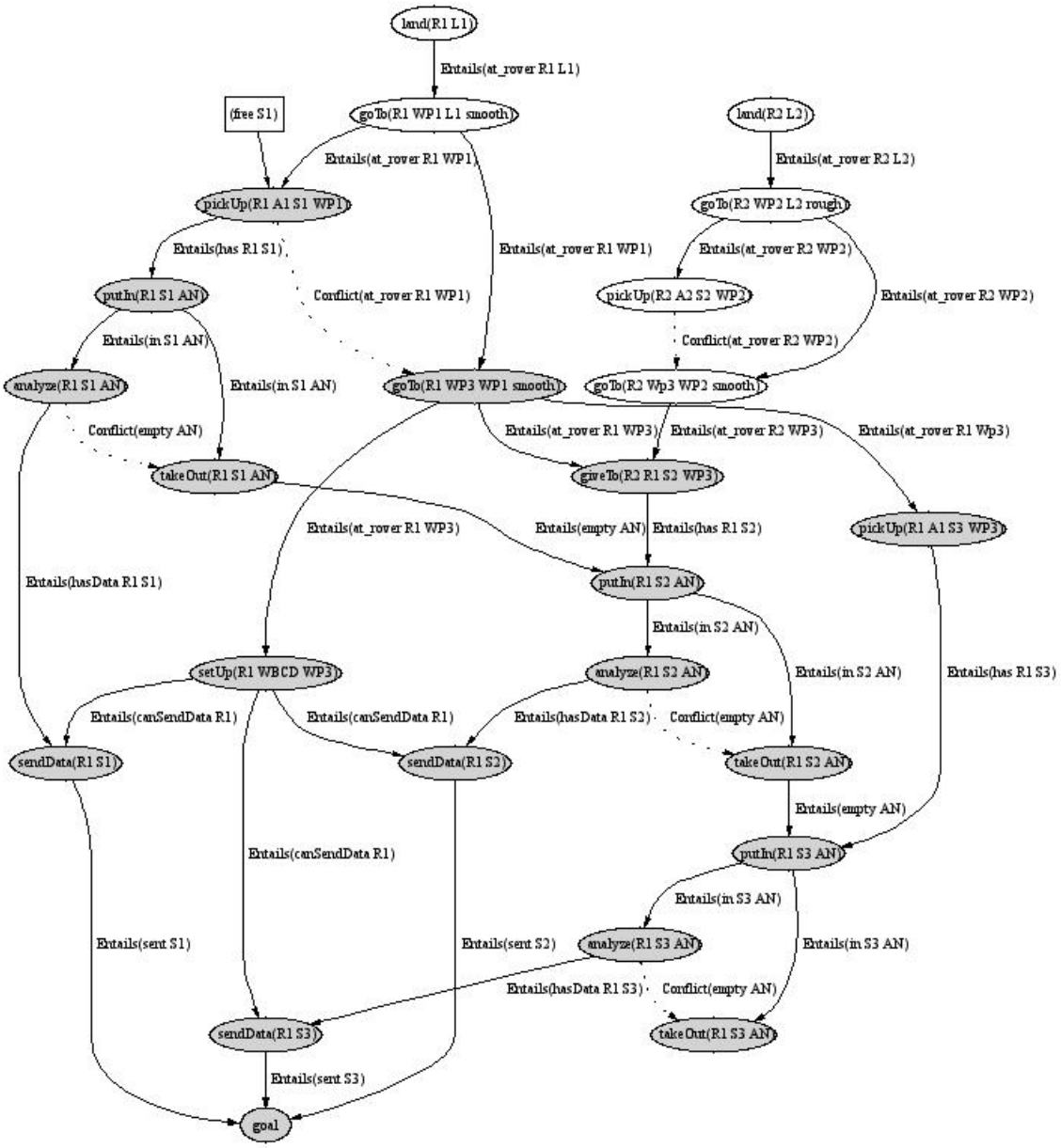


Figure 6.4. The shaded activities are the broken subplan for the changed condition “not(free S1)” shown in the rectangular node.

The unshaded activities are translated into strongly linked activities. The new action domain, initial conditions, and goals are given to Rover 1’s planner. The planner returns a new plan as shown in Figure 6.5 with a close up of the new activities and their

supporting initial conditions in Figure 6.6. Figure 6.7 shows the set of activities that result from invoking differentiate on the new and old plans.

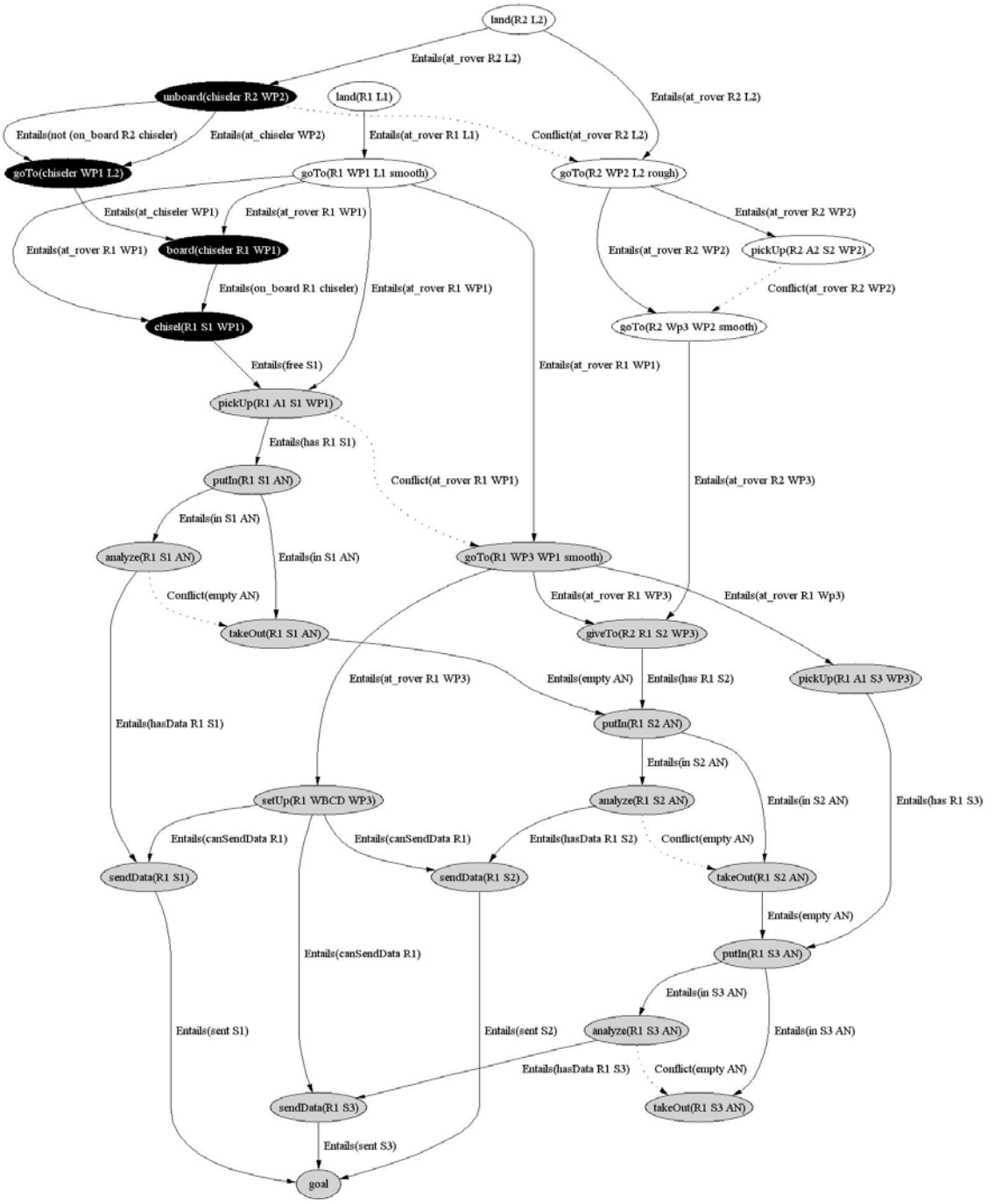


Figure 6.5. The new plan generated by Rover 1's planner for the changed condition "not(free S1)".

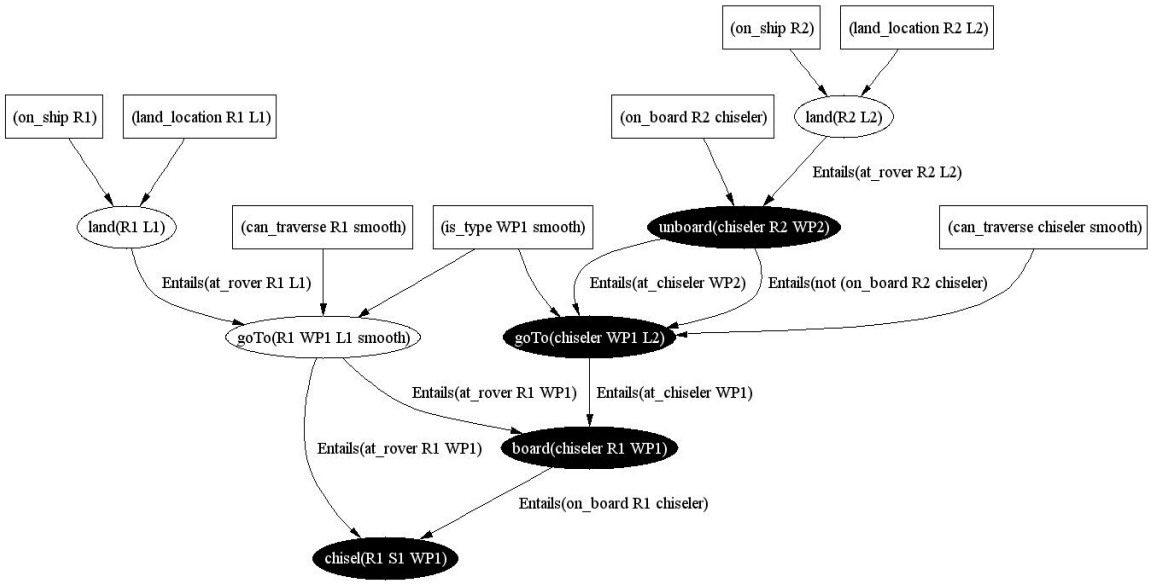


Figure 6.6 is a close up of the new activities that now need to be performed and the strongly linked activities that support the new activities.

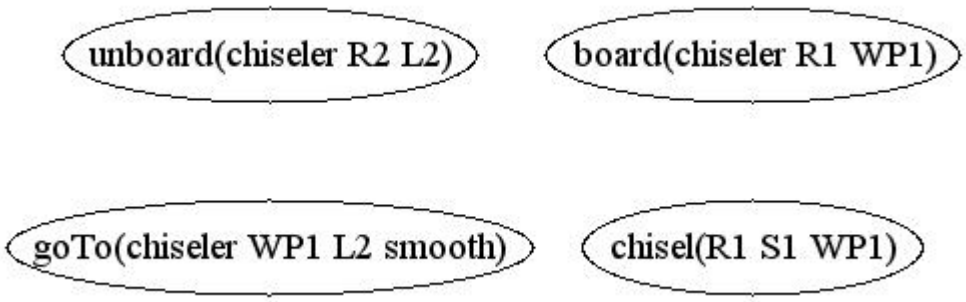


Figure 6.7. The differentiated activities showing new activities that need to be performed due to the changed condition “not(free S1)”.

The initial conditions that support the activities in the differentiated new plan are:

1. (on_ship R1)
2. (on_ship R2)
3. (landing_location R1 L1)

4. (landing_location R2 L2)

5. (on_board R2 chiseler)

6. (on_board R1 arm)

7. (can_traverse R1 smooth)

8. (can_traverse R2 smooth)

9. (can_traverse R2 rough)

10. (can_traverse chiseler rough)

11. (can_traverse chiseler smooth)

12. (terrain_type WP1 smooth)

13. (terrain_type WP2 rough)

14. (at_specimen S1 WP1)

For this example, we assume the same owners as in the previous example. The initial conditions about the chiseler are owned by Rover 2, since Rover 2 initially had the chiseler on board. Rover 1 verifies initial conditions 2, 4, 5, 8, 9, 10, 11,and 13. Since none of these initial conditions have changed so far, Rover 1 receives all positive confirmations from checking the values of the initial conditions. By the end of this stage, 9 messages are communicated, 8 from the initial conditions that need to be verified and 1 from Rover 2 responding that all initial conditions are correct.

Rover 1 now finds the set of agents who need to know about the changed initial condition. Just by looking at the differentiated new plan, we know Rover 2 needs to know

about the changed initial condition. The number of messages communicated rises by one to 10.

Once Rover 2 receives the changed initial condition, it will also find a broken subplan. Figure 6.8 shows the broken subplan for Rover 2.

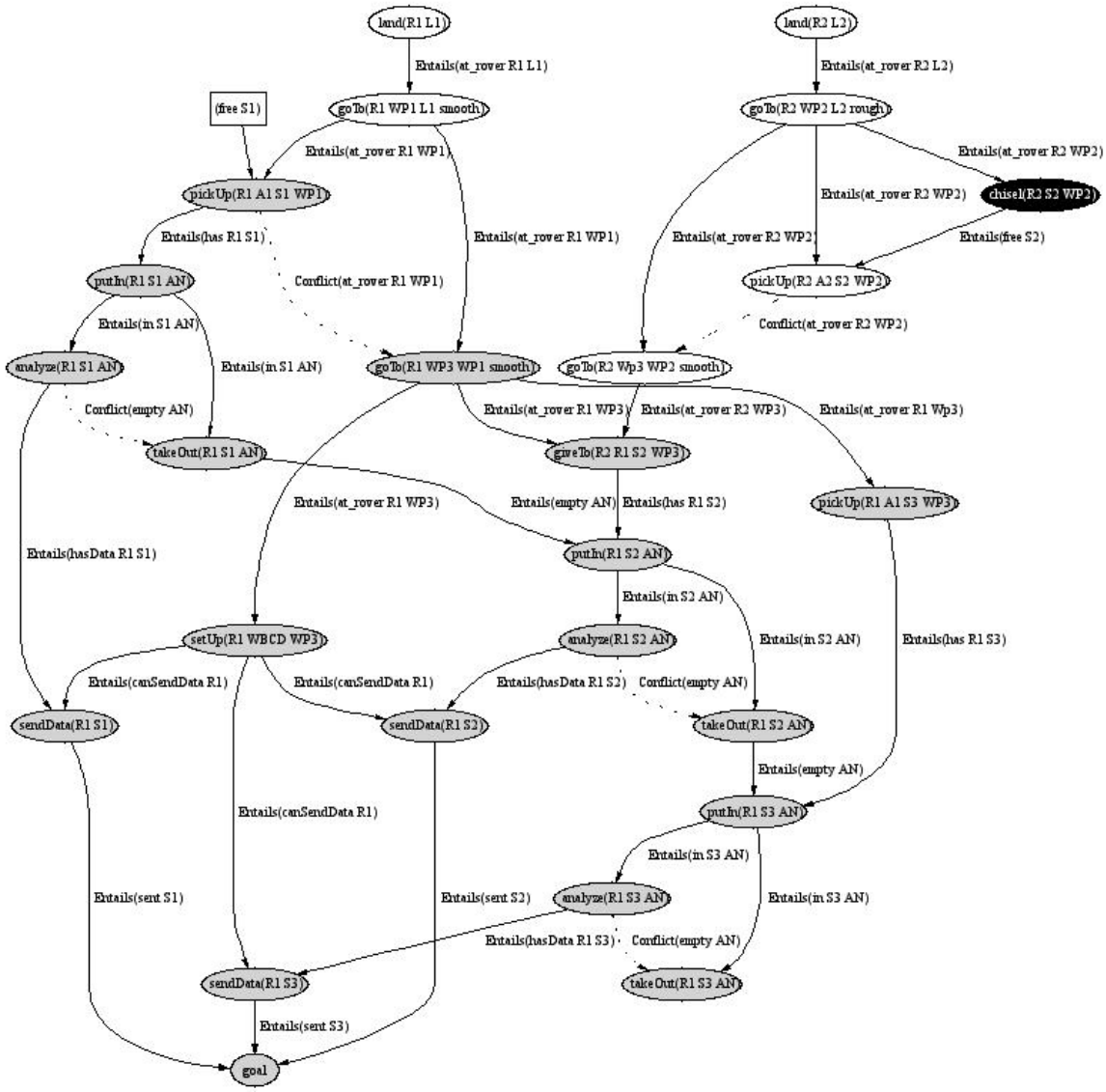


Figure 6.8. The shaded nodes represent the Rover 2's broken subplan for the changed condition "not(free S1)". The black node is a node that had previously been changed due to the changed condition "not(free S2)" as in the previous example 2.

Rover 2 goes through the process of making strongly linked activities and finding a new plan. The new plan for Rover 2 is shown in Figure 6.9.

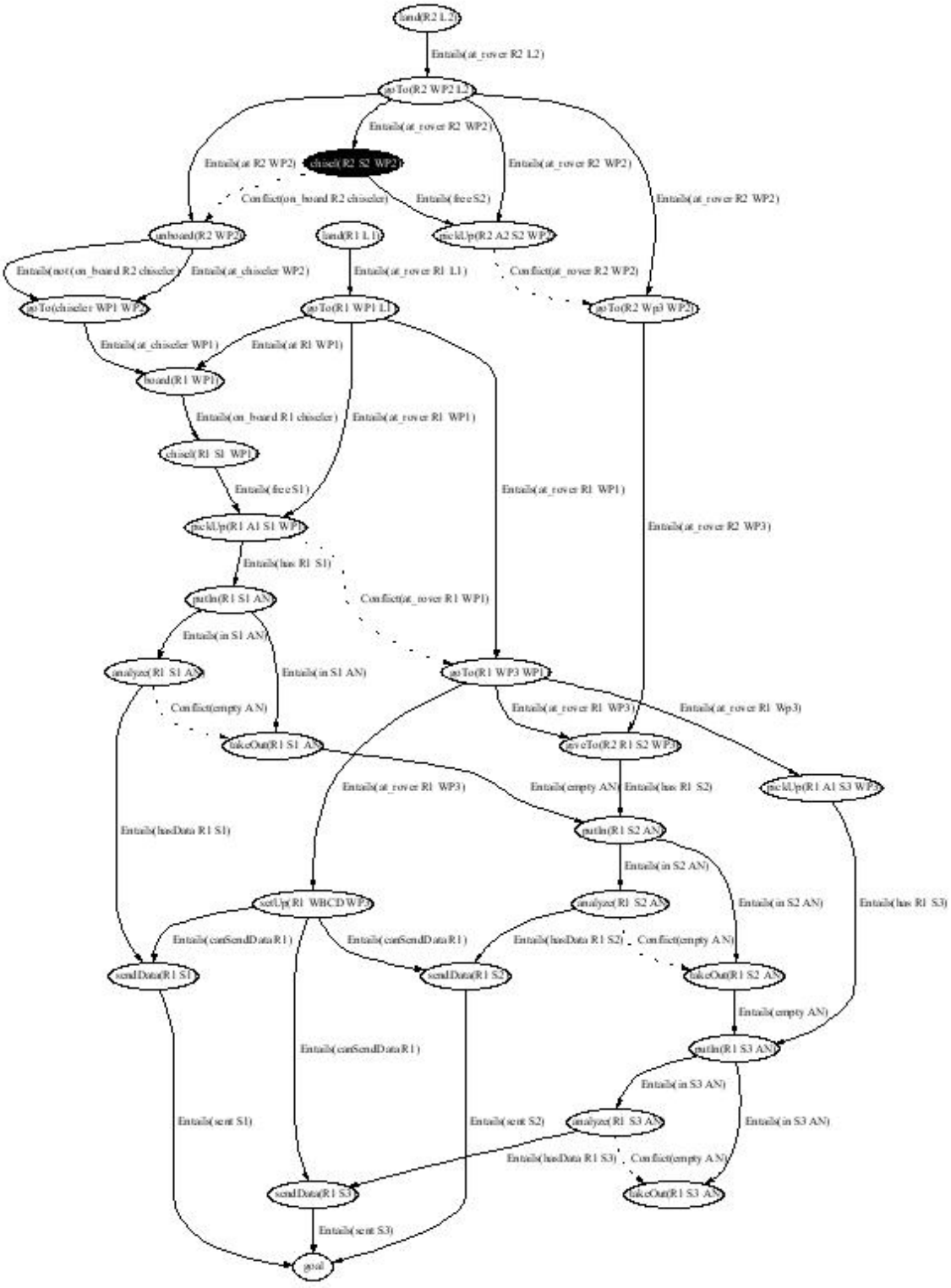


Figure 6.9. The new plan generated by Rover 2's planner for the changed condition "not(free S1)." Note that there is a conflict arc leaving the black node that had previously been changed. This node prevents Rover 1 from controlling the chiseler until Rover 2 is done with chiseling. Rover 1 needs to be aware of the sequence of actions.

Rover 2 notes that there is a conflict arc from the marked activity chisel(R2 S2 WP2) to newly added activity “unboard(R2 WP2)”. Rover 2 looks up in the RKB the initial conditions that support the activity “chisel(R2 S2 WP2)”. These are “not (free S2)”, “(on_board R2 chiseler)”, etc. It notes that “not (free S2)” is marked as changed. Rover 2 then sends Rover 1 the condition “not (free S2)”. The number of messages communicated has now increased to 11. Rover 1, backtracks to perform plan repair on the old plan with the condition “not(free S2)”. Since its old plan was the same as Rover 2’s, it goes through the same steps as Rover 2 in Figures 6.1 to 6.3. Then it goes through the same repair steps as Rover 2 in Figures 6.8 to 6.9 to arrive at the same plan as Rover 2.

In the no communication example, the changed condition had no effect on Rover 1, and thus, Rover 1 did not need to know. However, the changed condition “not(free S2)” had an effect on Rover 1’s changed plan in this example, so Rover 1 needed to know about the changes. Once Rover 1 knew what changes Rover 2 had made, it was able to come up with a plan that does not conflict with Rover 2’s plan.

# Chapter 7

# Discussion

## 7.1 Current Work

### 7.1.1 Implementation

The PIP algorithm is implemented in C++. An original plan was generated using the FF planner given an action domain and a problem file that included the set of initial conditions and the set of goals. Agents were created and given the original plan, a database associating initial conditions to owner agents, the action domain, and the problem file.

Each agent has a relational knowledge base, a plan analyzer, a translator, a planner, and a communicator.

*RKB*: the RKB is implemented as represented in Section 4.3.

*Plan Analyzer*: The plan analyzer performs the methods “find broken subplan,” “differentiate,” “initial condition checking,” and “find relevant agent set.”

*Unbroken Subplan Translator*: The translator can translate from a plan to strongly linked activities and back. The translator takes as input a set of activities, a set of causal links, an action domain, and a problem file that describe the initial conditions and the goals. For each activity, the translator makes a translation to a strongly linked activity operator and adds the operator to the action domain. The new goal is added to the problem file. The translator returns a mapping of the name of the strongly linked activity to the actual activity it represents.

To translate from a plan to a strongly linked activity, the translator takes the mapping created by the translation and replaces the names of strongly linked activities with the actual activity name.

*Planner*: The planner used for this implementation was FF. The action domain, initial conditions, and goals are inputs to the planner. The planner returns a plan that includes the existing plan and the new subplan.

*Communicator*: The communicator allows agents to communicate with other agents. The communicator acts as a messenger and sends information from one agent to the other. It is used when initial conditions need to be validated and when initial conditions need to be passed on.

### 7.1.2 Results

The PIP algorithm was used on the three examples presented in Chapters 5 and 6. The table below compares the number of messages exchanged using the PIP strategy to the number of messages exchanged using two other strategies: 1) a message is broadcast to all other agents each time a initial condition is changed and 2) a centralized planning agent collects all changed conditions and sends the new plan to the relevant agents, those who need to change their activities, rather than just the changed initial condition.

In the storyline set out in the introduction, there were 3 agents in the team; Rover 1, Rover 2, and the orbiter. However, there could be any number of agents in the team. For instance, there could be other rovers sent to explore a different area of the planet, which have tasks independent from the two rovers. There could also be other orbiters nearby that are part of the exploration team, and also have rovers capable of exploring planets but which do not send any to the three waypoints because they were covered by Rover 1 and Rover 2. With a growing number of agents in the team, the number of messages passed by the broadcasting method increases.

Table 7.1 contrasts the number of messages passed by each of the three methods. The first column shows an increasing number of agents in the team starting with the three that are in our rover scenario and increasing by two. Example 1 is where the changed conditions are “Waypoint 1 is rough terrain” and “Waypoint 1 is not smooth terrain.” Example 2 is where the changed condition is “not(free S2).” Example 3 is where the

changed condition “not(free S1)” is received after the changed condition “not(free S2).” The broadcast method and PIP are both instances where initial conditions are communicated, whereas the centralized method is an instance where actions are communicated. We will treat each action and each initial condition as one piece of information.

| Example | Number of Agents | Method | | |
|---|---|---|---|---|
| | | Broadcast | Centralized | PIP |
| **Example 1** | | | | |
| | 3 | 6 | 47 | 11 |
| | 5 | 10 | 47 | 11 |
| | 7 | 14 | 47 | 11 |
| | 9 | 18 | 47 | 11 |
| | 11 | 22 | 47 | 11 |
| | 13 | 26 | 47 | 11 |
| | 15 | 30 | 47 | 11 |
| **Example 2** | | | | |
| | 3 | 3 | 23 | 0 |
| | 5 | 5 | 23 | 0 |
| | 7 | 7 | 23 | 0 |
| | 9 | 9 | 23 | 0 |
| | 11 | 11 | 23 | 0 |
| | 13 | 13 | 23 | 0 |
| | 15 | 15 | 23 | 0 |
| **Example 3** | | | | |
| | 3 | 6 | 79 | 11 |
| | 5 | 10 | 79 | 11 |
| | 7 | 14 | 79 | 11 |
| | 9 | 18 | 79 | 11 |
| | 11 | 22 | 79 | 11 |
| | 13 | 26 | 79 | 11 |
| | 15 | 30 | 79 | 11 |

Table 7.1. Summarized Results comparing the three methods.

As can be seen, the number of messages communicated in both the centralized strategy and the PIP strategy do not increase with the number of agents since messages

are only sent to the agents who need to know about the changes. As can be seen, the distributed PIP method performs better than a centralized method. However, the broadcast method performs better than PIP in the examples where the number of rovers in the team is small and the changed condition changes activities with a large number of supporting initial conditions. The number of initial conditions in the broadcast method is multiplied with the numbers of rovers in the team. Because of this multiplying effect, PIP performs better than the broadcast method in teams where the number of rovers is large.

PIP performs best when agents are not very dependent upon each other so that as in Example 2, changes to the plans require almost no communication. It offers benefits when the team is very large but only a few rovers are mutually dependent upon each other.

## 7.2 Future Work

### 7.2.1 Activities with Durations

In this thesis, activities are assumed to be atomic actions, all of which take the same amount of time. For more applications, activities need to be modeled as having some duration. These durations are given metric values representing the duration of the activity.

By adding durations, the agents must not only plan the order of activities, but determine if they are schedulable given these metric constraints. Scheduling problems are

usually solved by transforming them into constraint satisfaction, linear programming, or network flow problems. The PIP algorithm will need to be expanded so that agents pass not only changed initial conditions, but also changed time constraints to ensure plan agreement. Future research should examine how to extend this algorithm to include numerical time constraints. Key issues include determining how to pass numerical time constraints and still ensure temporal consistency.

The planner used will also need to be modified to handle time constraints. The 2003 AIPS competition extended the PDDL language to handle activities with durations [7]. Many planners at the competition were able to handle plans with temporal activities [8][9][12][17].

### 7.2.2 Hierarchical Decomposition

Plans can grow to be extremely large and complicated if every low level detail that an agent has to perform were to be included in the plan. In a worst case scenario, an initial condition that has changed will require the agent to repair the entire plan. The complexity of the problem grows as the number of actions and vehicles involved increases. Plans with greater complexity will take a longer time to solve.

One solution is to introduce a hierarchical structure to the actions. The hierarchical structure would consist of levels, each level containing macro activities that can be broken down into lower level activities within the next lower level. The bottom

most level would consist of primitive activities that cannot be broken down any further. For example, the action “pick-up” could further be decomposed into “agent-bend-over”, “grasp-object”, “agent-straighten-up”, and “put-object-in-storage”. Rather than include all these steps in the plan, the planner simplifies the problem by representing the sequence of actions as “pick-up”. Whenever the planner includes the “pick-up” action in a plan, it will then decompose the pick-up action into its separate components.

Consequently, the planner can repair the plan at the level which the initial condition that has changed affects the plan. The planner would first try to repair the plan at the macro level that encapsulates the affected level. Rather than making the goals of the repair the goals of the entire mission, the goals are now the effects of the macro. The initial conditions are the preconditions of the macro. If the planner fails at the first macro level, the algorithm would then move to the next level in the hierarchy.

Many practical planners today make use of a hierarchical task structure [3][5][14]. Future work would incorporate hierarchical decomposition to increase efficiency in the algorithm. Key issues include how to ensure that the activities within the macros do not conflict, and how to decompose activities in a way that decreases dependency between agents and reduces communication.

## 7.3 Summary

Cooperative distributed robotic systems can reduce communication by trading off computation. In this thesis, agents who discover discrepancies between their beliefs and

the true state communicate to agree on a new shared plan. The agents communicate on a need-to-know basis and only communicate initial conditions as opposed to plan activities and plan states.

The PIP algorithm changes as few activities as possible by performing plan repair only on the broken subplan, thereby, affecting and consequently requiring communication with fewer agents. In order to allow the use of a traditional planner to generate local subplans that are consistent with global constraints, strongly linked activities are introduced. Differences between the new plan and the old plan help identify which agents need to know about the changed initial condition.

The results show that the algorithm offers the most benefit in large cooperative teams where agents are not highly dependent. Future work includes extending the algorithm to handle richer durative activities and to increase efficiency by introducing hierarchical decomposition.

References

[1] AT&T GraphViz http://www.graphviz.org

[2] A.Blum & M.Furst. Fast Planning Through Planning Graph Analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95),* 1995

[3] B. Clement & E. Durfee. Top-Down Search for Coordinating the Hierarchical Plans for Multiple Agent. In *Proceedings of the Third International Conference on Autonomous Agents,* 1999.

[4] Cormen, Leiserson, and Rivest. *Introduction to Algorithms.* MIT Press, 1995.

[5] M. E. desJardins, E.H. Durfee, C.L. Ortiz, and M.J. Wolverton. A Survey of Research in Distributed Continual Planning. *AI Magazine*, 1(4):13-22, 1999.

[6] E. Durfee, V. Lesser, & D. Corkill. Trends in Cooperative Distributed Problem Solving. *IEEE Transactions on Knowledge and Data Engineering.* 1995.

[7] M. Fox & D. Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains, Technical Report, Department of Computer Science, University of Durham, 2001.

[8] A. Gerevini & I. Serina. LPG: a Planner based on Local Search for Planning Graphs. *Proceedings of the Sixth International Conference on AI Planning and Scheduling.* AAAI Press, 2002.

[9] Keith Halsey. Temporal Planning with a Non-Temporal Planner. In *Proceedings of the Seventh International Conference on AI Planning and Scheduling*. 2003.

[10] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. RoboCup: The Robot World Cup Initiative. In *Proceedings of the First International Conference on Autonomous Agents,* 1995.

[11] Phil Kim, Brian Williams, and Mark Abramson. Executing reactive, model-based programs through graph-based temporal planning. In *Proceedings of IJCAI-2001, Seattle,* WA, 2001.

[12] D. Long & M. Fox. Fast Temporal Planning in a Graphplan Framework. *Proceedings from the Sixth International Conference on Artifical Intelligence Planning and Scheduling.* 2002.

[13] D. McDermott & the AIPS'98 Planning Committee. PDDL- The planning Domain Definition Language. Technical Reportm Department of Computer Science, Yale University. 1998.

[14] D. Nau, Y. Cao, A. Lotem, H. Munoz-Avila. SHOP: Simple Hierarchical Ordered Planner. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 95)*, 1999

[15] J. Rosenschein. & M. Genesereth. Communication and Cooperation Among Logic-Based Agents. *IEEE*, 1987.

[16] S.Russel, P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[17] B. Selman, H. Kautz, and B. Cohen. "Local Search Strategies for Satisfiability Testing." *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, 1993.

[18] D. Smith, J. Frank , & A. Jonsson. Bridging the Gap Between Planning and Scheduling. *Nasa Ames Research Center*, 1997.

[19] R.G. Smith. The Contract Net Protocol: High-level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, 1980.

[20] D. Weld. Recent Advances in AI Planning. *AI Magazine, 1999.*

[21] M. Wolverton and M. desJardins. Controlling Communication in Distributed Planning Using Irrelevance Reasoning. *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. AAAI-1998.

[22] K.S. Decker and V.R. Lesser. Designing a Family of Coordination Algorithms. In *Proceedings of the First International Conference on Multi-Agent Systems*. 1995.

# Appendix A
## Rovers Exploration Action Domain in PDDL

```
(:action land
        :parameters (?r rover ?l location)
        :condition (and (on ship ?r)
                        (land_location ?r ?l))
        :effect    (and (not(on ship ?r))
                        (at ?r ?l))
)

(:action goto
        :parameters (?r rover ?new_l location ?old_l location ?t – terrain type)
        :condition  (and    (at ?r ?old_l)
                        (is_type ?l ?t)
                        (can_traverse ?r ?t))
        :effect    (and (not(at ?r ?old_l))
                        (at ?r ?new_l))
)

(:action pickup
        :parameters (?r - rover ?l - location ?s - specimen ? ?a – arm)
        condition (and      (at ?r ?l)
                        (at ?s ?l)
                        (on_board ?r ?a)
                        (free ?s))
        :effect    (and      (has ?r ?s))
)

(:action putin
        :parameters (?r – rover ?a – analyzer ?s – specimen)
        :condition  (and    (on_board ?r ?a)
                        (empty ?a)
                        (has ?r ?s))
        :effect    (and      (not (empty ?a))
                        (in ?s ?a))
)

(:action analyze
        :parameters (?r - rover ?a - analyzer ?s - specimen)
        :condition (and     (has ?r ?s)
                        (on_board ?r ?a)
                        (in ?s ?a))
        :effect    (and      (analyzed ?s))
)

(:action takeout
        :parameters (?r – rover ?a – analyzer ?s – specimen)
        :condition (and     (in ?s ?a)
                        (on_board ?r ?a)
                        (not (empty ?a)))
        :effect (and        (not (in ?s ?a))
                        (empty ?a))
)
```

```
(:action giveTo
        :parameters (?r1 – rover ?r2 – rover ?o – object ?l – location)
        :condition  (and   (has ?r1 ?o)
                           ( not(has ?r2 ?o))
                           (at ?r1 ?l)
                           (at ?r2 ?l))
        :effect (and       (not(has ?r1 ?o))
                           (has ?r2 ?o))
)

(:action setup
        :parameters (?r – rover ?d – device ?l – location)
        :condition  (and   (at ?r ?l)
                           (on_board ?r ?d))
        :effect    (and    (ready ?d))
)

(:action sendData
        :parameters (?r – rover ?d – data)
        :condition  (and   (analyzed ?d)
                           (ready WBCD))
        :effect    (and    (sent ?d))
)

(:action chisel
        :parameters (?r – rover ?s – specimen ?l – location)
        :condition  (and   (not(free ?s))
                           (on_board ?r chisel)
                           (at ?r ?l)
                           (at ?s ?l))
        :effect  (and      (free ?s))
)
```

Appendix B
Problem File

Initial conditions
(on_ship R1)
(on_ship R2)
(on_board R1 A1)
(on_board R2 A2)
(on_board R2 chiseler)
(on_board R1 analyzer)
(on_board R1 WBCD)
(land_location R1 L1)
(land_location R2 L2)
(empty AN)
(is_type WP1 smooth)
(is_type WP2 rough)
(is_type WP3 smooth)
(at_specimen S1 WP1)
(at_specimen S2 WP2)
(at_specimen S3 WP3)
(free S1)
(free S2)
(free S3)
(can_traverse R1 smooth)
(can_traverse R2 smooth)
(can_traverse R2 rough)

Goals
(sent S1)
(sent S2)
(sent S3)