Plan Recognition in Open Worlds

Karen E. Huff and Victor R. Lesser

COINS Technical Report 88-18

Revised December 1988

Abstract: Many applications of plan recognition involve *open worlds*, where information about the state of the world is incomplete. Without this information, competing interpretations cannot be disambiguated, predictions of future actions lack precision, and certain errors go undetected. One solution to acquiring additional state information is to use domain knowledge to make plausible assumptions about the missing values using the observable values. We show how a plan recognition architecture (based on the hierarchical planning paradigm) can be extended to incorporate a new type of domain knowledge: knowledge that is imperfect, supporting assumptions that may have to be revised. *Credibility*, the degree of agreement between an interpretation and current *assumptions* about the world, provides the basis for improved disambiguation, prediction, and error detection. The additional discrimination power is flexible—an interpretation that violates some current assumptions can be pursued after *reconciliation,* the process of revising assumptions to bring the world state into conformance with the interpretation. This extension to plan recognition exploits deeper domain knowledge about the context for actions, enabling deeper modeling of open worlds.

This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB, DC 20332 under contract No. F30602-C-0008. This contract supports the Northeast Artificial Intelligence Consortium (NAIC).

1.0 Introduction

Plan recognition has proved a useful paradigm for a variety of applications, such as automated consultation [7], natural language discourse [1, 13], story understanding [16], explanation-based learning [5], and interpreting behavior of adversaries [2]. Many of these applications involve *open worlds*, where information about the state of the world is incomplete. When it is impossible (or infeasible) to acquire the missing information, plan recognition is adversely affected. Competing interpretations of actions cannot be disambiguated, some distinctions between legal and illegal actions/plans cannot be made, predictions of future actions lack precision, and there is no basis to explain why observed actions/plans were chosen over other alternatives.

As an example, consider a recognition-based intelligent assistant that supports a programmer carrying out the process of software development [8]. The observable actions consist of tool invocations, and the observable state of the world is reflected in the software objects (source code, testcases, etc.) that exist and the relationships among them. One goal might involve building a new system version; in turn, that goal might be decomposed into creating source modules, compiling/linking, testing and finally archiving. To test a system means running all applicable testcases, and only those cases (neither under- nor over-testing is desirable). The problem is that applicability of testcases cannot be directly observed as a result of past actions nor computed with certainty from observable data. When a testcase is run, there is no *independent* basis for determining if the testcase is indeed applicable, or if this signals the end of testing. Knowledge about testing strategies (where a given strategy implies that certain categories of testcases are applicable) underlies the choice of testcases, but this deeper knowledge cannot be exploited since the operative testing strategy cannot be determined—it too is not observable.

Predicates such as *applicable(<testcase>,<system version>)* whose truth or falsity cannot be determined cannot be used in operator definitions to define the relevancy of an operator (preconditions), the decomposition and completion criteria (subgoals), or restrictions on parameter bindings (constraints). When such predicates are omitted entirely, the operator definitions are under-constrained and plan recognition is too permissive—"illegal" plans will be accepted, predictions will be too general, and alternatives that are actually irrelevant cannot be discarded. Attempting to compensate by substituting another expression for the missing predicate may yield a definition that is over-constrained and plan recognition will be too rigid.

One solution to acquiring additional state information is to use the observable state information to make plausible assumptions about the missing values. In the testing example, there is a correlation between the types of changes made during source editing (something that can be measured) and the operative testing strategy, which in turn determines the categories of tests that are applicable. For example, when changes are simple (affecting a few lines of code), a weak test strategy would typically be appropriate—only base testcases would be applicable. Otherwise, standard testing would typically be appropriate—base and normal testcases would be applicable. This reasoning is inherently non-monotonic; assumptions are made in the absence of information to the contrary, while later, additional information may be acquired that defeats the earlier conclusion and its consequences.

When assumptions about test strategy and applicability are added to the observable state of the world, it is possible to evaluate the *credibility* of alternatives. Interpretations of actions (or predictions of future actions) that agree with the current assumptions have the highest credibility; interpretations lose credibility with each assumption that is violated. If the operative testing strategy is assumed to be standard testing, then an action to run one of the normal cases is fully credible, because that case is assumed to be applicable. On the other hand, an action to archive the system, which is predicated upon testing being completed, would have less than full credibility as long as there are applicable (i.e., base or normal) cases still to be run.

Given two alternative interpretations that differ in credibility, the more credible alternative is more likely to be the correct interpretation. Given two choices for completing an unsatisfied subgoal, the more credible alternative is the better prediction of the future. An action whose "best" interpretation is below a certain credibility threshold is a possible user error. Thus, the programmer can be advised of a possible oversight when archiving prior to running all applicable testcases. Finally, it is possible to give the underlying reason for the credibility of running or not running a particular testcase, by citing the operative testing strategy and its implications.

Credibility can be combined with other discriminators to determine which interpretations to pursue. Sometimes it will be necessary or desirable to pursue an interpretation that is not fully credible; for example, the interpretation may still be the "best" considering all available discrimination information. In order to proceed with such an interpretation, it is necessary to *reconcile* the assumptions about the state of the world with the requirements of the desired interpretation. For example, suppose the operative testing strategy is assumed to be standard testing. To pursue an interpretation in which archiving starts when only base cases have been run, it is necessary to revise the assumption that testing is not done. While this can trivially be accomplished by simply recording that testing is done, it is far more interesting to provide some rationale for testing being done. And, indeed, the preferred *reconciliation* is to change the operative testing strategy from standard to weak testing, after which it follows that testing is now done. The full implementation of this example is given later.

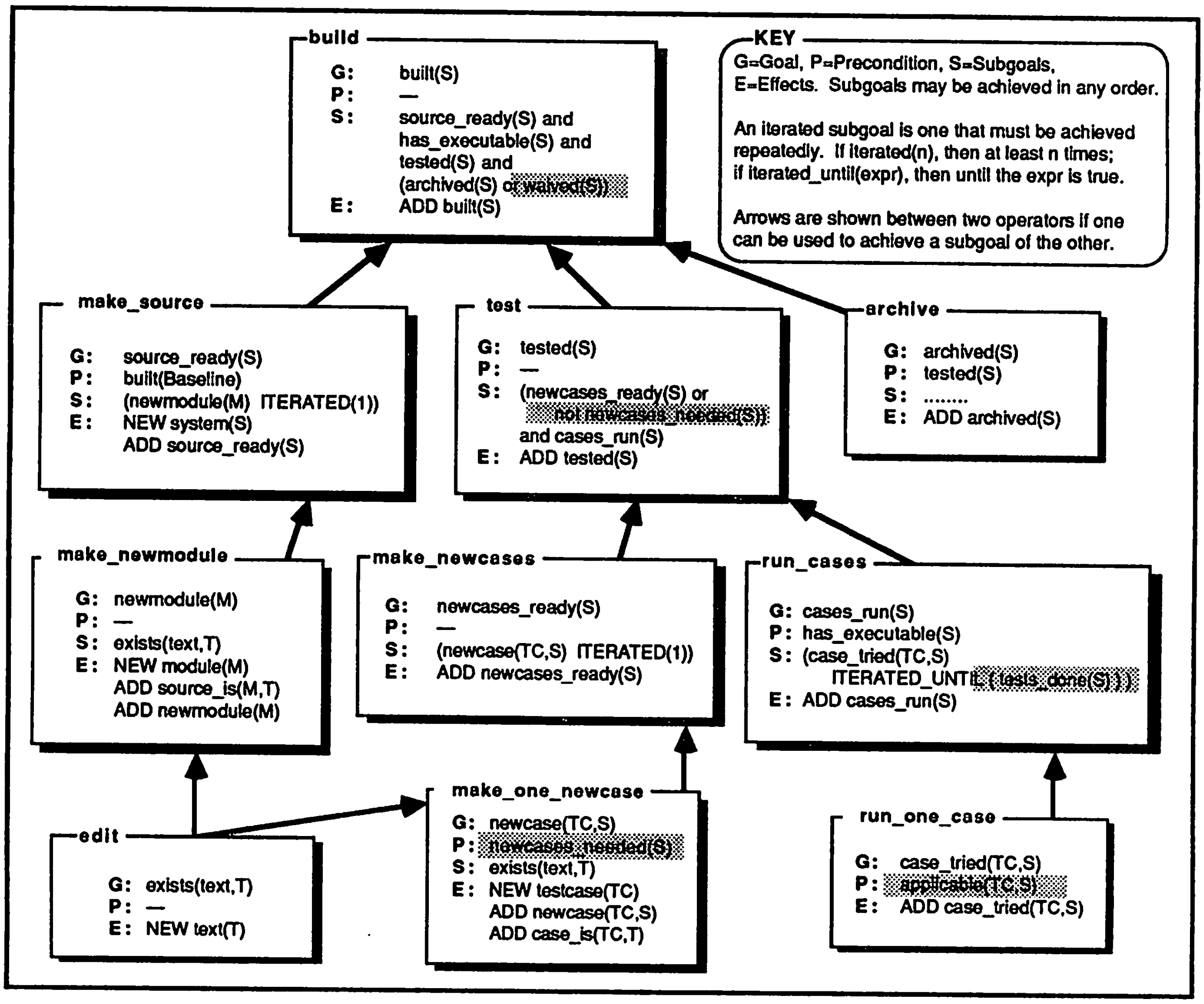
In this paper, we describe how a plan recognition system, based on the classical hierarchical planning paradigm, can be extended to incorporate a new type of domain knowledge about the context for actions; unlike the knowledge already reflected in operator definitions, this knowledge is approximate rather than absolute. This approach allows plausible inference and plausible explanation within a deeper model of domain activities than is otherwise possible. In Section 2, we show how to capture this deeper knowledge, and how to exploit it for reasoning about world state. This is accomplished through monotonic and non-monotonic rules in a truth maintenance system (TMS). In Section 3, we explore the impact on a plan recognition architecture. Credibility represents a new perspective from which competing interpretations can be disambiguated. Reconciliation is the means by which assumptions are revised when that is necessary. We end with a brief summary.

2.0 Reasoning About the State of the World

Reasoning about the state of the world is a potentially complex activity. It includes gathering evidence from many sources, refining this evidence by applying additional domain-specific knowledge, and reaching plausible assumptions as well as hard conclusions. For the predicate *applicable*, there are two disparate sources of evidence (the nature of changes made during editing and the categories to which individual testcases belong); additional domain-specific knowledge, defining test strategies and relevancy of testcase categories, is needed to put this evidence to actual use. It is difficult to see how the knowledge about testcase applicability can be parceled out among operator definitions (which are the standard vehicles for expressing domain knowledge). A separate rule system is needed, allowing all the new domain knowledge, detailing what evidence to gather as well as how to refine the evidence, to be collected in one place, so that related rules can be grouped together. In order to meet the requirement for making assumptions, the rule system should be non-monotonic.

A truth maintenance system (TMS) [6] is one approach to implementing non-monotonic reasoning, based on multi-valued logic. A TMS maintains a network of *nodes*, each of which can be labelled IN or OUT. Separate nodes are used for a predicate and its negation. If the node for a predicate is IN and the node for its negation is OUT, the predicate is *true*; if the node is OUT and the negation is IN, the predicate is *false*. If both are OUT, the truth value is *unknown*; if both are IN, there is a *contradiction*. *Justifications* capture the

Figure 1: Example Operators



relationships between the nodes, correlating a set of *support* nodes and a set of *exception* nodes with a *conclusion* node. A justification of the form A EXCEPT B —> C means that if A is IN and B is OUT then C is IN. The exception node B represents the non-monotonic content of the justification; a monotonic justification has an empty list of exceptions. In order for a node to be IN, it must have at least one *valid* justification; a justification is valid if all its support nodes are IN and all its exception nodes are OUT. A *premise* justification has empty support and exception lists, so it is always valid.

2.1 TMS Justifications

As an example, take the selection of operators in Figure 1 for building a system version. These operator definitions follow the standard hierarchical planning paradigm [12, 14, 15]; their effects determine the *core state* of observable facts. The use of the *extended state* predicates that cannot be observed is highlighted. For example, the precondition in *run_one_case* requires that the testcase be *applicable;* the iteration completion criteria in *run_cases* is that *tests_done* be true; and, there is a subgoal in *build* that will allow archiving to be skipped when waiving it is appropriate. The objective is to use TMS justifications to derive truth values for these extended state predicates.

Example domain knowledge about *applicable* and *tests_done* is given in justification form in Figure 2. In order to express this knowledge, several additional predicates have been introduced. Changes made during editing are used to conjecture whether the test strategy should be standard or not (rules J1-J2). The non-monotonic rule J1 can be read "If substantive changes are made during editing, then *typically* a standard testing strategy is appropriate." Testing strategy determines whether normal testcases are *relevant* (rules J4-J5); base cases are always *relevant* (rule J3). Rules J6-J7 establish that cases that are *relevant* are *applicable* unless they are *specifically_excluded* (as they would be if infected by some catastrophic bug that is not yet fixed). Rule J8 states that typically cases are not *specifically_excluded. Tests_done* does not hold until all applicable testcases have been tried (J9-J10). Note that both monotonic and non-monotonic rules have been used; the meaning of the testing strategies is defined monotonically, for example.

Figure 2: Justifications

```
Operative Test Strategy
    J1: substantive_changes(Sys) EXCEPT not standard_test(Sys) —> standard_test(Sys)
    J2: EXCEPT standard_test(Sys) —> not standard_test(Sys)       %weak testing = not standard_test

Relevancy of Cases, By Category
    J3: type(Case,base) —> relevant(Case,Sys)
    J4: type(Case,normal) and not standard_test(Sys) —>  not relevant(Case,Sys)
    J5: type(Case,normal) and standard_test(Sys) —> relevant(Case,Sys)

Applicability of Cases
    J6: relevant(Case,Sys) and not spec_excluded(Case,Sys) —> applicable(Case,Sys)
    J7: EXCEPT applicable(Case,Sys) —> not applicable(Case,Sys)

    J8: EXCEPT spec_excluded(Case,Sys) —> not spec_excluded(Case,Sys)

Completion of Testing
    J9: applicable(Case,Sys) and not case_tried(Case,Sys) —> not tests_done(Sys)
    J10: EXCEPT not tests_done(Sys) —> tests_done(Sys)
```

2.2 TMS Node Labels

Justifications provide a way to derive the extended state from the core state. When the justifications are instantiated, and the truth values of core state predicates entered (with premise justifications), the truth maintenance process will label the nodes, giving a read-out on the truth values of the extended state predicates. When the core state changes as a result of an action, the nodes will be relabelled and the extended state predicates may change*; in this way, the extended state predicates may vary over time**. The TMS will also determine whether the truth value of a predicate is *certain* or *by-assumption.* A predicate is certain unless one or more non-monotonic rules were used to determine its truth value. (The truth values of predicates that are certain cannot be changed in order to reconcile an interpretation.)

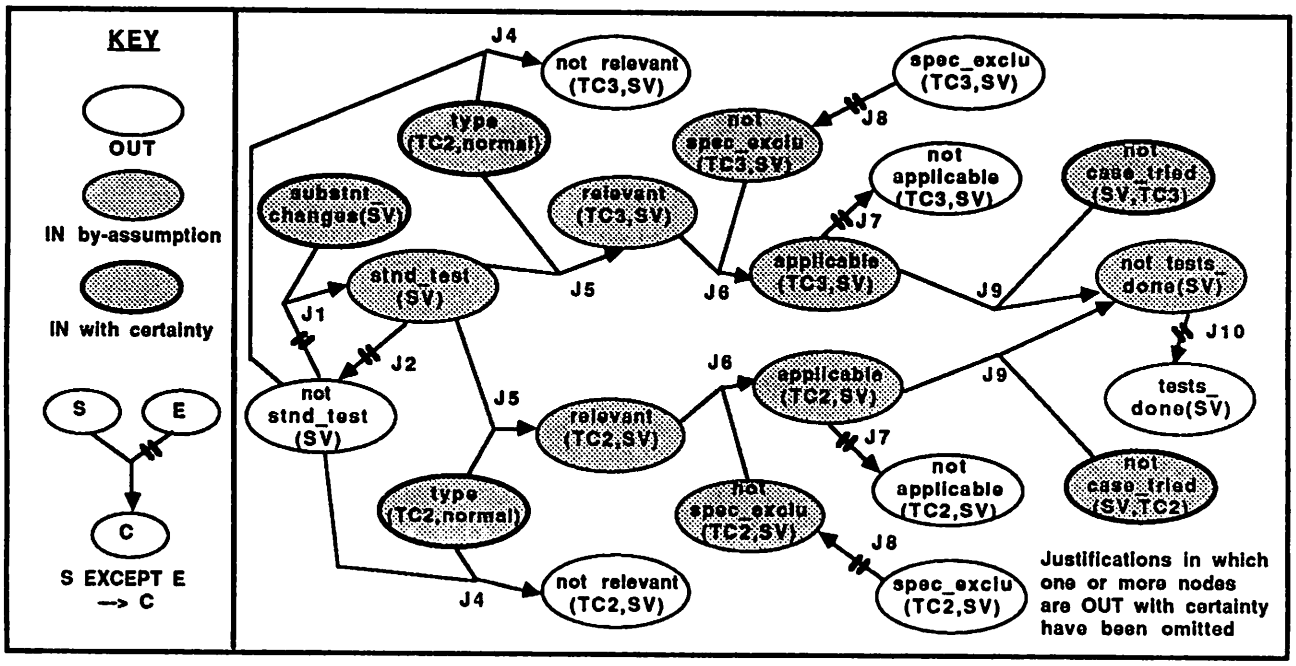
As an example, consider a situation where the building of system version SV has progressed to the point where testing is in progress; suppose also that the source editing changes were substantive. Let there be three testcases, where TC1 is a base case, and TC2 and TC3 are normal cases; suppose TC1 has been run. The state of instantiated justifications is given in Figure 3, showing that standard testing is assumed to be operative,

---

* A preference facility is used to distinguish stronger rules from weaker ones; as the state changes, preferences ensure that weak evidence does not block stronger evidence supporting the opposite conclusion. Preferences address the issue of multiple labellings in a TMS with even loops [4]. Rules such as J1 and J2 (which are in normal form [11], to borrow a term from default logic [10]), cause even loops. Preferences would be used, for example, to ensure that if J2 currently supports *not standard_test(x)* when *substantive_changes(x)* becomes true, J2 will be made invalid and J1 valid.

** One might want to sanction certain inferences about system Y (that is in some way related to system X) only during the time that X is the current customer release; a rule to do this would include *curr_cust_rel(X)* as one of its hypotheses.

Figure 3: Instantiated Justifications



that TC2 and TC3 are assumed to be applicable in testing SV, and that testing is not done as there are (two) applicable cases that have not yet been run.

Normally in plan recognition, predicates describing the world state evaluate to true or false, the evaluation is known to be certain, and interpretations are either valid or invalid. With the introduction of TMS justifications, there is a middle ground between valid and not valid—described by degree of credibility. In the next section we discuss how a plan recognizer can capitalize on this new information.

3.0 Impact on Plan Recognition Architecture

Effective plan recognition involves making timely and knowledgeable choices between competing interpretations. Deferring decisions, thereby allowing further actions to narrow the field, restricts the ability to make inferences about the current situation. Making arbitrary choices is computationally expensive since they often have to be re-visited. In [9], the domain-independent assumption that the preferred interpretation contains a minimal number of top-level plans is used to make reasoned choices in the presence of uncertainty. In [3], both domain-independent and domain-dependent heuristics could be expressed and used to guide choices. Credibility represents an additional source of discrimination knowledge, as described below.

3.1 Use of Credibility

As an example of the use of credibility, consider the following scenario. Let building of system version SV be in progress as described in Figure 3, where substantive changes were made during source editing, standard testing is assumed operative, cases TC1-TC3

Figure 4: Plan in Progress

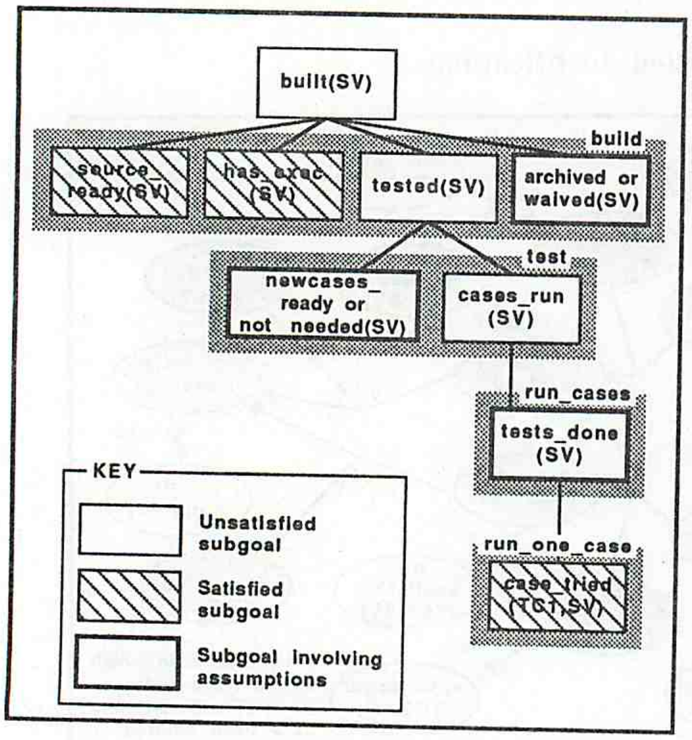
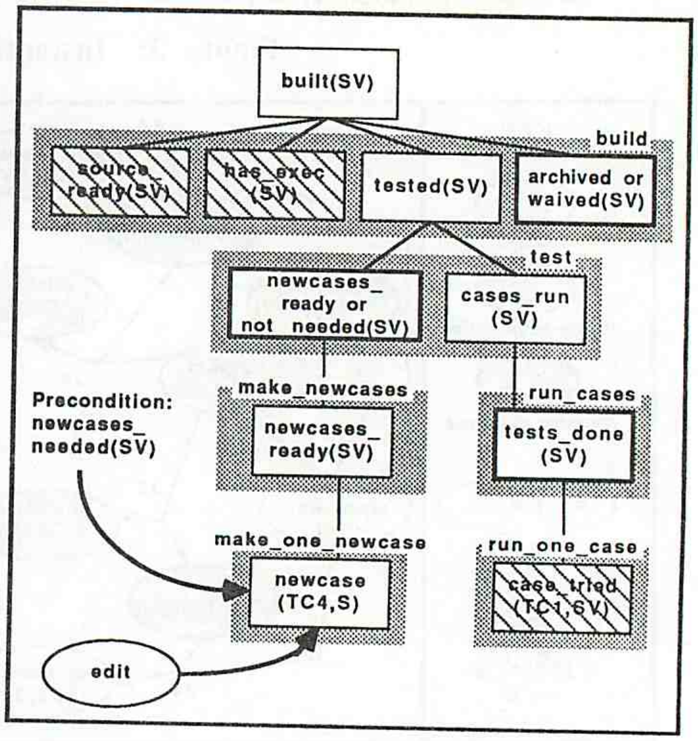


Figure 5: Edit as Make_one_newcase



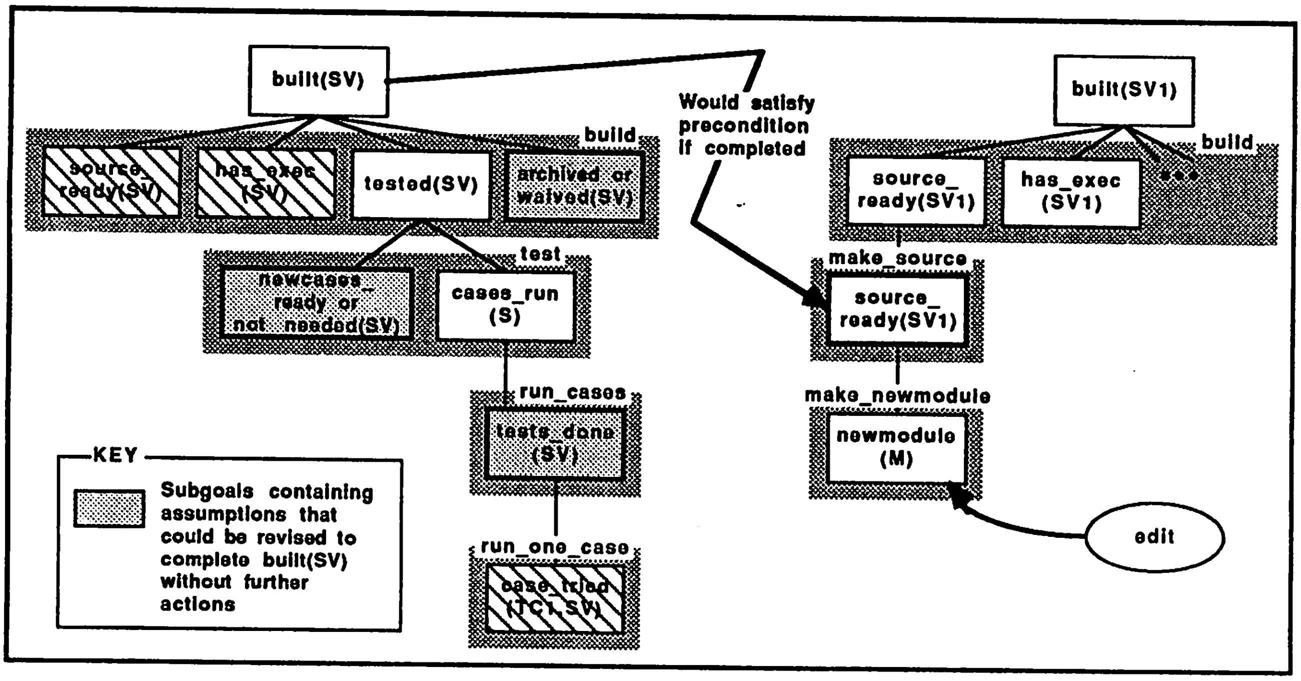
are assumed applicable, and only case TC1 has been run. Further, let there be assumptions that new testcases are needed, and that archiving is not waived. The state of the plan for building SV is shown in Figure 4. Let the next action be *edit*. Given the operator library of Figure 1, there are two interpretations for *edit*: editing to make a new testcase and editing to get source code ready.

Consider the interpretation of *edit* as part of making new testcases; in this interpretation (Figure 5), *edit* continues the work of building system SV. *Edit* itself has no preconditions, but it inherits the precondition *newcases_needed* from *make_one_newcase*. Since *newcases_needed* is true by assumption, this interpretation is fully credible—it depends on (one) extended state predicate that is assumed to be satisfied. Note that if *newcases_needed* had been true with certainty, then the interpretation would be valid absolutely.

Now consider the interpretation of *edit* as part of getting source code ready; in this interpretation, *edit* starts a new top-level plan (shown to the right in Figure 6). Again, *edit* has no preconditions, and *make_newmodule* has no preconditions, but *make_source* has a precondition that the baseline system (on which this new system version is to be based) is built. *Built(SV)* is not true, but there is a plan to achieve it that is in progress. Three assumptions enter into believing that this plan is not finished*. They are that new testcases

* The algorithm that discovers this is simply a planning algorithm—in general, it may be necessary to choose and instantiate complex operators, bind variables, and the like. Here, the planner must find a plan that can be considered satisfied (not one that calls for overt actions to be taken). In that context, making assumptions about the world state is an important means of completing a plan. Including a planner within a plan recognizer has other advantages, such as the ability to handle missing observations of actions and to predict future actions.

Figure 6: Edit as Make_newmodule



are needed (affecting the first subgoal of *test*), that testing is not done (affecting the iterated subgoal of *run_cases*), and that archiving is not waived (affecting the last subgoal of *build*). Thus, the interpretation of *edit* as part of starting a new *build* plan has low credibility— it conflicts with *three* current assumptions.

Credibility is a basis for distinguishing the relative likelihood of these two competing interpretations, establishing a clear preference for edit as making testcases. It so happens that discrimination based on the domain-independent heuristic preferring interpretations with the minimal number of top-level plans reinforces this preference, although in other cases there will be conflicts between credibility and the minimal-plans criterion. The best discrimination decisions will result from combining the evidence from multiple, independent perspectives. Credibility represents a new perspective, derived from deeper modeling of the context of actions.

3.2 Reconciliation

Reconciliation is the process of revising assumptions to make world state conform to the requirements of an interpretation. This is only necessary when the "best" alternative (considering all available discriminators) still violates a few assumptions about the state of the world, or when other more attractive alternatives were originally chosen but led to contradictions. Reconciliation can be trivially accomplished by adopting the desired assumptions; but then, clues that other assumptions are wrong will be ignored. Since the justifications provide the accepted rationale for various assumptions, it is better to find a rationale for the desired assumption than to adopt it outright. We first describe an algorithm for reconciliation, and then give an example.

The reconciliation algorithm is based on dependency-directed backtracking for TMS's [6], although different in several respects. Originally, TMS's were used both to record chains of inference and to record the dependency of specific problem-solving decisions on those inferences. A difficulty, reported by various researchers, was that when it was discovered that a particular decision led to a contradiction, there was no way to make an *informed* choice about which assumption(s), of the many that entered into the decision, to retract. In contrast, we use the TMS only to record the inference chains leading to the extended state predicates; dependencies between these predicates and a particular interpretation are tracked by the plan recognizer. Thus, when reconciliation is required, the particular assumptions that need revising have already been identified. While this does not guarantee that there is a unique way to achieve reconciliation, in practice there are very few viable choices, and these can usually be disambiguated by a simple domain-independent heuristic.

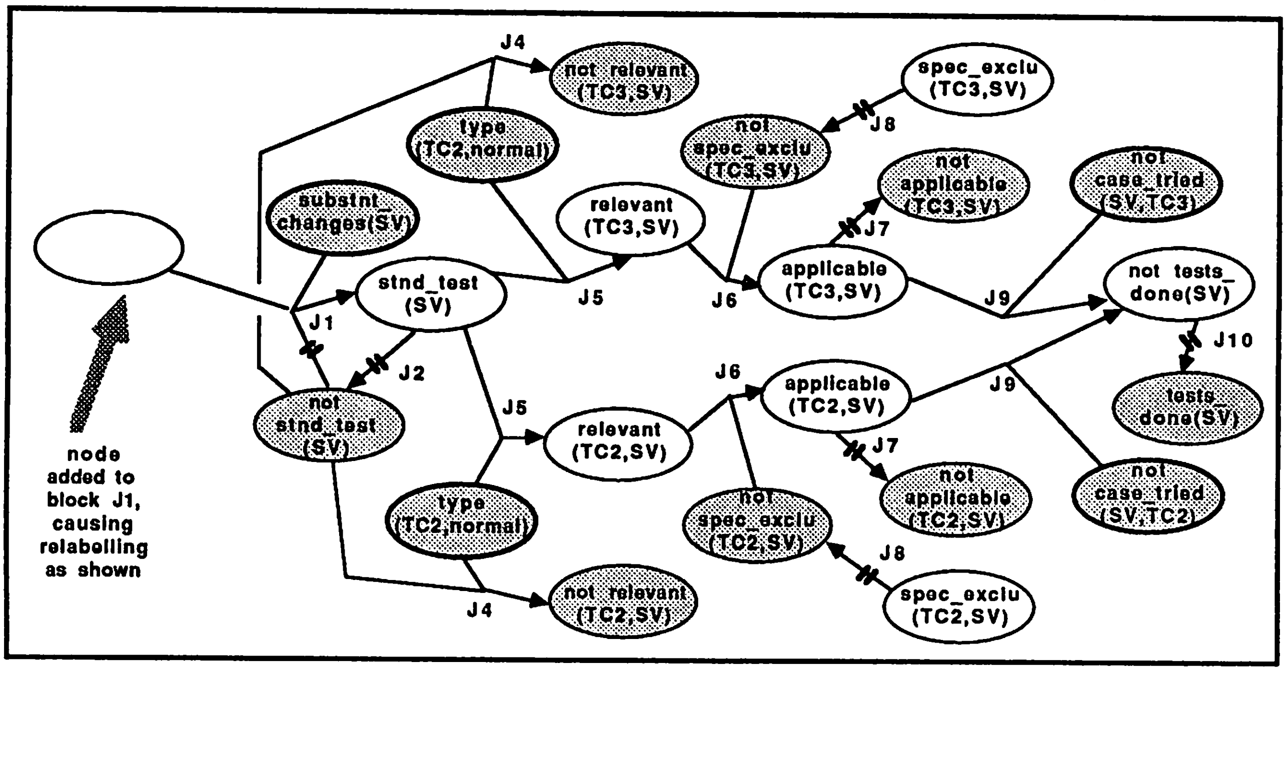
The reconciliation process begins with the identification of all alternatives for bringing a list of nodes IN; this involves backtracking through the inference chains, moving from a conclusion node via a particular justification to its support and exception nodes. Two types of atomic changes are allowed when one can backtrack no further: adding a "dummy" justification to bring IN a node that is OUT and blocking a valid non-monotonic justification to force OUT a node that is IN. Alternatives are ranked according to the number of changes required; the alternative with the minimum number of changes is selected (this is the simple heuristic). Thus, if changing a single node will bring two desired nodes IN, that solution is preferred to any alternative requiring two or more node changes. Because the analysis of possible changes does not consider interactions, the "best" choice may be infeasible; a method of bringing one node IN may bring another node OUT. If a choice fails to bring all desired nodes IN, it is rejected and another choice made. (Reconciliation will fail when multiple assumptions are being revised but cannot be made true simultaneously due to interactions among them. In this event, the interpretation is rejected—it depends upon inconsistent assumptions.)

As an example of reconciliation, consider how to explain that testing is in fact done given the situation diagrammed in Figure 3; this is one part of reconciling the interpretation shown in Figure 6. To bring *tests_done(SV)* IN, we must force *not tests_done(SV)* OUT; that means forcing OUT both the nodes *applicable(TC2,SV)* and *applicable(TC3,SV)*. This can be done in two ways. One is to bring IN both *spec_excluded(TC2,SV)* and *spec_excluded(TC3,SV)* by adding two "dummy" justifications to support these nodes. The other is to block justification J1 that currently supports *standard_test(SV)*; note that this change affects only this instance of rule J1, not any other instances. This latter alternative is preferred, since it involves one rather than two changes. The result of choosing and installing this alternative is shown in Figure 7.

4.0 Summary

Effective problem-solving performance in plan recognition requires extensive domain knowledge. In open world applications of plan recognition, the only way to exploit deeper knowledge about the context for actions is to introduce uncertainty, in the form of plausible assumptions about missing data. We have shown how to formalize this deeper knowledge, how assumptions determine the credibility (rather than validity) of interpretations, how credibility can be used to make reasoned choices, and how faulty assumptions can be revised when necessary. This extension to plan recognition has been implemented in the GRAPPLE system, which runs the examples in this paper.

Figure 7: Example of Reconciliation



5.0 References

[1] Allen, J.F. "Recognizing Intentions from Natural Language Utterances." in *Computational Models of Discourse*, ed. M. Brady, Cambridge, MA: MIT Press, 1983.

[2] Azarewicz, J. et al. "Plan Recognition for Airborne Tactical Decision Making." *Proceedings of AAAI* (1986), 805-811.

[3] Carver, N.; Lesser, V.R.; and McCue, D. "Focusing in Plan Recognition." *Proceedings of AAAI* (1984), 42-48.

[4] Charniak, E.; Reisbeck, C.K.; and McDermott, D.V. *Artificial Intelligence Programming*. Hillsdale, New Jersey: L.Erlbaum Associates, 1980.

[5] DeJong, G.F., and Mooney, R.J. "Explanation-Based Learning: An Alternative View." *Machine Learning* 1:2 (April 1986), 145-176.

[6] Doyle, J. "A Truth Maintenance System." *Artificial Intelligence*, 12 (1980), 231-272.

[7] Genesereth, M. "The Role of Plans in Automated Consultation." *IJCAI*(1979), 311-319.

[8] Huff, K.E., and Lesser, V.R. "A Plan-based Intelligent Assistant that Supports the Software Development Process." *Proceedings of Third Symposium on Software Development Environments,* ACM (November 1988), 97-106.

[9] Kautz, H.A., and Allen, J.F. "Generalized Plan Recognition." *Proceedings of AAAI* (1986), 32-37.

[10] Reiter, R. "A Logic for Default Reasoning." *Artificial Intelligence*, 13(1980), 81-132.

[11] Reiter, R., and Criscuolo, G. "On Interacting Defaults." *IJCAI* (1981), 270-276.

[12] Sacerdoti, E.D. *A Structure for Plans and Behavior*. New York: Elsevier-North Holland, 1977.

[13] Sidner, C.L. and Israel, D.J. "Recognizing Intended Meaning and Speaker's Plans." *IJCAI*(1981), 203-208.

[14] Tate, A. "Project Planning Using a Hierarchical Non-linear Planner." Department of Artificial Intelligence Report 25, Edinburgh University (1976).

[15] Wilkins, D.E. "Domain-Independent Planning: Representation and Plan Generation." *Artificial Intelligence*, 22(1984), 269-301.

[16] Wilensky, R. *Planning and Understanding*. Reading, MA: Addison-Wesley, 1983.