Planning from partial and multiple episodes in a case-based planner

Robert McCartney*
Department of Computer Science and Engineering
University of Connecticut
Storrs, CT 06269-3155
robert@cse.uconn.edu

Abstract

Case-based planning deals with two of the problems inherent in classical planning methods: the difficulty of determining the effects of a sequence of actions, and the inefficiency of building large, complex plans from low-level primitive actions. A plan is developed from a single case (a previous execution), and the execution of that plan provides another case from which to plan. In this paper we extend this model by allowing plan generation from multiple cases and parts of cases. These extensions are consistent with the approximate nature of case-based planning, and require limited additional information to be stored with the cases.

Introduction

Case-based planning deals with two of the problems inherent in classical planning methods: the difficulty of determining the effects of a sequence of actions from knowledge about the individual actions, and the inefficiency of building large, complex plans from low-level primitive actions. The insight is that it can sometimes be easier to plan by modifying a previously executed plan than by planning from scratch, and that if the domain is reasonably benevolent, the new plan is likely to have the same results as the old one. Under the case-based approach, previously executed plans (cases) take the role of plan operators, or schemas. How the set of primitive actions lead to the effects, whether individually or through interactions, is unimportant: the plan is based on the case as a whole. Using case-based planning at its most basic, a plan is developed from a single case, and the execution of that plan provides another case from which to plan.

An obvious extension of this model is to allow plans to be developed from multiple cases or parts of cases. As an example of the first, a plan to repair a door handle and change the shock absorbers on an automobile might be generated by generating a plan to do each task (from individual cases), then merging the two plans into one that does both. The execution of this plan will provide a case that does both repairs; it would be desirable if this new case could provide the information from which to plan either repair. Given that we can plan both from multiple cases and from parts of cases, we can generate more complex plans by combining plans generated from parts of multiple cases (so, for example, the door handle-shock absorber case and a case of adjusting the brakes and aligning the wheels could be used to generate a plan to change the shocks and align the wheels). At the logical limit, we could plan by combining primitive actions as with a classical planner.

Case-based plan generation (idealized)

In this paper, we present a model for planning from multiple and partial cases that makes minimal information demands on the representation of cases: in particular, it does not demand that the relationships between actions and effects be known or explicit. Our goal is to characterize the mechanisms common to virtually all case-based planning systems, then examine how generation of plans from multiple and partial episodes might be done within extensions to this characterization. The point of this paper is to examine these problems within an idealized, simplified model, and try to identify problems, concerns, and possible solutions that would arise (and have arisen) when designing and implementing real systems.

Case-based planning, as typified by CHEF [Hammond, 1989], is an integrated approach to planning, execution, and learning: plans are generated by adapting executed plans, the plan is verified by execution (or simulated execution), and the system learns as a result of that generation and execution. Generally speaking, case-based planning systems do not generate correct plans, but are either expected to learn to avoid incorrect plans over time, or are used in domains where the occasional incorrect plan is not a problem[1]; minimally

*This work has been supported in part by the National Science Foundation, grant IRI-9110961.

[1]Exceptions to this are the PRIAR system [Hendler and Kamphampati, 1988] and the SPA system [Hanks and Weld, 1992]. These use case-based techniques for increased efficiency only.

these systems learn by acquiring the results of their own executions. In this paper we are only concerned with the process of plan generation, not with the execution and learning components of a case-based planner, but the approaches offered here are designed to work within and interact with other parts of an integrated planning-execution-learning system.

A number of case-based planning systems use a hybrid approach for plan generation, combining features from case-based planning with features from classical approaches—JULIA [Hinrichs, 1991], for example, combines case-based planning with hierarchical problem solving and constraint-propagation. In this paper we examine the purely case-based process, not because it is superior to the hybrid approach, but simply because it may be easier to examine certain problems in isolation; any progress in this area should also lead to progress with the hybrid approaches as well.

Finally, much of the emphasis in case-based reasoning has been on similarity assessment, efficient retrieval, and adaptation. We do not consider these problems in this paper because our focus is elsewhere, but this should not imply that these problems are easy, or insignificant, or that we have solved them.

In the following sections, we first present a simple model for case-based plan generation from a single case, including a representation of plans and episodes that supports this generation. Next, we look at how this model might be extended to allow plan generation from multiple and partial cases, including assumptions and information requirements. Next, we look at some practical considerations and observations from our work with a particular case-based planning system that supports planning from multiple and partial cases. Finally, we discuss how other researchers have considered these issues.

Planning from a single case

Informally, case-based plan generation involves retrieving a previously executed plan (a *case*), and modifying it so that its effects include the current goal and its context matches the current situation (things that were true when the case was executed will be true when the plan is executed). This "plan" is meant to be replayed in the current situation, with the expectation that the effects in the plan will occur when it is executed. Notably, the relations between the actions in the case and the various effects are *not* necessarily explicit. We assume two things about our cases: the sequence of actions in a case led to the set of effects in that case, and that if we execute a similar sequence of actions in a similar context, we will get similar effects. These assumptions allow case-based plans to be generated with a minimal domain theory vis-a-vis actions and their effects, and allow us to avoid much of the complexity inherent in predicting the effects of a sequence of actions from knowledge about the individual actions' effects.

A case representation that supports this simple view of case-based planning can be based on two distinguished situations (points in time): the initial situation (before execution) and the final situation (after execution). The case includes facts for each of these situations, facts that are true over intervals of time that fall somewhere between these situations, actions that occur at various points between these situations, and temporal constraints relating these actions. By our assumptions, cases can generally be replayed: that is, if we execute the plan starting from a point where all of the initial situation facts hold, and we execute the actions subject to their temporal constraints, then the facts in the final situation will hold after the execution, and the intermediate facts will hold at the appropriate intermediate times. The represented case is actually an approximation of one that will always replay; it may not include some of the salient facts and actions (necessary for the case to replay) and may include some superfluous facts and actions. Adaptation can be introduced to this model by defining adaptations as transformations of cases that preserve behavior; if a case can generally be replayed, so can the transformed version of that case if the adaptation is applicable to the situation.

Given such a representation, we can represent a goal as a subset of a case; a partial specification of some desired behavior. Plan generation, then, is the process of finding a case and a set of adaptations, such that the goal is a subset of the adapted case; the adapted case is the plan, and it can be executed by simply "doing it again." This model is presented more formally in the Appendix. This leads to a close correspondence between plans and cases: each plan corresponds to the adaptation of a single case.

Multiple and partial cases

Multiple cases

Informally, using multiple cases should work as follows:

1. Find a set of cases that combine to satisfy your goals.
2. Generate a plan from each case.
3. Merge[2] the set of plans into a single one.

In the previous section (and in the Appendix), we discussed step 2 of this process, generating a plan from a single case. To plan with multiple cases, we need two additional mechanisms, one to partition the goals among individual cases and one to take a set of plans, each solving part of the goal, and merge them into a single plan that solves the entire goal.

Partitioning the goals among cases can be done by finding the set of plans that solve part of the goal, then choosing a set of plans that "cover" the goal set. This

[2]We use the term merge in a generic sense, that of combining the propositions and actions of separate plans into a single plan. This is distinct from the usage in [Foulser *et al.*, 1992], who use it to describe certain plan optimizations.

potentially may be difficult, depending on whether or not there is some notion of optimality of the partitioning; if the constraint is the minimum or maximum number of cases, this reduces to set cover, an NP-complete problem [Aho *et al.*, 1974], even if all of the partial plans have been generated. In most domains we have to deal with this to some degree: optimality may not be necessary, but there will likely be different costs associated with using different sets of cases.

When merging a set of plans into a single plan, we face two issues: synchronization and partial plan interaction. Consistent with the general approach of case-based planning, we can assume that partial plans do not interact unless we have reason to believe that they will—in most domains where case-based planning might be appropriate we are unable to detect such potential interactions anyway—and then try to repair problems caused by such interactions at execution time. Over time, we would hope to learn to avoid combining partial plans that led to problems in the past.

Synchronization is still an important issue, however—the situations mentioned in the goals need to be consistent across the set of partial plans. How this is to be accomplished depends on how the goal is partitioned: what situations, if any, are shared by different partial plans.

Under certain conditions, plans can be combined sequentially. Consider a goal that can be satisfied by combining two plans A and B, where A's initial situation is the initial situation of the goal, and B's final situation is the final situation of the goal. If B's initial situation propositions are a subset of A's final situation propositions, then B can immediately follow A. Failing this, if each of B's initial situation propositions is either a proposition of A or in the initial execution situation, then B can follow A if we can assume that B's initial situation propositions persist from where we know they are true until B's initial situation.[3]

For some situations, sequential combination is not possible: the component plans share situations (such as the initial or final situations of the plan). In this case, we make use of any temporal information present in the plans being merged. As discussed in the Appendix, our formal model allows ordering constraints (such as $\mathit{precedes}(S_k, S_j)$ and time stamps such as $\mathit{value\text{-}in}(S_k, \mathit{clock\text{-}time}) = T$) to be placed on situations. Any temporal relations between situations in the final plan must reflect those in the constituent plans.

In simple cases, for example when two plans have the same initial situation, it may be possible to achieve this by simply normalizing the partial plans to a common relative clock time. For cases when more than one situation needs to be shared, however, the problem is more serious, requiring various adjustments and persistence assumptions. A particularly interesting simple case is the one where two partial plans need to share the same initial and final situations. If the partial plans have different durations, then one or both need to be adjusted. The methods for changing the duration are generally domain-dependent; in some domains it may be possible to increase the duration of the shorter plan by inserting a predecessor to the initial situation or a successor to the final situation and assuming that the appropriate propositions persist. This is not possible in general; if we allow global temporal constraints, such as a contraint on the duration of the plan, it may be impossible to produce a plan within that constraint by merging a set of partial plans that are all within the constraint.

For either the sequential or parallel approaches to lead to a correct plan, the plans must not interfere with each other. Unless we can reason about resource constraints and interactions of the actions from the two plans, we will sometimes find that the combined plans do not do what we expected.

[3]Actually, we could use something like TWEAK's modal truth criterion [Chapman, 1987], that is, show that the proposition is always true before B, or if it is false in some situation before B we can show that it becomes true in a situation between that situation and the start of B.

Partial cases

Similarly, an informal description of planning from a partial case is as follows. Given a goal,

1. Find a part of a case that satisfies your goals.
2. Generate a plan from that part of a case.

To do this requires that we be able to extract partial cases from which to plan, in particular, partial cases that include the initial situation information corresponding to preconditions for the partial plan. This is not necessarily possible given our model for case-based planning: even for a well-behaved case (one that gives sufficient conditions to allow replay to succeed), there is no assumption about the effects of subsets of the plan actions, nor is there any assumption that the intermediate data are complete in terms of predicting what is true at intermediate parts of the plan. This aside, we can propose two characteristics of these extractable partial plans that support their use.

First, the partial cases should be cases: they should have identifiable initial and final situations, and they should be self-contained relative to their events (any ordering constraint in the partial case must be in terms of events in that partial case). Second, all of the information in the partial case is information that is in the complete case, that is, if C' is a partial case of case C, then $\mathit{propositions\text{-}of}(C') \subseteq \mathit{propositions\text{-}of}(C)$[4]

These partial cases can be used in planning as if they were complete cases, and they may or may not result in working plans. They are cases, and we can still assume that they generally can be replayed, but

[4]The propositions of a case are the facts and actions of a case without distinguishing the initial and final situations. This allows a partial case's initial and final situations to be different from those in the case of which it is a part.

the assumption is weaker, as the evidence of it working is only in the context of the case that it is part of. Moreover, forcing it to be a subset of the case may make it a worse approximation, since salient information for the partial case need not be salient for the case. As with the cases, the predictive quality of the partial cases may be best assessed by experience, a problem made more difficult since a "good" partial case may be contained in a "bad" case.

Combining partial and multiple plans

Both multiple and partial cases can be used in the same case-based planner; indeed, given the nature of a case-based planner reusing its executions it seems likely that using one would imply the use of the other:

- If we plan by combining multiple plans, the resulting case is one that has identifiable partial cases: the parts corresponding to each multiple plan.
- If we plan from partial plans, the extraction of partial cases provides examples (in the original cases) of executed plans that could be viewed as combinations of multiple cases—the associated partial plans.

More importantly, if we have both of these methods, we have the flexibility to plan by combining parts of more than one execution. While these plans may not be as reliable as ones generated from a single execution, it should greatly increase the range of plans we can generate.

Augmenting the representation

The primary representational concern here is how to support extracting partial cases from cases. This could be done on the fly, extracting partial cases as they were needed, but this has two disadvantages: it requires a good deal of reasoning about the case during generation, and it ignores the information about partial cases that has been learned from previous plan generations. The alternative is to explicitly represent the partial plans of a given case. This would allow easy extraction of partial cases, and provide a notation for the "parts" in cases obtained by executing plans generated from multiple cases. While we might choose to extract some partial cases as needed, the easy availability of this information from executions suggests that we augment the case representation by another element, the set of partial cases. These partial cases may in turn have partial cases associated with them, so a given partial case may have a number of successively containing supercases. Once the case or subcase has been chosen to use to generate all or part of a plan, its partial case information is no longer used in generating that plan, but it can easily be included in the case resulting from the new plan's execution.

Practical considerations and observations

Planning from multiple and partial cases is supported in COOKIE [McCartney, 1990], an integrated case-based planning and execution system that works in the domain of meal planning and preparation. Some of the methods have been simplified or otherwise influenced by the system's domain; the knowledge representation has been affected by the interpretation of goals by agents specifying the goals in using this system. Although certain practical considerations are reflected in this system, it is essentially true to the model presented in the last few sections.

The goals supplied to COOKIE are sets of propositions that may include variables: variables are bound via unification with propositions in the case, and a case satisfies a goal if the goal is a subset of the case. The goals given to COOKIE are typically underconstrained: for a given goal there are multiple possible cases (or combinations of cases) to satisfy it. COOKIE generates plans in order, favoring single episodes over multiple episodes and cases requiring no adaptation over those requiring adaptation. When it is necessary to plan from multiple cases, the goals are partitioned using a greedy algorithm, choosing the case that matches the most of the goal, then the one that matches the most of the remaining goal, and so on. The system proposes a solution that meets the user goals, then allows the user to ask for other solutions interactively for the same or a modified goal (more about this below). Once an acceptable plan is generated, the execution monitor DEFARGE [McCartney and Wurst, 1991] controls the execution of the plan via interaction with an external cook. The results of the execution are stored as a new case.

Two considerations in COOKIE regarding the use of multiple and partial plans bear further explanation: how multiple plans are synchronized, and how partial plans are identified and chosen.

Synchronization of multiple plans

One characteristic of COOKIE's episodes is that the final situation of a meal preparation is when the meal is served (with certain exceptions, such as desserts). Synchronization, therefore, is typically done on the final situation—multiple plans are made equal in duration by adding time to the beginning of the shorter subplans (after the initial situation, before its first action). This is a reasonable approach in this domain: it is based on the assumption that facts are more likely to persist before the cooking has started than after. It also has the effect of synchronizing the initial situations, so all of the initial conditions can be verified before execution commences—akin to the notion of collecting all of the ingredients before starting.

Choosing the right partial plans

The cooking episodes stored in COOKIE typically have a number of partial cases—meals are usually made up of a number of somewhat disjoint dishes—and these partial cases often have partial cases of their own. For a typical American Thanksgiving dinner, for example, there are partial plans corresponding to stuffed turkey, various vegetables, desserts, and beverages. The stuffed turkey partial plan has a partial plan corresponding to preparing stuffing, which in turn has partial plans related to cubing bread and preparing sauteed onions, and so on. Suppose the system is given the goal of preparing stuffing. Is the appropriate case the stuffing subcase, the turkey-and-stuffing subcase, or the Thanksgiving dinner case in entirety? Any of these would satisfy the goal under the definition. Originally, COOKIE used the "minimal partial plan" criterion: choose the smallest partial case that satisfies the goal. While this may be reasonable in some domains, it was not here: implicitly, a goal in this domain means "a meal in which this goal is met"; if I ask for pasta, I would also like a sauce even if I did not mention it. On the other hand, a "maximal partial plan" criterion also can have problems: if I request parts of three meals I do not necessarily want all of the parts of all three.

For some domains one of the extreme criteria may be appropriate, or it may be possible to determine the goal-specifier's intent from the goal. Our solution is to involve the user: we offer the user the plan made up from maximal partial plans, then allow him or her to specify what parts of the plan to keep. The system then uses the minimal partial plan criterion on the new goal: the original goal plus the user-identified characteristics. This is rather a hack, but it allows the use of highly underconstrained goals. This process of refining specifications based on what they lead to has been useful in other domains—software design for example [Balzer, 1985]—where the goals do not adequately capture all of the success criteria.

Resource conflicts and optimizations

In COOKIE, no attempt is made to recognize potential resource conflicts inherent in merging plans. Although many of these conflicts could be detected fairly easily, some are quite subtle (if an oven has been used for baking, it stays too warm for letting bread rise for a long time after it has been turned off, for example). Our rationalization is that the easily-detectable conflicts are probably the easiest to repair at execution time, so the efforts are being put into repair mechanisms that could be useful for difficult to detect conflicts as well. Similarly, we do not attempt to optimize merged plans, although the potential for merging subplans (like cooking two things in the same pan) certainly exists. If a plan has been optimized by combining steps, we can still represent the individual parts by including the shared steps in each partial plan, so inclusion of some optimization in merging would cause no great problems for future reuse of parts.

Related work

A number of other researchers have looked at the problem of multiple/partial cases in planning.

The use of MACROPS by STRIPS and PLANEX [Fikes *et al.*, 1972] illustrated the approach: after a plan was generated, it was generalized into a MACROP, or macro-operator, which encoded the structure of the plan in a table. This MACROP can be used in generating subsequent plans; in fact, it defines a set of partial plans. Once the desired characteristics are chosen, the appropriate operator sequence (a subset of the sequence in the original plan) can be extracted and used as an operator.

PLEXUS [Alterman, 1988] planned by adapting specific previously developed plans when faced with a similar situation. If a failure occurred during execution, one of the alternative ways to repair the failure was to patch in part of a different plan. This was possible in part because of the amount of detailed information associated with the partial plans: preconditions, goals, and taxonomic relations for example.

CELIA [Redmond, 1990], a case-based problem solver used for automotive troubleshooting, plans and executes sequences of diagnostic procedures by using parts of multiple diagnosis cases. Each diagnosis is structured into subsequences of events called ***snippets***. Snippets are associated with the goals that they lead to, plus applicability conditions for their use, their effects, and what snippets they followed and preceded in the original case. These are combined sequentially during execution based on the result of executing previous snippets in the sequence. They provide explicit partial plans, and their containing case provides a potential sequence of snippets to solve a particular problem.

The *multicase* [Zito-Wolf and Alterman, 1992] provides another method for representing and combining partial plan information. The multicase is a structure for representing a number of episodes. It starts with a simple directed graph representing a sequence of operations, then combines graphs that do similar tasks by merging common subsequences. The resulting graph defines a set of possible sequences, each case being a traversal of this graph corresponding to one possible sequence. The choice points in the graph correspond to alternative partial plans; planning can be viewed as choosing the paths in the traversal. This approach provides a highly storage efficient alternative to snippets, with the advantage that not only actual neighbors of partial cases are easily found, but potential neighbors as well. The combination of partial cases is sequential and constrained by multicase topology.

One difference between these approaches and the one in COOKIE is that these other systems use partial plans sequentially, while in COOKIE partial plans are usually merged in parallel. For sequential combination, the position within a sequence can be quite useful; the equivalent information in COOKIE for a partial plan is what other partial plans were executed with it in parallel, information that is included implicitly in the case representation.

A more fundamental difference may be due simply to COOKIE's notion of goal as a subset of an episode; given this, if a partial plan satisfies a goal, then its containing plan (and all superplans) will likely satisfy it as well (as long as they violate no constraints). COOKIE's merging of complete plans, then allowing further specification to determine which partial plans to keep, is reasonable for situations with underspecified goals, and the specification of hierarchies of partial episodes means that the decision on what part of the episode should be used is made at generation time rather than at storage time—each case can provide a number of possible plans at different levels of granularity.

The NoLIMIT nonlinear problem solver [Veloso *et al.*, 1990] does the equivalent of extracting partial plans on the fly. It builds a partial-order of operations from a totally ordered sequence of operators, which identifies partial plans that can potentially be executed in parallel. Following this, it uses this representation to reason about resource allocation among the parallel subplans. To build these partial orders (in NoLIMIT) requires detailed information about operator effects and preconditions that a case-based planner would not generally have. However, this work suggests the possibility of generating a partial-order from information that is available (explicit ordering dependencies) or easily inferred in the context of a case-based planner; the partial-order generated will likely be less complete than those used in NoLIMIT, but may still be quite useful. This area is one that we are currently exploring with COOKIE.

Conclusions

This paper presented a simple approach to planning from multiple and partial cases in a case-based planner. The approach puts minimal information requirements on the represented case, and is based on straightforward assumptions about non-interaction between partial plans. It does not produce plans that are guaranteed to succeed—which would be unrealistic given that the underlying model of planning from a single case cannot do this—but proposes solutions that have some likelihood of success. The approach increases the range of plans that a case-based planner can generate from a given case-base without making unrealistic assumptions on how well the planner can reason about the state of the world during the execution of a plan. Little in this approach is new—most of the functionality of this approach has been realized in a number of systems—but it provides a simple model that can be used as a basis for comparison and explanation of alternative case-based planning approaches and implementations.

Acknowledgements

Many of the ideas contained in this paper came from discussions with current and former members of the COOKIE group at UConn: Barbara Cuthill, Jung-Jin Lee, Ben Moreland, Madeleine Pukinskis, David Towers, Alma Whitten, and Karl Wurst. Thanks also to Kate Sanders for technical and editorial discussions, and to David Leake and the anonymous reviewers whose suggestions I tried to incorporate. This work has been supported in part by the National Science Foundation, grant IRI-9110961.

References

Aho, Alfred V.; Hopcroft, John E.; and Ullmann, Jeffrey D. 1974. *The design and analysis of computer algorithms.* Addison-Wesley, Reading, MA.

Alterman, Richard 1988. Adaptive planning. *Cognitive Science* 12:393–421.

Balzer, Robert 1985. A 15 year perspective on automatic programming. *IEEE Transactions on Software Engineering* 11(11):1257–1268.

Chapman, David 1987. Planning for conjunctive goals. *Artificial Intelligence* 32(3):333–377.

Davis, Ernest 1990. *Representations of Commonsense Knowledge.* Morgan-Kauffmann Publishers, Inc., San Mateo, CA.

Fikes, Richard E.; Hart, Peter E.; and Nilsson, Nils J. 1972. Learning and executing generalized robot plans. *Artificial Intelligence* 3:251–288.

Foulser, David E.; Li, Ming; and Yang, Qiang 1992. Theory and algorithms for plan merging. *Artificial Intelligence* 57(2-3):143–181.

Hammond, Kristian J. 1989. *Case-based planning: viewing planning as a memory task.* Academic Press.

Hanks, Steven and Weld, Daniel S. 1992. Systematic adaptation for case-based planning. In *Proceedings of the first international conference on AI planning systems*, College Park, MD. 96–105.

Hendler, James A. and Kamphampati, Subbarao 1988. Refitting plans for case-based reasoning. In Kolodner, Janet, editor 1988, *Proceedings of a workshop on case-based reasoning.* 179–181.

Hinrichs, Thomas R. 1991. Problem solving in open worlds: a case study in design. Technical Report GIT-CC-91/36, College of Computing, Georgia Institute of Technology. (PhD Thesis).

McCartney, Robert and Wurst, Karl R. 1991. DEFARGE: a real-time execution monitor for a case-based planner. In *Proceedings of the 1991 DARPA Case-based Reasoning Workshop*, Washington, DC.

McCartney, Robert 1990. Reasoning directly from cases in a case-based planner. In *Proceedings of the 12th annual conference of the Cognitive Science Society*, Cambridge, MA. 101–108.

McCartney, Robert 1992. Case-based planning meets the frame problem: case-based planning from the classical perspective. In *Proceedings of the first international conference on AI planning systems*, College Park, MD. 172–178.

Redmond, Michael 1990. Distributed cases for case-based reasoning: facilitating use of multiple cases. In *Proceedings of the eighth national conference on artificial intelligence*, Boston, MA. 304–309.

Veloso, Manuela M.; Perez, M. Alicia; and Carbonell, Jaime G. 1990. Nonlinear planning with parallel resource allocation. In *Proceedings of the 1990 DARPA Innovative Approaches to Planning, Scheduling and Control Workshop*, San Diego, CA.

Zito-Wolf, Roland J. and Alterman, Richard 1992. Multicases: a case-based representation for procedural knowledge. In *Proceedings of the 14th annual conference of the Cognitive Science Society*, Bloomington, IN.

Appendix: The formal model

We can formalize the model for case-based plan generation given earlier; see [McCartney, 1992] for more details.

We assume that cases are represented as explicit sets of propositions in a situation-based temporal logic that corresponds closely to the temporal representation in [Davis, 1990] (Chapter 5). Fluents, states, and event types are used to reify time-varying terms, time-varying relations, and events. The function *value-in* relates fluents to situations (situations correspond to instants in time), the relation *true-in* relates states to situations, and the relation *occurs* relates event-types to intervals of situations. We can associate clock times with situations by defining a "clock time" fluent and specifying its value in particular situations.

We base our representation on two distinguished situations: an initial situation which precedes all others in the plan, and a final situation which follows all others. Given these, we can represent a case by a tuple, $\langle \mathcal{I}, \mathcal{F}, \mathcal{P}, \mathcal{M} \rangle$, where $\mathcal{I}$ and $\mathcal{F}$ are subsets of the facts in the initial and final situations, $\mathcal{P}$ is the set of events (of the form *occurs*$([S_i, S_j], E)$, where $[S_i, S_j]$ is the interval from situation S_i to situation S_j and E is an event type) and their ordering constraints (such as *value-in*$(S_k$, *clock-time*$) = T$, or *precedes*(S_k, S_j)), and $\mathcal{M}$ is a set of facts that are true in situations other than the initial and final situation, of the form *true-in*(S, B), where B is a propositional fluent and S is a situation corresponding to the beginning or end of an event. We can represent goals in the same way, that is, a 4-tuple corresponding to the partial specification of a plan.

Plan generation is the following process; for goal $\langle \mathcal{I}_g, \mathcal{F}_g, \mathcal{P}_g, \mathcal{M}_g \rangle$, find a case C and a composition of applicable adaptations T such that

$$\text{propositions-of}(\langle \mathcal{I}_g, \mathcal{F}_g, \mathcal{P}_g, \mathcal{M}_g \rangle) \subseteq \text{propositions-of}(T(C))$$

Applicable adaptations are those that preserve behavior: if a case is well-behaved (defined below), then its transform is also well-defined. The "propositions of" a tuple are all of the facts without the initial-final situation groupings, *i.e.*, the the union of $\mathcal{P}$, $\mathcal{M}$, true-in(x, S_I) for all x in $\mathcal{I}$, and true-in(x, S_F) for all x in $\mathcal{F}$, where S_I and S_F are the initial and final situation respectively. Alternatively, we could use subset relations between the individual tuple elements, which would be equivalent if we required correspondence between the initial and final situations of the case and goal. The transformed case is a plan: for T(C) = $\langle \mathcal{I}_c, \mathcal{F}_c, \mathcal{P}_c, \mathcal{M}_c \rangle$, $\mathcal{P}_c$ is the set of actions to be taken with partial-order constraints, $\mathcal{I}_c$ is the set of facts that should be true in the initial situation of the execution, $\mathcal{F}_c$ is the set of facts that should be true in the final situation of the execution, and $\mathcal{M}_c$ is the set of facts that should be true at various points in the execution corresponding to the beginnings and ends of actions.

The bases for optimism here are assumptions that the adapted case T(C) approximates a well-behaved case, and that its behavior is approximately the same as the case it approximates. A well-behaved case is one for which replay always works:

$$\begin{aligned} &\text{well-behaved}(\langle \mathcal{I}, \mathcal{F}, \mathcal{P}, \mathcal{M} \rangle) \iff \\ &\quad (\forall_{S,S_f} \text{true-in}(S, \mathcal{I}) \land \text{occurs}([S, S_f], \mathcal{P}) \Rightarrow \\ &\quad\quad \text{true-in}(S_f, \mathcal{F}) \land \forall_\phi \phi \in \mathcal{M} \Rightarrow \phi) \end{aligned}$$

That is (extending *true-in* to sets of propositions and *occurs* to sets of actions in the obvious way), for any situation in which the facts in $\mathcal{I}$ hold, if we execute the steps in $\mathcal{P}$ subject to all of the timing constraints in $\mathcal{P}$, then all of the facts in $\mathcal{F}$ will hold in the final situation of the execution, and each intermediate fact in $\mathcal{M}$ will hold in the appropriate situation (beginning or end of an action).

By assumption, each case $\langle \mathcal{I}, \mathcal{F}, \mathcal{P}, \mathcal{M} \rangle$ has associated with it a well-behaved case $\langle \mathcal{I}_w, \mathcal{F}_w, \mathcal{P}_w, \mathcal{M}_w \rangle$. A case is a description of an actual plan execution; the corresponding well-behaved case is the ideal description of that same execution:

$$\begin{aligned} \exists_{S,S_f} \quad & \text{occurs}([S, S_f], \mathcal{P}') \land \text{true-in}(S, \mathcal{I}_w) \land \\ & \text{true-in}(S, \mathcal{I}) \land \mathcal{P} \subseteq \mathcal{P}' \land \mathcal{P}_w \subseteq \mathcal{P}' \land \\ & \text{true-in}(S_f, \mathcal{F}_w) \land \text{true-in}(S_f, \mathcal{F}) \land \\ & \forall_\phi \phi \in \mathcal{M} \Rightarrow \phi \land \forall_\theta \theta \in \mathcal{M}_w \Rightarrow \theta \end{aligned}$$

That is, our represented cases are approximations of well-behaved cases; the facts in the initial and final situations (as well as in $\mathcal{M}$) of our cases may be missing some salient facts and include superfluous facts, and similarly may be missing salient events and/or include superfluous events (defining salient and superfluous as present in and absent from the well-behaved case). The plans generated from these cases, therefore, will sometimes fail.