Generating Macro Operators for Decision-Theoretic Planning*

AnHai Doan **Peter Haddawy**
{*anhai,haddawy*}@*cs.uwm.edu*

Department of Electrical Engineering and Computer Science
University of Wisconsin-Milwaukee
PO Box 784
Milwaukee, WI 53201

1 Introduction

Generation of macro operators has long been known to be an effective speed-up learning technique for classical planners [3, 7]. It is to be expected that this benefit will carry over to decision-theoretic planners as well. But in decision-theoretic planning additional benefits can be gained from the use of macro operators due to the necessity of comparing alternative plans and the extremely high cost of plan projection. Projecting a probabilistic plan yields an exponential number of outcomes as a function of plan length. Trade-offs between the accuracy of projection and its cost are therefore necessary. One way to do this is to reduce the set of actions and/or the set of branches in each action by creating abstract actions. In a previous paper [4], we presented methods for abstracting alternative actions to reduce the set of actions, and abstracting groups of branches within an action. In this paper we show how abstract macro operators can be generated which compactly represent a sequence of actions and thus reduce the cost of projection. A comprehensive theory of abstraction can be found in [2].

Our work on generating macro operators was motivated by our application of the DRIPS decision-theoretic refinement planning system to the problem of selecting the optimal test/treat strategy in a particular medical domain [5]. DRIPS efficiently searches the space of possible plans to identify the optimal plan by representing the space using an abstraction/decomposition network (a kind of AND/OR tree). An abstraction node is a description of a set of alternative actions (its instantiations), while a decomposition node is a description of a sequence of actions. The planner uses the network to reason about and prune away suboptimal classes of plans. Currently the user must provide the abstraction/decomposition network. We found that specifying this information is a tedious and error prone task, particularly specifying the decomposition descriptions, and desired a tool to generate these automatically. In section 5 we provide a description of the medical domain and show how our implemented system generated a particular macro operator.

The rest of the paper is organized as follows: Section 2 briefly presents an uncertainty model for world and action based on probability intervals; Section 3 identifies various abstraction types and discusses sequential abstraction methods in particular; Section 4 presents an implementation that works with the representation used by the DRIPS planner; Section 5 presents an extended example; we conclude with discussions in Section 6.

2 World and Action Model

We call a complete description of the world a *state*. Exactly how a state describes the world depends on the application. A state may be a snapshot of the world or it may be a complete history. The theory developed is independent of the particular interpretation. Denote the set of all possible states as Ω, and the set of all probability distributions over Ω as $PD(\Omega)$. We start by defining various types of sets of probability distributions over Ω. A *mass assignment* $m : 2^{\Omega} \rightarrow [0,1]$ assigns to each subset of Ω a probability mass portion. We have, therefore, $\sum_{B \subseteq \Omega} m(B) = 1$ and $m(\emptyset) = 0$. Set $B \subseteq \Omega$ is called a *focal element* of m if $m(B) > 0$; and we say the pair $\langle B, m(B) \rangle$ forms a *branch* of m. A probability distribution P is said to be *consistent* with a mass assignment m iff $P(B) = \sum_{b \in B} P(b) \geq \sum_{C \subseteq B} m(C)$ for all $B \subseteq \Omega$. Denote the set of all intervals in $[0,1]$ as I; a *general mass assignment* $M : 2^{\Omega} \rightarrow I$ assigns to each subset of Ω a (probability) interval in I. M can be understood as a set of mass assignments $M = \{m | m : 2^{\Omega} \rightarrow [0,1] \ s.t. \ \forall B \subseteq \Omega \ m(B) \in M(B)\}$. Throughout the paper, M will be interpreted either as a function or a set depending on the context. Denote the set of all probability distributions consistent with a mass assignment m as $\wp(m)$; we define $\wp(M) = \bigcup_{m \in M} \wp(m)$.

*This work was partially supported by NSF grant #IRI-9207262.

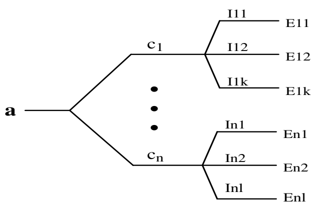

Figure 1: General form of an action description.

We represent uncertainty about the world and action effects with general mass assignments, i.e., with the set of probability distributions represented by the general mass assignment. This differs from most existing probabilistic planning models which represent uncertainty by a single probability distribution [6, 10]. With actions that can have metric effects, we must deal with mass assignments where we cannot determine in advance which state belongs to a focal elemment B, because B is represented by a set of constraints. In such an environment, general mass assignments have more expressive power than simple mass assignments. (For a more detailed discussion see [2].)

Throughout the paper we will be talking about action descriptions, but for brevity will refer to them simply as actions. Actions serve as transformations from a set of probability distributions $\wp_{pre}$ into a set of probability distributions $\wp_{post}$. The set $\wp_{post}$ is a function of a and $\wp_{pre}$, and will be denoted as $exec(\mathsf{a}, \wp_{pre})$.

Definition 1 (Action) *An action* a *is specified by a finite set of conditions* $\{c_1, c_2, \cdots, c_n\}$. *Each* c_i *is a logical sentence uniquely defining a set* $c_i = \{b \mid b \in \Omega; b \models c_i\}$. *The* c_i *are mutually exclusive and jointly exhaustive on* Ω. *They will be interpreted either as logical sentences or sets depending on the context. Each* c_i *is associated with a finite set* $\{(I_{ij}, E_{ij}) \mid j = 1, 2, \cdots, i_{k_i}; I_{ij} \in I; E_{ij} : 2^\Omega \rightarrow 2^\Omega\}$. *We define* $exec(\mathsf{a}, \wp_{pre})$ *recursively as follows*
a) for $b \in c_i$, $exec(\mathsf{a}, b) = \wp(M_{ib})$, *where for* $A \subseteq \Omega$

$$M_{ib}(A) = \begin{cases} I_{ij} & \text{if } A = E_{ij}(b), 1 \le j \le i_{k_i} \\ 0 & \text{otherwise} \end{cases} \qquad (1)$$

b) for $P_{pre} \in PD(\Omega)$, $exec(\mathsf{a}, P_{pre}) = \{P_{post} \mid P_{post}(A) = \sum_b P_{pre}(b) \cdot P_b(A)$ *for all* $A \subseteq \Omega; P_b \in exec(\mathsf{a}, b)\}$
c) for $\wp_{pre} \in PD(\Omega)$,
$exec(\mathsf{a}, \wp_{pre}) = \{P_{post} \mid P_{post} \in exec(\mathsf{a}, P_{pre}); P_{pre} \in \wp_{pre}\}$

When there is no ambiguity, we shall drop i in the subscipt of M_{ib}. An action can be represented as a tree as shown in Figure 1, or as a finite set of triples (branches) $\{(c_l, I_l, E_l)\}$. Each E_l is termed an *effect* of the action. The semantics of an action a is that, for $b \in \Omega$, the uncertainty about the state resulting from executing a on b is characterized by a set of probability distributions $\wp(M_{ib})$.

To enhance the practical applicability of this action model we would like to represent $\wp_{pre}$ and $\wp_{post}$ as compactly as possible; ideally they should both be general mass assignments. Unfortunately $exec(\mathsf{a}, \wp(Mpre))$ (Definition 1.c) in most cases can not be represented as a general mass assignment. Define M_{post} as an approximation of $exec(\mathsf{a}, \wp(Mpre))$ ($\wp(Mpost) \supseteq exec(\mathsf{a}, \wp(Mpre))$); we seek a projection rule to calculate M_{post} from a and M_{pre}. We define upper and lower conditional probabilty $P^*(c_i \mid B)$ and $P_*(c_i \mid B)$ such that if $B \cap c_i = \emptyset$ or $M_{pre}(B) = 0$ then $P^*(c_i \mid B) = P_*(c_i \mid B) = 0$; if $B \subseteq c_i$ then $P^*(c_i \mid B) = P_*(c_i \mid B) = 1$; otherwise $P^*(c_i \mid B) = 1$ and $P_*(c_i \mid B) = 0$. Denote the interval $[P_*(c_i \mid B), P^*(c_i \mid B)]$ as $P_I(c_i \mid B)$.

Projection rule 1 *Given an action* a *with effects* E_{ij}, *for* $B \subseteq \Omega$ *define* $E_{ij}(B)$ *to be a set such that* $E_{ij}(B) \supseteq E_{ij}(b)\ \forall b \in B$. *For condition* c_i *of* a *and* $B \subseteq \Omega$, *define* $M_{iB}(.)$ *general mass assignment similar to* M_{ib} *in (1) with* b *replaced by* B, *then for* M_{pre}, M_{post} *is calculated for all* $A \in 2^\Omega$ *as*

$$M_{post}(A) = \sum_{B \subseteq \Omega} M_{pre}(B) \cdot \left(\sum_i P_I(c_i \mid B) \cdot M_{iB}(A) \right) \qquad (2)$$

The above projection rule is correct, i.e., for any general mass assignment M_{pre}, action a, we have $exec(a, \wp(Mpre)) \subseteq \wp(M_{post})$, where M_{post} is calculated using (2). (See [2] for a proof of correctness.)

Projection rule 1, denoted as $project_1$, states that in order to compute M_{post}, we project each branch (triple) of a on each branch of M_{pre}. In practice, the set of M_{pre}'s branches is finite. Function $project_1$ can be calculated easily based on P_* and P^*. It is not hard to see that the complexity of $project_1$ is equal to that of straightforward projection [6] in single probability distribution schemes.

3 Sequential Action Abstraction

Projection rule 1 is sound, i.e., it does not leave out any possible "post-execution" probability distribution. Furthermore $project_1(\mathsf{a}, M_{pre})$ can be computed much faster than $exec(\mathsf{a}, M_{pre})$.

Unfortunately, in plan projection, even $project_1$ is not fast enough, since its computational time grows exponentially as a function of plan length. We would like to seek $project_2$ and action a^* (a^* as a function of a) such that M^*_{post} satisfying $M^*_{post} = project_2(\mathsf{a}^*, M_{pre})$ and $M^*_{post} \supseteq exec(\mathsf{a}, M_{pre})$ could be computed faster than M_{post}. As we observed, projecting action a on M_{pre} amounts to projecting every branch (c_l, I_l, E_l) of a on every branch of M_{pre}. We conclude therefore that if a^* has fewer branches than a then projecting

a^* should be faster. This can be done by abstracting a set of branches of a into a single branch of a^*. We refer to this process as *intra-action abstraction.*

Given two actions a_1 and a_2 we could seek $project_2$ and a^* (a^* as a function of a_1 and a_2) such that $M^*_{post} = project_2(\mathsf{a}^*, M_{pre})$ and $M^*_{post} \supseteq exec(\mathsf{a}_1, M_{pre})$, $M^*_{post} \supseteq exec(\mathsf{a}_2, M_{pre})$. Abstracting a set of actions into a single action in this way is referred to as *inter-action abstraction.*

In this paper, we will discuss the third type of abstraction, which is referred to as *sequential abstraction,* the abstraction of a sequence of actions $\mathsf{a}_1\mathsf{a}_2\cdots\mathsf{a}_n$ into a^* such that $M^*_{post} = project_2(\mathsf{a}^*, M_{pre}) \supseteq exec(\mathsf{a}_n, exec(\mathsf{a}_{n-1}, \cdots(exec(\mathsf{a}_1, M_{pre}))\cdots))$. Action a^* is commonly referred to as *macro operator.* Due to the space limitation, we will skip the constructive discussion of sequential abstraction, but state the abstraction procedure. (See [2] for more details.)

We define a new projection rule that is easier to implement, and can be applied in more situations, although it yields looser probability bounds than projection rule 1.

Projection rule 2 *Given a condition c_i of action* a *we define $P_+(c_i \mid B)$ and $P^+(c_i \mid B)$ such that*
a) if B does not satisfy c_i, that is, $B \cap c_i = \emptyset$, then $P_+(c_i \mid B) = P^+(c_i \mid B) = 0$
b) else if B satisfies at least two branches with different conditions in a *(including branch c_i) then $P_+(c_i \mid B) = 0$ and $P^+(c_i \mid B) = 1$*
c) else B satisfies only the branches with condition c_i and $P_+(c_i \mid B) = P^+(c_i \mid B) = 1$.
Projection rule 2 is obtained from projection rule 1 by replacing $P_I(c_i \mid B)$ in (2) with the interval $[P_+(c_i \mid B), P^+(c_i \mid B)]$.

We consider only the case of abstracting two actions; generalization by recursion is straightforward. We abstract the action sequence $\mathsf{a}_1\mathsf{a}_2$ into abstract action a^* by pairing every branch of a_1 with every branch of a_2 and create an abstract branch of a^* out of each of these sets. Consider abstracting one such pair: (c_1, I_1, E_1) of a_1 and (c_2, I_2, E_2) of a_2 into (c, I, E^*) of a^*. Let $c'_2 = \{b \in B | E_1(b) \cap c_2 \neq \emptyset\}$. We choose
a) $c = c_1 \wedge c'_2$,
b) $I = [\min I_1 \cdot \min I_2, \max I_1 \cdot \max I_2]$,
c) E^* such that $E^*(B) \supseteq E_2(E_1(B))$ for all $B \subseteq \Omega$.

In practice c'_2 is often a logical sentence derived from c_2 and E_1. If we can conclude that the sentence $c_1 \wedge c'_2$ is false then we do not have to create the branch (c, I, E^*) corresponding to the set with condition c_1 of a_1 and c_2 of a_2. A projection rule is correct wrt this abstraction if the general mass assignment M^*_{post} resulting from projecting the abstract action will subsume the set $exec(\mathsf{a}_2, exec(\mathsf{a}_1, M_{pre}))$. When the effect E_1 is one-to-one, i.e., for all $b \in \Omega \cap c_1$ $E_1(b) \in \Omega$, and $E_1(B) = \bigcup_{b \in B} E_1(b)$ or $E_1(B) = \bigcup_{b \in B \cap c_1} E_1(b)$, both projection rules 1 and 2 are correct; otherwise only projection rule 2 is correct[1].

4 An implementation

The DRIPS planner searches through a space of plans represented in an abstraction/decomposition network. A decomposition node describes the effects of a sequences of actions. To automate the process of generating this description, we have implemented the above algorithm for the representation used by the DRIPS planner.

We start by describing the world and action representations used by DRIPS. The world is characterized by a set of features called *attributes*. Each attribute takes a value from a set called its *domain*. Time is modeled to be continuous, but we are only concerned with a countable number of changes in time. At any time point each attribute is either undefined or takes on a unique value. Attributes are therefore functions from time into their domains. We will write *attribute-name(time-point)* to denote both the function corresponding to *attribute-name* and its value at *time-point*. The allowed forms of these value functions will be given at the end of this section.

A *snapshot* of the world at a particular time point is represented as a set of attribute/value pairs. An *abstract snapshot* is a (possibly infinite) set of snapshots at a time point t, often specified as a set of constraints on the possible values of attributes at t. We shall use the notation $s(t)$ to denote snapshot s at time point t, and refer to both snapshot and abstract snapshot as snapshot because their difference is irrelevant to our problem. We define *presnapshot*(a) as the snapshot on which we start executing action a. The *initial snapshot* is the snapshot on which we start executing our plan.

A *chronicle* is a possible realization of world's affairs, a function from time into the set of concrete snapshots S. We allow no exogenous events, so the only changes in chronicles (world) result from the planning agent's actions. An *abstract chronicle* is a function from time to 2^S. We will refer to both individual chronicles and abstract chronicles simply as chronicles when the meaning is clear from the context. Note that when we refer to a chronicle we will typically be interested in only a portion of it from some beginning time to some end time. We shall keep this in mind as we discuss properties of chronicles. Chronicles correspond to states in the uncertainty model discussed in Section 2. We

[1] When discussed abstraction types are repeatedly used on a set of actions, the abstraction depth of an action a can be greater than one. A projection rule is then said to be correct if the result of projecting a subsumes the result of "executing" *any* sequence of concrete actions obtained by refining a. Our result is also valid for this case.

define a special attribute *present*, which specifies the present time in a chronicle. We will assume that a chronicle is only a complete decsription of the world up to the timepoint specified by *present*.

Actions in DRIPS conform to Definition 1. Each condition is associated with a set of probability distributions over a finite set of chronicles, indicating possible further developments of world's affairs. Each c_i is a set of constraints on the presnapshot $s(t_0)$, involving only attribute values at t_0. The probability interval I_i specifies the probability of a possible branching of time constrained by c_i and E_i. Each E_i is a set of functions specifying attribute values starting from a time point $t_{i1} \geq t_0$ to some later time point t_{i2}. We say a chronicle is *constrained* or *specified* by E_i iff from the time point t_{i1} to t_{i2} the *only* changes in the chronicle are specified by the set E_i.

We are now in a position to specify the form of the attribute value functions in more detail. Attribute values $attr(t_{attr})$ will be specified either as part of the initial snapshot description ($t_{attr} = t_0$) or as a part of the effect of an action. For any time point $t \geq t_0$ we require that either 1) $attr(t)$ is given a value from its domain or 2) the value of $attr(t)$ is specified by an *algebraic expression* involving only other attribute values at time points no greater than t in such a way that $attr(t)$ is completely specified by forward projection. Our purpose is to rule out specifications such as $attr_1(t) = attr_2(t) + 2$ and $attr_2(t) = attr_1(t) - 2$, from which computing either value at t is impossible. For any other time point $t > t_{attr}$ for which we do not explicitly specify $attr(t)$ we define $attr(t) = attr(t_1)$, where $t_{attr} \leq t_1 < t$, $attr(t_1)$ is explicitly specified, and there does not exist $t_2 \in (t_1, t)$ such that $attr(t_2)$ is explicitly specified.

The above definition of attribute value function is consistent with the requirement of allowing no exogenous event. Note that action effects specify *changes* in attribute values, but do not impose *constraints* among them (such as *tons-delivered* $\leq$ *tons-in-truck*). Reasoning about the persistence of such constraints is difficult and, furthermore, they are more appropriately specified as domain axioms.

We now consider abstracting the sequence of two actions a_1a_2 into action a^*. Assume that a_1 will be executed at t_0 and a_2 at t_1. Consider the case of abstracting triple (c_1, I_1, E_1) of a_1 with (c_2, I_2, E_2) of a_2 into (c, I, E^*) of a^*. According to the abstraction procedure discussed in Section 3, $c = c_1 \wedge c'_2$ and we must calculate c'_2 from E_1 and c_2. Condition c_2 contains only constraints on attribute values at time point t_1, the starting time of action a_2. Due to our restrictions on the forms of attribute value functions, the value of each attribute *attr* at timepoint t_1 can be expressed through an algebraic expression referring to the values of attributes at any time point earlier and taking into account all changes that occurred between. The following algorithm generates this expression.

Reduce(attr, t_1, t_0)

1. If $attr(t_1)$ is a value in its domain then return that value;

2. else compute $t_2 \leq t_1$ such that $attr(t_2)$ is explicitly specified, and there does not exist $t_3 \in (t_2, t_1]$ such that $attr(t_3)$ is explicitly specified, (based on the attribute value function restrictions in section 4).

3. If $t_2 < t_0$ return the expression $attr(t_0)$ else if $attr(t_2)$ is a value, return that value;

4. else call **reduce** for each term in the algebraic expression for $attr(t_2)$, replace them with their result from the calls, and return the expression.

The algorithm can loop if the function specification is circular (such as in the set $attr_1(t) = attr_2(t)$, $attr_2(t) = attr_1(t)$). To prevent this we must keep track of the pairs *(attr, t)* we have referred to. If we refer to any pair that is already in the set then the algorithm terminates, reporting an incorrect effect specification; otherwise it terminates, yielding the result.

From the above algorithm it follows that any constraint on attribute values at t_1 can be translated into a constraint on attribute values at any timepoint $t_0 < t_1$ by replacing each attribute term with argument t_1 by its expression with argument t_0. Since c_2 is a set of constraints on attribute values at t_1, it can be transformed into a set of constraints c'_2 on attributes at time point t_0 using E_1 of action a_1. We call the above transformation a *reduction* of the set c_2 at t_1 to c'_2 at t_0. Condition c_2 is satisfied in the chronicle constrained by c_1 and E_1 iff the set $C = \{c_1, c'_2\}$ is consistent. Constraint solving techniques such as Screamer [8] can be used to check the consistency of C to see whether it is necessary to create branch (c, I, E^*). Note that the constraint solver must be sound but need not be complete; if the inconsistancy of C cannot be detected the only thing we loose is some extra projection due to the created branch (c, I, E^*).

We now proceed to create the macro operator a^* representing a_1a_2. Consider every possible pairing of (c_i, I_i, E_i) in a_1 and (c_j, I_j, E_j) in a_2. Reduce c_j to the set c'_j of constraints on attributes at time point t_0, the starting time of a_1 so that $C = c_i \wedge c'_j$ refers to the constraints on attributes at the same time point. If C is consistent (checked by any suitable constraint checking technique) then create an abstract branch of a^* with condition C, probability $I = I_i * I_j$, and effect $E^* = E_i \cup E_j$.

Sequential abstraction may be combined with other abstraction techniques to yield abstract macro operators. We illustrate this idea through the example in the next section.

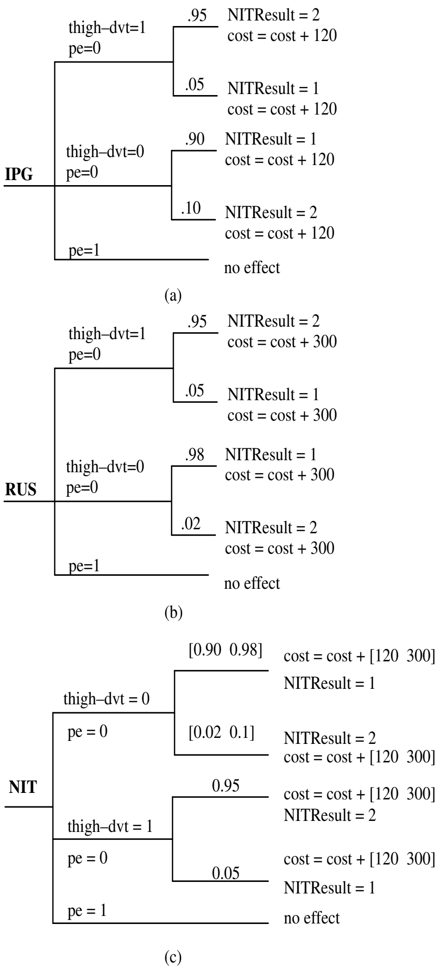

Figure 2: (a) Impedance plethysmography (b) Real-time ultrasonography (c) Non-invasive test.

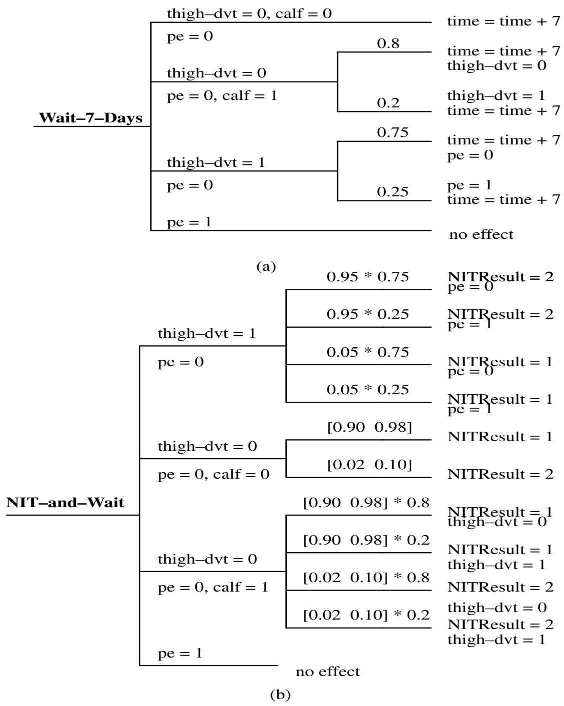

Figure 3: (a) Wait action (b) Macro operator NIT-and-Wait.

5 Examples

The following example is taken from a medical problem to which we successfully applied the DRIPS planning system [5]. The task is to choose the optimal test/treat strategy for managing patients suspected of having lower-extremity deep vein thrombosis (blood clot in the calf or thigh). We have several possible tests to choose from, including two non-invasive tests: real-time ultrasonography (RUS) and impedance plethysmography (IPG). We can perform a sequence of conditional tests with possible waiting periods in between after which we conditionally administer anticoagulation therapy. The two tests as well as one waiting action are shown in figures 2.a, 2.b, and 3.a.

Since the sequence of test and wait could be repeatedly executed many times, abstracting the sequence could substantially reduce the cost of projection. We will produce an abstraction of this sequence by applying our macro operator algorithm, as well as our previously developed abstraction techniques [4, 2]. We first use the inter-action abstraction technique to abstract IPG and RUS together into NIT, as shown in Figure 2.b.

We now create the macro operator NIT-and-Wait from the sequence NIT, Wait-7-Days. We will show in detail how the third triple of NIT and the first triple of Wait-7-Days are combined. Let c_i be $\mathit{thigh}(t_0) = 1$, $\mathit{pe}(t_0) = 0$, I_i be 0.95, and E_i be $\mathit{NITResult}(t_1) = 2$, $\mathit{cost}(t_1) = \mathit{cost}(t_0) + [120 \quad 300]$, where $(t_1 > t_0)$. Let c_j be $\mathit{thigh}(t_1) = 0$, $\mathit{calf}(t_1) = 0$ and so on with I_j and E_j.

The next step is to reduce c_j from t_1 to c'_j at t_0. From the definition of value function we have $\mathit{thigh}(t_1) = \mathit{thigh}(t_0) = 1$, $\mathit{calf}(t_1) = \mathit{calf}(t_0)$ since the only change E_i in the chronicle does not effect *thigh* or *calf*. So the set c'_j is $\{1 = 0, \mathit{calf}(t_0) = 0\}$. c'_j is inconsistent, so $C = c'_j \wedge c_i$ is also inconsistent and no triple corresponding to these two triples is created. The complete macro operator is shown in Figure 3.b. For simplicity we omit the specification of *time* and *cost* from the action's effects.

Due to the large number of branches of this operator, projection using it could be very costly. We reduce the cost of projection by using intra-action abstraction to group together some of the branches, resulting in the action description shown in Figure 4.

In applying inter- and intra-action abstraction we lose

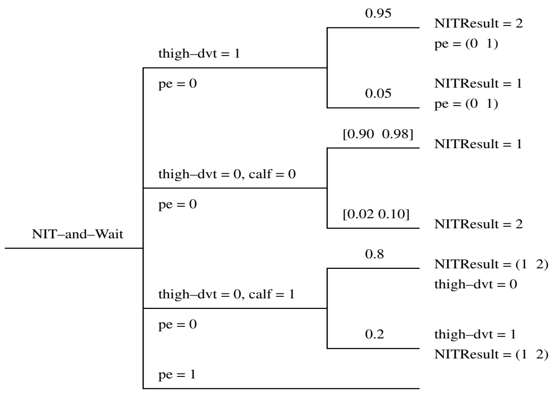

Figure 4: Abstraction of NIT-and-Wait.

some accuracy, but as we noted earlier, this is a necessary evil if we are to reduce the cost of projection. Now we can use NIT-and-Wait instead of the sequences RUS, Wait-7-Days and IPG, Wait-7-Days.

6 Discussion

Lower probability has long been used to represent a set of probability distributions by specifying constraints on the probability of subsets of the state space. Readers familiar with the above concept will recognize mass assignment as a subclass of lower probability. Dealing with metric domains made it necessary for us to generalize the concept of mass assignment into that of general mass assignment.

Chrisman [1] presents a model in which uncertainty is represented by a single mass assignment. He then provides a projection rule that computes a post mass assignment containing all possible post probability distributions resulting from "executing" the action. His rule is tighter than the rule $project_1$ we suggested in Section 2, but his rule requires exponentially more computation than ours.

Abstraction in probabilistic domains has long been recognized as an effective tool to faciliate faster reasoning. In this paper we focused on the technical side of obtaining a correct macro abstraction; a comprehensive theory of abstracting probabilistic actions is presented in [2]. Empirical results of applying the technique to several domains showed substantial reduction in the complexity of planning [2]. The sequential abstraction method presented here could be easily adopted to use in both open-loop probabilistic planning [6] and Markov models [9]. We have ignored the question of abstracting schematic actions. Further work will be focused on this problem, as well as on investigating abstractions based on qualitative properties of actions.

References

[1] L. Chrisman. Abstract probabilistic modeling of action. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, pages 28–36, June 1992.

[2] A.H. Doan and P. Haddawy. Decision-theoretic refinement planning: Principles and application. Technical Report TR-95-01-01, Dept. of Elect. Eng. & Computer Science, University of Wisconsin-Milwaukee, January 1995. Available via anonymous FTP from `pub/tech_reports` at `ftp.cs.uwm.edu`.

[3] R.E. Fikes, P.E. Hart, and N.J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3(4):251–288, 1972.

[4] P. Haddawy and A.H. Doan. Abstracting probabilistic actions. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 270–277, Seattle, July 1994. Available via anonymous FTP from `pub/tech_reports` at `ftp.cs.uwm.edu`.

[5] C.E. Kahn and P. Haddawy. Optimizing diagnostic and therapeutic strategies using decision-theoretic planning: Principles and applications. In *Proceedings of the Eighth World Congress on Medical Informatics (Medinfo'95)*, Vancouver, July 1995. (to appear).

[6] N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic least-commitment planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1073–1078, Seattle, 1994.

[7] S. Minton. Selectively generalizing plans for problem solving. In *IJCAI85*, pages 596–599, 1985.

[8] J.M. Siskind and D.A. McAllester. Nondeterministic lisp as a substrate for constraint logic programming. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 133–138, Washington,DC, July 1993.

[9] David E. Smith and Mike Williamson. Representation and evaluation of plans with loops. In *AAAI Spring Symposium 95 - Extending Theories of Actions*, Stanford, CA, 1995.

[10] S. Thiebaux, J. Hertzberg, W. Shoaff, and M. Schneider. A stochastic model of actions and plans for anytime planning under uncertainty. *International Journal of Intelligent Systems*, 1994. To appear.