Toward Combining Domain Theory and Recipes in Plan Recognition

Rogelio E. Cardona-Rivera
Department of Computer Science
North Carolina State University
Raleigh, NC 27695 USA
recardon@ncsu.edu

R. Michael Young
School of Computing
University of Utah
Salt Lake City, UT 84112 USA
young@cs.utah.edu

Abstract

We present a technique to further narrow the gap between recipe-based and domain theory-based plan recognition through decompositional planning, a planning model that combines hierarchical reasoning as used in hierarchical task networks, and least-commitment refinement reasoning as used in partial-order causal link planning. We represent recipes through decompositional planning operators and use them to compile observed agent actions into an incomplete decompositional plan that represents them; this plan can then be input to a decompositional planner to identify the recognized plan-space plan. Our model thus synthesizes the heretofore disparate recipe-based and domain theory-based plan recognition variants into a unified knowledge representation and reasoning model.

Introduction

Plan recognition is a form of activity recognition (Sukthankar et al. 2014) that attempts to predict the future behavior of an intelligent agent given a sequence of observations of that agent's past behavior. Plan recognition assumes *a priori* that there exist a set of possible goals that an agent cares to achieve, and given a sequence of agent action observations, a plan recognizer attempts to identify: a) the goals that explain an agent's observed actions, and b) the actions the agent will effect in pursuit of those goals.

Historically, plan recognition systems have relied on *plan libraries*, which encode a collection of *recipes* (goal-plan pairs) that record likely actions toward assumed goal states (Carberry 2001). Recent work on plan recognition has explored alternatives to the library-based model; e.g. planning-based recognizers that rely on a domain theory (Ramírez and Geffner 2009).

However, there are instances where recipes are still useful constructs in the plan recognition task. For example, if we were asked to recognize the plan of an agent engaged in a cyber-attack, we would need to identify typical and not necessarily optimal plans of the agent, who is presumed to want to avoid detection (Geib and Goldman 2001; Buchanan 2016). Further, recipes can be used to capture abstract or thematic notions of an agent's activity that could explain more than just an agent's plans for subsequent activity, such as their *intent* (Lesh, Rich, and Sidner 1999).

Ideally, we want to retain the flexibility, generality, and scalability of the domain theory-based approach and the representation of typical non-optimal action sequences afforded by library-based approach. In this paper we outline a plan recognition model, which enables just that.

Contributions We present the following technique to combine domain theory-based and recipe-based plan recognition: compile the observations of an agent into a partial plan that a *decompositional planner* can address during planning. Decompositional planning combines *hierarchical reasoning* as discussed in hierarchical task networks (HTNs) (Erol, Hendler, and Nau 1994) and *least-commitment refinement reasoning* as discussed in partial-order causal link planning (Weld 1994) in a unified knowledge representation, and a sound and complete reasoning procedure (Young, Pollack, and Moore 1994).

For this technique to work, we introduce an algorithm that can compile the observations into a sound but incomplete decompositional plan. This decompositional plan is then used to seed a decompositional planning process. To anticipate our later discussion, our technique proposes using *composite action schemata* and associated *decomposition schemata* (which decompose a composite action into more primitive actions) to represent *recipes*; collectively, composite actions and associated primitive actions achieved through decomposition enable formulating plan recognition recipes in a general manner (Kautz and Allen 1986). These recipes are used in a procedure that transforms observed agent actions into goals that a decompositional planner must address during planning; *i.e.* into *flaws* that are *refined* (Kambhampati, Knoblock, and Yang 1995) by the decompositional planner.

Further, we present a commentary on our proposed compilation procedure as well as a detailed walk-through of it in a motivating context: plan-recognition in a cyber-security domain.

Related Work

Recent research has cast plan recognition as a planning process (Ramírez and Geffner 2009; Sohrabi, Riabov, and Udrea 2016), wherein the agent is assumed to behave

optimally: the agent's observed actions are compiled as additional goals (to the set of assumed goals) that a recognizer must plan for at no extra cost using the planning domain theory (planning domain paired with an initial state). The use of a domain theory combined with state-of-the-art planning algorithms make the plan recognition process more general, flexible, and scalable.

The probabilistic approach proposed by Ramırez and Geffner (2010) relaxes the assumption of agent optimality, such that their domain theory-based recognizer admits non-optimal action sequences, but it suffers from two limitations: (1) it does not afford a way to record typical action sequences in a particular domain as a way to guide the recognizer, and (2) it assumes that we have access to a prior distribution over assumed goals, which (if inaccurate) may negatively bias the recognizer (Golan and Lumsdaine 2016).

Existing library-based approaches (e.g., Avrahami-Zilberbrand and Kaminka 2005, Amir and Gal 2011) and probabilistic grammar-based approaches (e.g., Geib and Goldman, 2011) fail to account for how individual actions have more than one effect; an action contributes to more than just the parent action or rule it belongs to. Asserting an action with multiple effects (in the STRIPS-sense) can enable an action sequence that does not fit neatly into a library recipe, nor a grammar rule. Further, while acyclic libraries and grammars can be compiled into a domain theory that represents them (Lekavỳ and Návrat 2007; Ramirez and Geffner 2016), the compilation technique produces a domain theory P_L for a library L that cannot be effectively combined with a non-library domain theory P_T to perform plan recognition. This is because the logical language (i.e. fluents) used in the domain theory P_L can express relations between actions only in terms of the hierarchy in the library L. Since P_T's fluents are different from P_L's, they cannot be combined in one plan recognition system because the approaches are using two different (logical) languages.

To date, there exists no plan recognition system that can combine a domain theory and a plan library in a principled manner for the production of a recognized plan.

Combining Domain Theory and Recipes

In this section, we (1) introduce the decompositional planning model, (2) present an example illustrating the intuition behind our technique, (3) discuss a compilation procedure that enables using the technique, and (4) reflect on the technique's overall strengths and weaknesses.

Decompositional Planning

The formal model of decompositional planning we adopt is taken from DPOCL, a decompositional partial-order causal link planning system previously developed by Young, Pollack, and Moore (1994).

DPOCL is based on classical partial-order causal link (POCL) planning (Weld 1994), which searches through a graph in which nodes are partially specified plans and arcs are plan-refinement operations that add constraints to a partial plan (Kambhampati, Knoblock, and Yang 1995). Refinements are carried out in a least-commitment style, wherein the planner only adds a refinement that is strictly needed to ensure plan soundness.

A partial plan is a partially-ordered sequence of actions (Sacerdoti 1975). We adopt a STRIPS-like representation for actions, where action step is identified by a unique name, an action *type* (e.g. RUN, PICK-UP), a set of *preconditions*, and a set of *effects*. Preconditions are literals that must hold in the world prior to an action's execution, and effects are literals made true through the execution of the action. For generality, literals in an action's preconditions and effects may contain variable terms; each may be bound to a constant term in the domain.

Our model enables capturing action hierarchies. In DPOCL, actions can be *primitive* or *composite*.

Definition 1 (Action Schema)**.** *An* action schema *is a template for an action which can occur in a planning domain. It is a tuple $\lambda = (T, P, E, \vartheta)$, where T is an action type that identifies the action schema (e.g., "move," "pick-up"); P is a set of* preconditions*, literals that must be true immediately before the action can be executed; E is a set of* effects*, literals made true by the execution of the action; and ϑ is a label that identifies this schema as* primitive *or* composite*. P and E can have variable terms to convey ideas such as "move from* `?x` *to* `?y`*."*

An action schema is also known as an *operator*; we use both interchangeably. An instantiated primitive action schema is referred to as a *primitive step* and an instantiated composite action schema is referred to as a *composite step*. *Steps* are uniquely named instances of action schemata; each step is assigned a unique label to distinguish that step from other instances of the action schema.

A primitive step is directly executable in a DPOCL planning domain (assuming its preconditions are satisfied), whereas a composite step is not; a *decomposition schema* is required to specify how the composite step is decomposed into more primitive steps.

Definition 2 (Decomposition Schema)**.** *A* decomposition schema *is a single-layer expansion of a composite step. It is a tuple of the form $\delta = (T, \mathcal{S}^*, B, \prec, L_C)$, where T is an action type; $\mathcal{S}^*$ is a set of* pseudo-steps*; B is a set of binding constraints over the variable terms of the steps in $\mathcal{S}^*$; $\prec$ is a set of orderings over the steps in $\mathcal{S}^*$; and L_C is a set of causal links over the steps in $\mathcal{S}^*$.*

In the same way that steps carry with them a label to distinguish them from other instances of a specific operator, so too do the steps of a decomposition schema carry with them a label to distinguish them from other instances of the same operator in the schema; this is why they are referred to as *pseudo-steps* in Definition 2. During plan generation, each pseudo-step will either be associated with another step already in the plan (and given that step's name), or will be instantiated as a new step and given a unique name.

Each decomposition specifies a sub-plan whose ultimate effect achieves the composite step being decomposed. Despite being partial, they can not be made up of arbitrary steps; each decomposition schema (1) contains a dummy initial step s_0 whose effects are the preconditions of the

parent step; (2) contains a dummy final step s_∞ whose preconditions are the effects of the parent step; (3) has ordering constraints ensuring that s_0 precedes all other steps in the sub-plan, and that s_∞ follows all other steps in the sub-plan; and (4) each effect of s_0 has a path of causal links that terminates in precondition of s_∞.

During plan construction, a DPOCL planner adds steps to the partial plan in order to guarantee two things.

First, for each step in the partial plan, all of the step's preconditions are true before it is executed, and not undone between the moment at which the preconditions are true and when they are needed. A precondition can be true in the initial state or made true by the effect of an earlier step. *Causal links* explicitly record these precondition satisfaction relationships. A causal link connects two plan steps s_1 and s_2 via a literal p, denoted $s_1 \xrightarrow{p} s_2$, when s_1 establishes a condition p in the world needed by s_2 to execute.

Second, if a composite step has been added to the partial plan, it must be fully decomposed to the level of primitive steps. *Decomposition links* explicitly record the choice of decomposition used to refine a composite step. A decomposition link connects three plan steps s_c, s_0^d, and s_∞^d, denoted $s_0^d \overset{\curvearrowleft}{} s_c \overset{\curvearrowright}{} s_\infty^d$, where s_c is a composite step whose decomposition is the partial sub-plan bounded by step s_0^d, which encodes s_c's preconditions as its effects, and step s_∞^d, which encodes s_c's effects as its preconditions.

A plan is *sound* if its variable bindings and ordering constraints are consistent. It is *complete* if it contains no *flaws* (Kambhampati, Knoblock, and Yang 1995). A flaw in DPOCL planning is one of three things: an *open precondition*, a *threatened causal link*, or an *unexpanded composite step*. An open precondition flaw is the case when there exists a step in the partial plan with a precondition that has not been established via a causal link. A threatened causal link flaw is the case when there exists a step in the partial plan that can possibly be ordered such that it undoes a condition established via a causal link. An unexpanded composite step flaw is the case when there exists a composite step in the plan with no decomposition link that specifies its sub-plan. The original DPOCL Planning Algorithm is not presented due to space considerations, but is provably *sound* and *primitive complete*[1] (Young, Pollack, and Moore 1994).

Our Technique

Our technique combines domain theory (i.e. a planning domain) and recipes in the production of a recognized plan. Recipes are encoded using decompositional planning data structures, which are added to the planning domain definition. We assume to be working with a *keyhole plan recognizer*, which does not account for any information beyond the observed agent's actions. As mentioned, recipe-based plan recognition operates over a plan library, a set of recipes that record likely actions toward assumed goal states. Because recipes in our formulation are in the context of plan-space planning, recipe goals are framed as the effects of

[1]For every solution π to a planning problem where π only contains primitive steps that are causal ancestors of the goal state, DPOCL will produce a plan whose primitive steps are π.

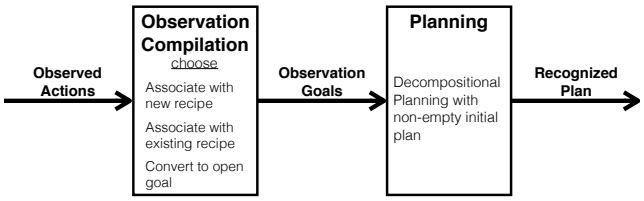


Figure 1: Illustration of the proposed technique for combining domain theory and recipes in plan recognition. Arrows represent information and boxes represent algorithmic modules. Observed actions are transformed into an incomplete DPOCL plan that is completed by a planner to produce a recognized plan.

composite action schema, as in Definition 1. A recipe is thus a pairing of a composite action schema and a decomposition schema. Formally:

Definition 3 (Recipe)**.** *A* recipe *is a tuple* $R = (\lambda, \delta)$*, where* $\lambda = (T, P, E, \vartheta)$ *is a* composite *action schema as in Definition 1; and* $\delta = (T, \mathcal{S}^*, B, \prec, L_C)$ *is a decomposition schema as in Definition 2, such that the action type* T *of both* λ *and* δ *is equivalent.*

Recipes in DPOCL are encoded implicitly via a planning domain; every combination of compatible (i.e. of the same action type) composite action schema-decomposition schema pairs in the domain is a recipe. Given a planning domain that includes recipes, and a sequence of observed agent actions, our technique proceeds in two pipelined phases as illustrated in Figure 1:

Phase 1: Compiling the observed agent's actions into an incomplete DPOCL plan that represents those actions.

Phase 2: Using a DPOCL planner to refine the plan produced in Phase 1 until no flaws remain.

In this paper, we present an algorithm that accomplishes Phase 1, enabling our overall technique of combining domain theory and recipes. As discussed, Phase 2 is accomplished by a DPOCL planner; we use the Longbow planning system (Young 1994).

Example To illustrate the intuition behind our approach, we apply our technique to our motivating context: plan recognition in a cyber-security domain. In particular, we assume to be attempting to recognize the plan of an agent engaged in a *cyber-attack*, as illustrated in Figure 2. The phases in a cyber-attack are considered to be composite in the sense that there are multiple candidate ways they can be achieved (Buchanan 2016).

In this example, we process the observation sequence in Figure 3 one action at a time. When we dequeue the first observed action "open net ports" from the sequence, our technique attempts to recognize the action as part of a recipe in the library. This association process looks for every recipe in the library whose decomposition contains a pseudo-step of the same action type as the observation. Assume that our technique finds one such recipe, namely $R =$ ("establish foothold", "foothold-through-malware"), whose decomposition "foothold-through-malware" contains a pseudo-step of type "open net ports." Once found,

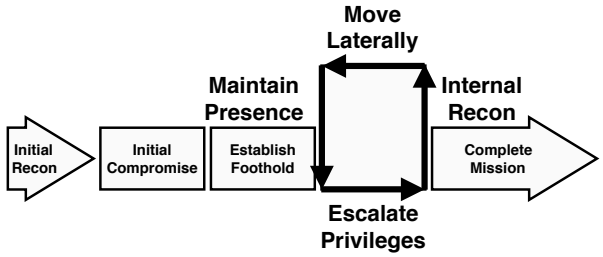


Figure 2: One interpretation of the *Cyber-Attack* life-cycle. Each stage represents a composite action schema with multiple possible more primitive realizations (i.e. decomposition schemata). The fourth stage (from left to right) presents a loop of abstract actions, which may continue for an undetermined period of time.

Figure 3: An example sequence of agent observations in a cyber-security domain. The black boxes represent primitive actions of the named type (i.e. types are "open net ports," "install backdoor," and "trigger cross-side script" respectively), and are processed in order from left to right. Dotted circles represent preconditions, and filled-in circles represent effects.

this recipe is added to our empty plan-space plan with the respective pseudo-step replaced for the observed action, as illustrated in Figure 4a. The empty partial plan is updated by: (1) adding the composite action "establish foothold," (2) adding the decomposition schema "foothold-through-malware" (with corresponding pseudo-steps, bindings, orderings, and causal links), (3) replacing the pseudo-step "open net ports" with the observation of the same type, and (4) adding a decomposition link $s_0^d \swarrow$ "establish foothold" $\searrow s_\infty^d$ where s_0^d and s_∞^d are "start malware sub-plan" and "end malware sub-plan," respectively. The remaining pseudo-steps "install backdoor," "enter," and "download malware" remain as pseudo-steps that are available for integration of future observations.

When we dequeue the second observed action "install backdoor" from the sequence, our technique (instead of associating it with a new recipe as before) matches it with the recipe constructed previously by simply replacing the corresponding pseudo-step in the action, as illustrated in Figure 4b. Further, because "install backdoor" was observed after "open net ports," an ordering is added to the plan to reflect as such.

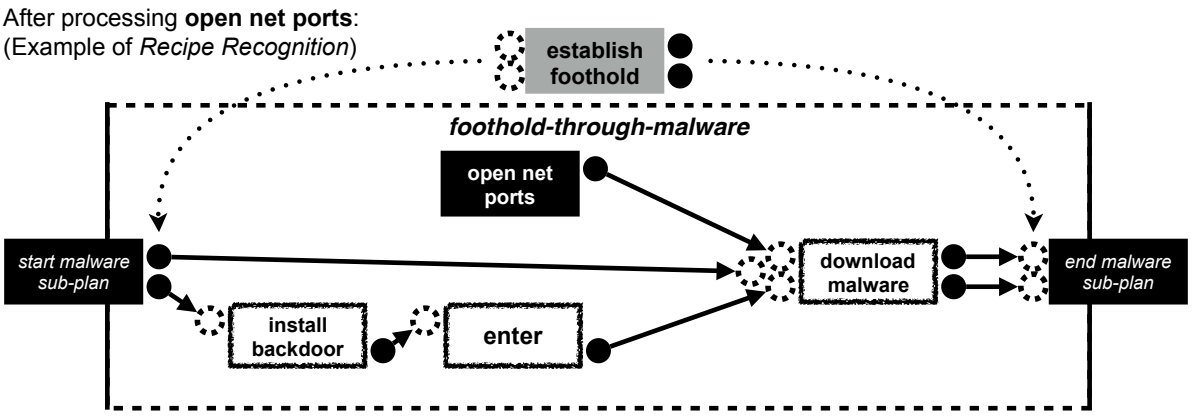
When we dequeue the third observed action "trigger cross-side script" from the sequence, our technique cannot associate it with an existing recipe, as illustrated in Figure 4c. Assume that our technique cannot find an applicable recipe for this action; in this case, we simply add it to the partial plan and introduce an ordering to the partial plan to reflect it coming after the action "install backdoor."

At this point, we have run out of observations and have constructed a consistent but incomplete DPOCL plan. This plan is then used to initialize a search by a DPOCL planner, which should ensure it becomes sound and complete.

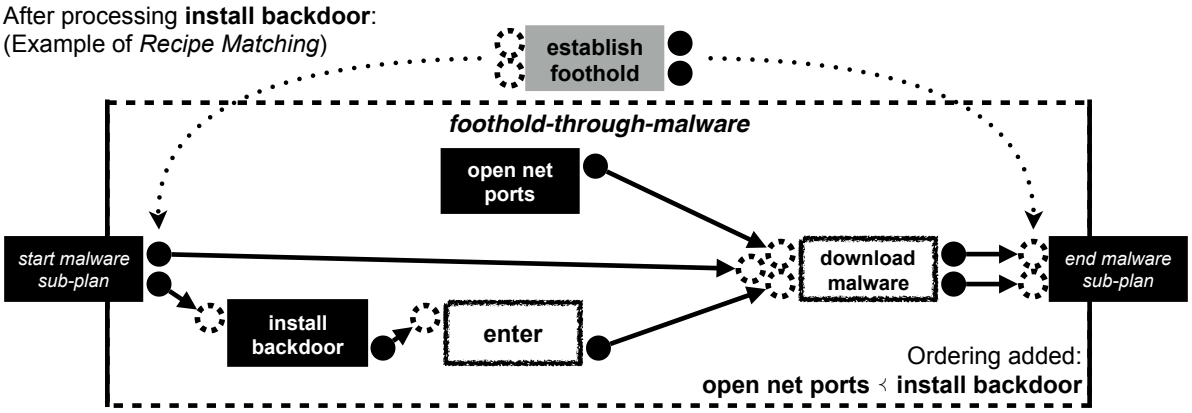
Compiling Observations We present an observation compilation procedure for the first phase of the technique illustrated in Figure 1. The second phase uses a DPOCL planner, originally defined by Young, Pollack, and Moore (1994). The compilation builds upon Ramírez and Geffner's intuition of *observations as goals*, and leverages the framing of *goals as islands* (Hayes-Roth and Hayes-Roth 1979): intuitively, we compile observations of actions into a DPOCL plan with which to seed a plan-space planning process. This compiled plan represents an intermediate state of the search (plan-)space through which all solutions to the planning problem must pass. Thus, starting a DPOCL planner with a non-initial plan controls the form of the solutions the planner generates, since it effectively prunes a portion of the search space that would be accessible from the (more general) initial plan. Conceptually, the pruned portions represent plans that would be incompatible with the observations that the plan recognizer takes as *ground truth* of activity in a domain.

The compilation procedure we propose is described in the algorithm listing. The algorithm begins by initializing the input plan if it is empty (Step 0), which will typically only occur on the initial call to the algorithm. The recursive termination criteria is then checked (Step 1): if the plan so far is inconsistent (i.e. if the orderings lead to a cycle or the bindings afford a mapping between opposite literals), the algorithm fails; otherwise, if there are no further observations to compile, it returns the DPOCL plan. However, if there are further observations to check (Step 2), then it dequeues the next observation in the input sequence, and does one of three things: (A) instantiates a new recipe with a pseudo-step that could be swapped for the observation (and swaps the pseudo-step), (B) finds an instantiated recipe that contains a pseudo-step that could be swapped for the observation (and swaps the pseudo-step), or (C) simply adds the observation to the plan. In all cases, the algorithm ensures that if there are already other observations in the plan that temporally preceded (*i.e.* appeared before) the observation under consideration within the input sequence, then ordering constraints are added to the plan to reflect as such (Step 3). The procedure then makes a recursive call to handle the rest of the observations (Step 4). This algorithm produces a *consistent*, but not *complete* DPOCL plan, as evidenced by the fact that Step 1 does not admit inconsistent plans, and returns failure if such a plan is detected.

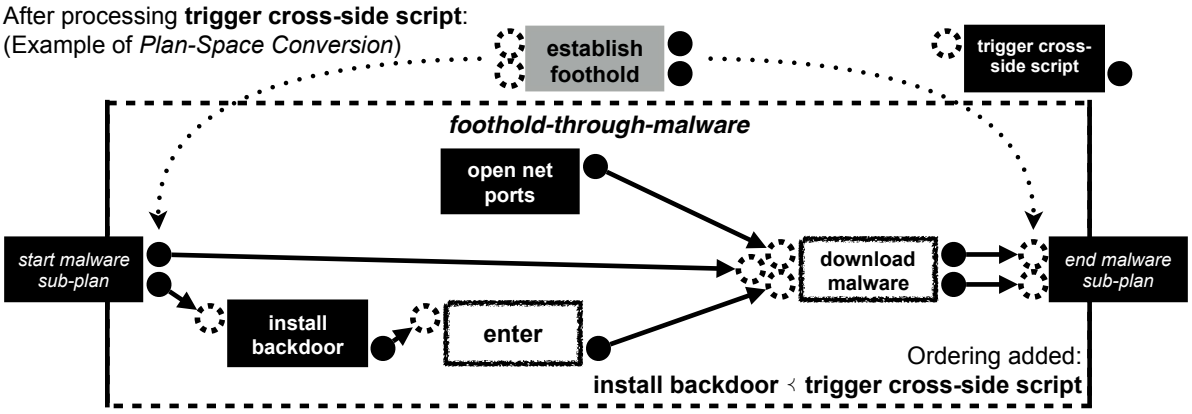
Given a sequence of observations and a domain theory, a DPOCL plan recognizer is a system that: (1) compiles observations into a DPOCL plan that represents them as in Algorithm , and (2) uses the compiled DPOCL plan as input to a DPOCL planner (Young, Pollack, and Moore 1994), which further refines the input plan to make the plan complete. The compilation process in our technique will produce a consistent DPOCL plan with flaws that will be refined (Kambhampati, Knoblock, and Yang 1995) by the DPOCL planning process to produce a *recognized plan*. If a flaw can never be refined at the planning stage, the plan recognition system will produce no recognized plan.



(a) Illustration of DPOCL-OBSERVATION-COMPILATION Step 2.A.



(b) Illustration of DPOCL-OBSERVATION-COMPILATION Step 2.B.



(c) Illustration of DPOCL-OBSERVATION-COMPILATION Step 2.C.

Figure 4: Evolution of a partial decompositional plan during observation compilation. The above sub-figures illustrate what happens after dequeueing the first (a), second (b), and third (c) observation illustrated in Figure 3. In each figure, the gray box "establish foothold" represents a composite step, black boxes represent primitive steps, and white boxes represent pseudo-steps of the decomposition schema "foothold-through-malware" (itself denoted as a partially-dotted bounding box). The recipe being applied in this observation compilation process is $R =$ ("establish foothold", "foothold-through-malware") and is represented implicitly. The dotted arrows that go from the composite action to the dummy actions "start malware sub-plan" and "end malware sub-plan" represent the decomposition link that records how "establish foothold" is realized via the (decomposition) sub-plan "foothold-through-malware." The black arrows that go from effects of actions (black circles) to preconditions of other actions (dotted circles) represent causal links.

Algorithm A DPOCL Observation Compilation Algorithm, which compiles a sequence of observed agent actions into a *consistent*, but not *complete* DPOCL plan that represents them. During generation, the algorithm can apply recipes that must be resolved during DPOCL plan recognition planning.

Input:

- A planning problem $\mathcal{P} = (\mathcal{D}, i, g)$, where $\mathcal{D}$ is a domain model that specifies the available logical fluents $\mathcal{L}$, composite action schemata Λ, and decomposition schemata Δ; i is a set of fluents that define the initial state; and g is a set of fluents that define the goal state the agent is assumed to pursue.
- $\pi = (\mathcal{S}, B, \prec, L_C, L_D)$, a DPOCL plan.
- $O = [o_0, \ldots, o_n]$, a sequence of observations such that $\forall o_i \in O$, o_i's action schema exists in $\Lambda \in \mathcal{D}$.

Output: $\pi = (\mathcal{S}, B, \prec, L_C, L_D)$, a DPOCL plan that represents the observed actions.

procedure DPOCL-OBSERVATION-COMPILATION($\mathcal{P}, \pi, O$)

0. Initialization If π is empty, initialize it as: $S = \{s_0, s_\infty\}$, $\prec = \{s_0 \prec s_\infty\}$, $B = L_C = L_D = \{\varnothing\}$, where s_0's effects are the fluents in i, and s_∞'s preconditions are the fluents in g.

1. Termination: If B or $\prec$ is inconsistent, **fail**. Else, if O is empty, return π.

2. Observation Compilation: Let $o_i \Leftarrow O.dequeue()$.
Nondeterministically **choose** one of the following:

A. **Recipe Recognition**

A.1 **Recipe Selection:** Nondeterministically **choose** a recipe $R = (\lambda, \delta)$, such that δ contains at least one pseudo-step $t \in \mathcal{S}^*$ of the same action type as o_i (if no such recipe exists, backtrack). Replace t/o_i in δ.

A.2 **Implied Step Addition:** Let π' be a DPOCL plan $(\mathcal{S}', B', \prec', L'_C, L'_D)$, where:

- $\mathcal{S}' \Leftarrow \mathcal{S} \cup \mathcal{S}^*_\delta \cup \{\lambda\}$ ▷ Including pseudo-steps.
- $B' \Leftarrow B \cup B_\delta$
- $\prec' \Leftarrow \prec \cup \prec_\delta$
- $L'_C \Leftarrow L_C \cup L_{C_\delta}$
- $L'_D \Leftarrow L_D \cup \{(\lambda, t^\delta_0, t^\delta_\infty)\}$

B. **Recipe Matching**

B.1 **Recipe Assignment:** Let π' be a DPOCL plan $(\mathcal{S}', B', \prec', L'_C, L'_D)$ such that $\pi' = \pi$. Nondeterministically **choose** a pseudo-step $t \in S'$ of the same action type as o_i (if no such step exists, backtrack). Replace t/o_i in $\mathcal{S}'$, and let $B' \Leftarrow B' \cup \{o_i\text{'s bindings}\}$.

C. **Plan-Space Conversion**

C.1 **Observation Transformation** Let π' be a plan $(\mathcal{S}', B', \prec', L'_C, L'_D)$, where:

- $\mathcal{S}' \Leftarrow \mathcal{S} \cup \{o_i\}$
- $B' \Leftarrow B \cup \{o_i\text{'s bindings}\}$
- $\prec' \Leftarrow \prec \cup \{(t_0 \prec o_i), (o_i \prec t_\infty)\}$
  ▷ Order it between the initial and final steps.
- $L'_C \Leftarrow L_C$, $L'_D \Leftarrow L_D$

3. Relative Observation Ordering:
If there exists an observation $o_{i-1} \in \mathcal{S}'$ that is sequentially prior to o_i in O, then let $\prec' \Leftarrow \prec' \cup \{(o_{i-1} \prec o_i)\}$.

4. Recursive Invocation:
Call DPOCL-OBSERVATION-COMPILATION($\mathcal{P}, \pi', O$).

end procedure

Comments on our Compilation Algorithm The compilation algorithm was intentionally designed to be very general, because different applications could require different control strategies for deciding how to perform the various open decisions.

Firstly, the decision of Step 2 (Observation Compilation) is left open; that is, observation compilation can delegate to any of three sub-procedure branches as discussed earlier. Each branch adds an observation to the plan in a different way; namely, by applying a new recipe, by using an existing recipe, or by not using a recipe at all. Regardless of which is desirable for the application context, our algorithm can accommodate using all three in an interleaved fashion.

Secondly, the decision of Step A.1 (Recipe Selection) is left open; that is, recipe selection can place the observation in any recipe that is *applicable* (*i.e.* which contains a decomposition schema with a pseudo-step of the observation's action type). An example of using a particular control scheme is the work by Lesh, Rich, and Sidner (1999), which uses the concept of a *user's focus* during human-computer interaction to limit the range of recipes available to predict the user's intended plan.

Thirdly, the decision of Step B.1 (Recipe Assignment) is left open; that is, recipe assignment can replace any pseudo-step of the same action type as the observation. This would be useful to explore various alternatives that could explain the observed action, since different pseudo-steps of the same type might serve different purposes in the schema.

Because the compilation does not produce a complete plan, the DPOCL plan recognition process is dominated by the performance of the DPOCL planning algorithm. It is possible that decisions taken at the observation compilation phase make it more difficult for the plan recognition process at the planning phase. If the flaws that are introduced by the recipes are impossible to refine, the plan recognition process would have to backtrack beyond the planning phase to the compilation phase, which makes the process potentially computationally complex. In future work, we hope to identify informed control strategies for Steps 2, A.1, and B.1 that minimize the amount of overall work needed to find a plan to explain an agent's observed actions. Observation compilation is executed once for every goal the agent is assumed to have. Thus, the identification of informed control strategies is a fruitful area of future work for this model of plan recognition.

Discussion

This paper presents a technique for combining recipe-based and domain theory-based reasoning during plan recognition.

Importantly, the work we propose here is a general case of *plan recognition as plan-space planning*, which itself has not previously been discussed. If we disallow composite action schemata and introduce no decomposition schemata, then our compilation algorithm would have no recipes to apply, thereby fixing the choice of Step 2 to the C branch. Transitively, the DPOCL planner would fall back to POCL planning as discussed by Young, Pollack, and Moore (1994). Thus, our formulation here also paves the

way for researchers interested in plan recognition via plan-space planning who operate in non-hierarchical domains.

We do not assume that the agent is necessarily optimal, a condition that is typically operationalized in terms of the minimum cost of the recognized plan. However, what that means for our model is that the DPOCL planner could potentially fix flaws introduced during compilation, but not reuse compiled information in the computation of the solution plan. Plainly, the steps added to the plan during observation compilation would appear in the solution plan only because they were observed to have occurred, not because they are on the agent's plan to solve an assumed goal (unlike models which assume optimality). To account for this, we assume that the agent acts with *intent*, a condition that must be enforced by the DPOCL planner. That is, the effects of steps added to the compiled plan must be intended, as defined by Young and Moore (1994): informally, an effect is intended if it is used in a causal link, and the step that asserts that effect is a causal ancestor of the final step of the plan. To assume that the agent must always act with intent constrains the kind of activity our model can effectively recognize; the actions of exploratory agents, for example, will likely not be well-recognized in our model (for which additional modifications would be needed).

Another difference relative to prior work of our plan recognition model is the form of the recognized plan that is output. Due to the plan-space oriented nature of the DPOCL planning process, the output of the DPOCL plan recognition pipeline is a partial plan, which represents a family of plans that satisfy the constraints identified during the refinement process (Kambhampati, Knoblock, and Yang 1995). This is unlike most prior work on plan recognition, which only produces a single recognized plan as output. A notable exception is the work by Geib and Goldman (2011), which admits the possibility of an agent pursuing multiple plans; their work is unlike ours because we produce an artifact that is a family of totally ordered and ground plans (all those compatible with the partial plan), whereas their formalism computes a probabilistic distribution of potentially pursued totally ordered plans.

In our example, we constrained the observation compilation to work over primitive steps exclusively. However, there is nothing in our formalism that would prevent the compilation from working with recognized composite steps. If the planning domain contains decomposition schemata that contain composite steps as part of the sub-plan, then the same reasoning procedure can introduce a higher level of hierarchy into the recognized plan. One limitation, however, is that these added levels of hierarchy must be triggered by the observation of a composite step. That is, once a recipe's decomposition schema has been added to the partial plan by our algorithm, the procedure does not further check to see if that recipe's composite action schema can itself form a part of another decomposition in the partial plan. An artificial solution to this limitation is to enqueue the recipe's composite action schema into the queue of observations, but we have not explored the potential ramifications of doing so.

The immediate next step is to empirically evaluate our plan recognition system. While this plan recognition model can be built atop the Longbow decompositional planning system (Young 1994), we are interested in extending the top POCL performer of the 3rd International Planning Competition (Long and Fox 2003), VHPOP (Younes and Simmons 2003) to handle decompositional reasoning. VHPOP's built in support for heuristic guidance of plan-space search suggests that it would be amenable to incorporating recently-developed hierarchy-related heuristics (e.g., Bercher, Keen, and Biundo, 2014) that allow us to further speed up total recognition time. Empirical evaluation, however, remains challenging due to lack of agreed upon domains and metrics for plan recognition (Goldman et al. 2011).

Conclusion

The work we present proposes (to our knowledge) the first synthesis of recipe-based and planning-based plan recognition in a unified knowledge representation, and sound and (primitive) complete decompositional planning reasoning procedure. Our synthesis is straightforward and it enables narrowing the gap between the existing and well-established community of recipe-based plan recognition researchers, and the more nascent but rapidly growing community of planning-based plan recognition researchers.

Acknowledgements

We thank Miquel Ramírez as well as the members of the Liquid Narrative Group at NC State for their insightful comments on earlier versions of this paper.

References

Amir, O., and Gal, Y. 2011. Plan recognition in virtual laboratories. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 2392–2397.

Avrahami-Zilberbrand, D., and Kaminka, G. A. 2005. Fast and complete symbolic plan recognition. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, 653–658.

Bercher, P.; Keen, S.; and Biundo, S. 2014. Hybrid planning heuristics based on task decomposition graphs. In *7th Annual Symposium on Combinatorial Search*.

Buchanan, B. 2016. The life cycles of cyber threats. *Survival* 58(1):39–58.

Carberry, S. 2001. Techniques for Plan Recognition. *User Modeling and User-Adapted Interaction* 11(1-2):31–48.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. UMCP: A Sound and Complete Procedure for Hierarchical Task-Network Planning. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems*, 249–254.

Geib, C. W., and Goldman, R. P. 2001. Plan recognition in intrusion detection systems. In *Proceedings of the DARPA Information Survivability Conference & Exposition II*, volume 1, 46–55. IEEE.

Geib, C., and Goldman, R. 2011. Recognizing plans with loops represented in a lexicalized grammar. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*, 958–963.

Golan, A., and Lumsdaine, R. L. 2016. On the construction of prior information – an info-metrics approach. In *Essays in Honor of Aman Ullah (Advances in Econometrics, Volume 36)*. Emerald Group Publishing Limited. 277–314.

Goldman, R. P.; Geib, C. W.; Kautz, H.; and Asfour, T. 2011. Plan Recognition (Dagstuhl Seminar 11141). *Dagstuhl Reports* 1(4):1–22.

Hayes-Roth, B., and Hayes-Roth, F. 1979. A Cognitive Model of Planning. *Cognitive Science* 3(4):275–310.

Kambhampati, S.; Knoblock, C. A.; and Yang, Q. 1995. Planning as refinement search: a unified framework for evaluating design tradeoffs in partial-order planning. *Artificial Intelligence* 76(1):167–238.

Kautz, H. A., and Allen, J. F. 1986. Generalized Plan Recognition. In *Proceedings of the 5th National Conference on Artificial Intelligence*, 32–37.

Lekavỳ, M., and Návrat, P. 2007. Expressivity of STRIPS-like and HTN-like planning. In *Proceedings of the KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications*, 121–130. Springer.

Lesh, N.; Rich, C.; and Sidner, C. L. 1999. Using plan recognition in human-computer collaboration. In *Proceedings of the 7th International Conference on User Modeling*, 22–32.

Long, D., and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research* 20:1–59.

Ramírez, M., and Geffner, H. 2009. Plan Recognition as Planning. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 1778–1783.

Ramırez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, 1121–1126.

Ramirez, M., and Geffner, H. 2016. Heuristics for planning, plan recognition and parsing. *arXiv preprint arXiv:1605.05807*.

Sacerdoti, E. D. 1975. The nonlinear nature of plans. In *Proceedings of the 4th International Joint Conference on Artificial Intelligence*, 206–214.

Sohrabi, S.; Riabov, A. V.; and Udrea, O. 2016. Plan Recognition as Planning Revisited. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, 3258–3264.

Sukthankar, G.; Geib, C.; Bui, H. H.; Pynadath, D.; and Goldman, R. P., eds. 2014. *Plan, Activity, and Intent Recognition: Theory and Practice*. Elsevier.

Weld, D. S. 1994. An Introduction to Least Commitment Planning. *AI Magazine* 15(4):27.

Younes, H. L., and Simmons, R. G. 2003. VHPOP: Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research* 20:405–430.

Young, R. M., and Moore, J. D. 1994. DPOCL: A Principled Approach to Discourse Planning. In *Proceedings of the Seventh International Workshop on Natural Language Generation*, 13–20. Association for Computational Linguistics.

Young, R. M.; Pollack, M. E.; and Moore, J. D. 1994. Decomposition and Causality in Partial-order Planning. In *Proceedings of the Conference on Artificial Intelligence Planning Systems*, 188–194.

Young, R. M. 1994. A developer's guide to the Longbow discourse planning system. Technical report, University of Pittsburgh.