Pattern Based Temporal Inference In Forward Search Temporal Planning

Atif Talukdar, Maria Fox, Derek Long
Department of Informatics
King's College London
Strand, London
WC2R 2LS, UK
firstname.lastname@kcl.ac.uk

Abstract

Temporal Planners are developed to address problem domains of a more realistic nature, where time is a factor. Some of the most difficult problems to solve are those containing required concurrency since the planner must execute actions in parallel. Forward search planners are generally good at finding plans using search with heuristic guidance. However, temporal planners still face limitations in their use of inference. This paper discusses patterns of required concurrency and how we can solve problems of this nature, with reduced search and more inference, in a forward search framework.

1 Introduction

In temporal planning, some of the most challenging problems are those that have complex temporal interactions between actions. An example of such interaction are where actions must occur concurrency for a valid plan to be produced. Problems with required concurrency are where all solutions to the problem must have two or more actions execute concurrently (Cushing et al. 2007). A problem of this type has no sequential plans that reach the goal.

Where there is required concurrency between two actions, it means that neither action can execute separately. This phenomenon provides an opportunity to perform inference in forward search temporal planners, where the use of inference is limited. This paper presents our work to automate the detection of required concurrency between pairs of actions, by analysing the domain structure. We then explain how we use these detections for doing temporal inference. During the detection, we extract the information required for inference and propagate it throughout the planning phase. Our work currently focusses on detecting required concurrency between two actions. We handle this detection in temporal domains written according to the specifications of PDDL 2.1 (Fox and Long 2003) that use durative actions.

The patterns identified in this paper are based on those identified by Cushing et al.(2007). We build on previous work presented in (Talukdar 2016), that identified the atomic cases of these patterns, where the actions are non-parameterised and only one grounded instance of each action exists. The inferred temporal constraints for the parameterised versions of the patterns are currently the same as specified in (Talukdar 2016). We propose a systematic approach for incorporating the detection of these patterns and the triggering of the corresponding inferences in POPF (Coles et al. 2010). We have opted to use POPF as our initial planner to extend with our machinery, as it is a forward search temporal planner that utilises a partial order. The partial ordering component is important, since the inferences may require reordering of the actions in the current plan when the inference occurs. Throughout this paper, we refer to action template definitions specified in the domain as *operators* and refer to grounded instances of operators as *actions*.

We currently focus on *contingent required concurrency*. This is where given a problem to which there is a sequential solution and a path containing required concurrency; the planner will prefer the path with required concurrency to reach the goal.

2 Related Work

TPSHE (Jimnez, Jonsson, and Palacios 2015) is a planner that deals with required concurrency, however it is limited to handling cases in the form of a single hard envelope (Coles et al. 2009). This is the type of required concurrency we see in pattern A displayed in section 3.3.

Currently, inference has been well exploited in the use of temporal planners that are based on constraint programming (CP). CP based planners such as CPT (Vidal and Geffner 2004) and eCPT (Vidal and Geffner 2005) use inference as a strong pruning mechanism. C3 (Lipovetzky and Geffner 2009) is a planner that performs inference, which although is not complete, can solve a number of problems without backtracking, across a variety of domains.

3 Parameterised Patterns

In this section we illustrate the parameterised versions of the pattern structures presented in (Talukdar 2016). We currently restrict the cases of required concurrency that we handle, to those where all predicates that are part of a pattern structure, can only have a single achieving operator in the

domain. Each action pair displayed in this section is an example instance of each pattern of required concurrency with parameters, that we handle. The labels at the top of each pattern example, with either "Unique Inference Pattern Instance" or "Choice Inference Pattern Instance", are specific to the example shown. It is possible for instances of all pattern types, to have either unique or choice inferences possible. This depends on the parameters that are part of the predicate(s) in the pattern structure, in relation to the parameters of the inferred action. Unique and choice inferences are explained in section 4.3.

3.1 Managing the Complexity of Parameterised Patterns

Actions with parameters are considered a basic and important component of domain modelling; it allows us to model multiple instances of operators in the domain, using objects defined in the problem. Whilst this is of benefit in modelling, in the case of doing inference it provides a challenge for the planner. This is since there are potentially many grounded actions that could be inferred to satisfy the constraints of the required concurrency in a pattern instance.

Consideration of parameters means that for any pattern detected in the domain, there could be many grounded instances of that pattern, in the same way that an operator may have many grounded actions. We use an approach that avoids constructing grounded pattern instances of this kind, and saves the planner from needing to search over a set of grounded pattern instances when an inference is triggered. We do this by recording only one instance of a particular pattern type that includes the same two operators. We record the parameter indexes that need to match between the two operators for their grounded counterparts. We then go through the grounded actions and record for each pattern instance, all of the grounded action IDs for the inferred operator; this occurs after the pattern instances are initially constructed, but before search for a plan begins. Using this approach, we avoid constructing multiple grounded pattern instances that each contain a different combination of the grounded action pairs. Another pattern instance containing the same two operators is created, if the pattern instance is of a different pattern type or the pattern instance is being recorded again using the other action as the trigger for inference.

We briefly discuss our method for identifying pattern structures and recording the necessary pattern information to perform inference during search. During a pre-search phase, we analyse the operators defined in the domain, and collect the predicates from the action precondition and effects lists, that could construct part of the structure for a pattern. These predicates are associated with the operators that contain them as either a precondition or an effect of a specific type. Instances of the pattern structure for each pattern type, A to G, are detected from comparisons of these predicates. If an instance of a pattern structure is found, then we search for two operators utilising the predicate(s) in the right combination of precondition and effect, in the set of associated operators collected previously. If this is confirmed, then a pattern instance has been detected. At this point the operators and predicate(s) making up the pattern structure are recorded as part of the pattern instance.

3.2 Binding Parameter Indexes

The parameters of an action include all of the parameters for each predicate that is included in the action's preconditions and effects. The subset of action parameters that are included in the predicate(s) that are in the pattern instance, are the ones we care about for parameter index binding between the two actions of the pattern instance. This is since the number of parameters, the order of the parameters, or the parameter names between the two actions in their definitions may be different. We therefore record by index position which arguments passed to the actions need to match, for the constraints of the required concurrency to be fulfilled. The indexes for the pattern structure predicate(s) as they are positioned in the first operator's parameter list are bound to the index positions of the same predicate(s) in the second operator's parameter list. These index bindings are stored along with the rest of the information for each pattern instance. Once a pattern instance is created it is stored in a map structure, using the trigger operator as the key. A grounded action of that operator will trigger the inference during search.

3.3 Pattern Structures

In this section, we present an example of required concurrency for each parameterised pattern type. The parameters that must match between the two actions in each pattern instance are colour coded. The index positions of these parameter pairings are denoted in the 'Index Bindings' box on the right of each figure. The remaining parameters may take any grounding with regards to the required concurrency.

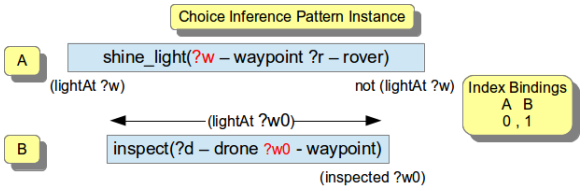

Figure 1: Pattern A pair. The 'inspect' action can only occur while 'shine_light' provides the resource 'light'.

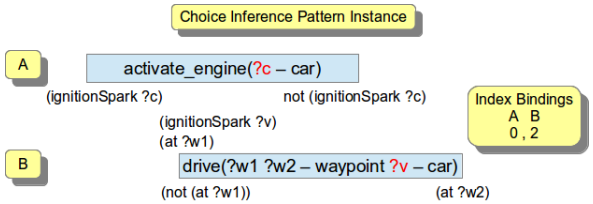

Figure 2: Pattern B pair. The 'drive' action can only start while 'activate_engine' of the same car provides the ignition spark for the 'drive' action to begin.

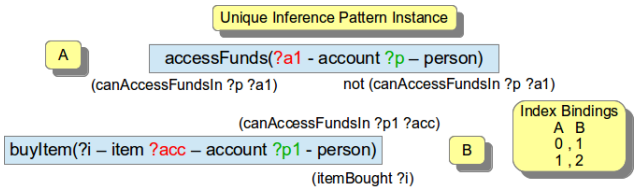

Figure 3: Pattern C pair. 'buyItem' action can only end while 'accessFunds' provides access to the account used to make payment for the shop items at the end of 'buyItem'.

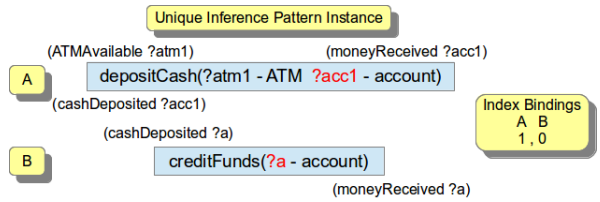

Figure 4: Pattern D pair. 'creditFunds' is only needed and must occur entirely within the duration of 'depositCash'.

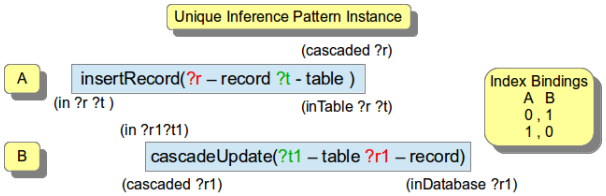

Figure 5: Pattern E pair. The start of 'insertRecord' allows 'cascadeUpdate' to start, which in turn allows 'insertRecord' to end.

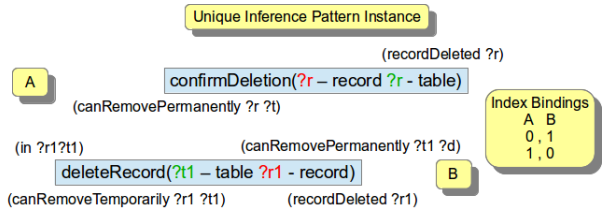

Figure 6: Pattern F pair. When a database record is deleted, it requires confirmation to verify that the record should be deleted.

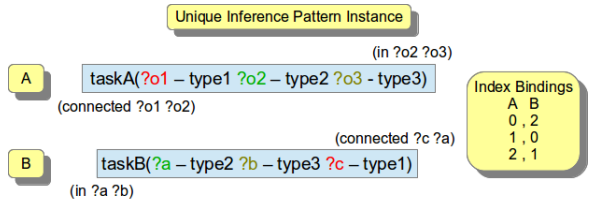

Figure 7: Pattern G pair. 'taskA' and 'taskB' can both start independently, but both must start before the other can end.

4 Inference

Currently, we restrict the use of our inference machinery to be used during search while in Enforced-Hill Climbing (EHC) (Hoffmann and Nebel 2011) mode only. We perform our pattern analysis on the domain as a pre-search phase, to prevent the cost of pattern detection being added to the search phase. During the search for a plan, the planner has access to the pattern information for all of the domain's patterns. The manner in which the pattern instances are created before search, allows the planner to efficiently perform temporal inference during search.

4.1 Inference in EHC

In EHC search, a greedy approach is used in an attempt to perform a faster search with less state exploration than in breadth-first search. Helpful actions are used to progress the search in standard EHC. Our approach is to trigger inferences during EHC search using either the same helpful actions added via search, or actions that have already been added via a previous inference, as would be the case in a chain of inference. We currently only allow grounded start actions to trigger inference. When a start action is selected to be added to the plan next, our inference machinery will perform a lookup in the map of trigger operators to pattern instances. If the root operator of the trigger action matches a key in the pattern instances map, then an inference has been triggered. The planner will visit the first element in the list of pattern instances triggered by this operator. The grounded actions associated with inferred operator in the pattern instance are then analysed one at a time, until one that has the same arguments as the trigger action, for the required parameter index bindings, is found. If an inferred grounded action is matched with the same required arguments, then it is viable for inference and it is added to a list of inferred actions. At each state, the inferred list is checked for actions to apply before performing the normal search process. If the inferred list is populated and contains an action that is applicable, then it is chosen to produce the successor state instead of using the standard EHC search approach.

Our approach modifies the process of action selection, to prioritise using inferred actions before standard helpful actions. Our machinery will check if the inferred action is applicable and if so, will select it instead of the standard helpful action that is otherwise chosen by EHC. Currently, if a pattern instance is triggered but the action added to the inferred list is not applicable, then this inference will not be applied. The next pattern instance in the list mapped to the trigger operator will be selected, and the process is continued to find an action that will satisfy the constraints of the required concurrency attached to the trigger action. If there are no other pattern instances or none of the inferred actions are yet applicable, then the helpful action that would have been added via normal search is applied. The trigger action would either have to be removed or scheduled later. If removed, the inference can be applied later on in the search process, if the trigger action is added again. .

4.2 Inferred Temporal Constraints and Inference Power

Table 1 displays the actions that must exist in the plan to trigger the inference, and the temporal constraints that are

inferred as a result of the inference for each pattern type. The constraints in brackets are inferred through transitivity. All of the inferred temporal constraints are determined when the trigger action(s) is added to the plan. Figure 8 displays the increase in the power of the inference for the patterns types and shows the strength of each pattern with respect to each other, based on the inferred constraints displayed in table 1. The letters in the 'Current Plan' and 'Constraints' in table 1 represent the actions that make up the pair in a pattern. The '$\vdash$' symbol in subscript after the letter denotes the start of the action and the '$\dashv$' symbol represents the end of the action.

Pattern	Current Plan	Constraints Inferred
A	$A_{\vdash} < B_{\vdash}$	$B_{\dashv} < A_{\dashv}$ $(B_{\vdash} < A_{\dashv})$
B	$A_{\vdash} < B_{\vdash}$	$B_{\vdash} < A_{\dashv}$
C	$B_{\vdash}$	$A_{\vdash} < B_{\dashv}$ $B_{\dashv} < A_{\dashv}$ $(B_{\vdash} < A_{\dashv})$
D	$A_{\vdash}$	$A_{\vdash} < B_{\vdash}$ $B_{\dashv} < A_{\dashv}$ $(B_{\vdash} < A_{\dashv})$
E	$A_{\vdash}$	$A_{\vdash} < B_{\vdash}$ $B_{\vdash} < A_{\dashv}$
F	$A_{\vdash} \vee B_{\vdash}$	$A_{\vdash} < B_{\dashv}$ $B_{\dashv} < A_{\dashv}$ $(B_{\vdash} < A_{\dashv})$
G	$A_{\vdash} \vee B_{\vdash}$	$A_{\vdash} < B_{\dashv}$ $B_{\vdash} < A_{\dashv}$

Table 1: Inferred temporal constraints. First appeared in (Talukdar 2016).

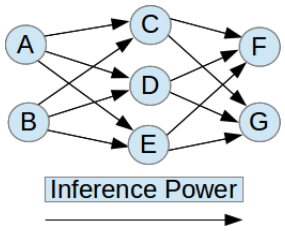

Figure 8: Increase in Power of Inference. First appeared in (Talukdar 2016).

4.3 Unique or Choice Inference Pattern Instances

When it comes to the search phase, if a particular pattern instance is triggered, it may be that there are multiple grounded actions that could be inferred as the action that needs to be added to the plan. However, it could also be the case that there is only one possible inferred grounded action that can be used. If all of the parameters of the inferred operator are parameters that are part of the predicate(s) that make up the pattern structure, then there can only be one grounded action that can be inferred if the pattern is triggered during search. This is since the objects passed as the arguments for the inferred action will have to be the same as the objects passed as arguments for the trigger action; this is a unique pattern inference. This is also the case, if parameters for the inferred operator that are not part of the pattern structure or indeed all of its parameters, have only one possible instantiation. This would be due to a single object of the operator's parameter types being defined in the problem instance. The choice inference case exists if there are one or more parameters for the inferred action operator, that are not amongst the parameters in the predicate(s) of the pattern and there are multiple objects in the problem definition that could be used as arguments for those parameters. In this situation, the planner will have many grounded actions that could be used to satisfy the required concurrency constraints for the pattern.

4.4 Example of State Progression using Inference

Figure 9 displays an example of the inference that can be performed using our approach. The example pair of actions used are for the pattern D instance shown in figure 4. The inference is deemed valuable if the state produced by the last action in the inference or chain of inference, has a strictly lower heuristic value than the last state before the inference began. As we seen in figure 9, when an inference is triggered the planner should only navigate down that path in the search tree, not generating other alternative states. At the end of the inference, the state generated is evaluated as having a lower heuristic than the state before the inference began. As a result the planner carries on with its standard search strategy, as it did before the inference. If the heuristic value of the state generated at the end of the inference is higher, then the planner would need to backtrack to the state before the inference trigger action was added, and would need to choose an alternative path down the search space.

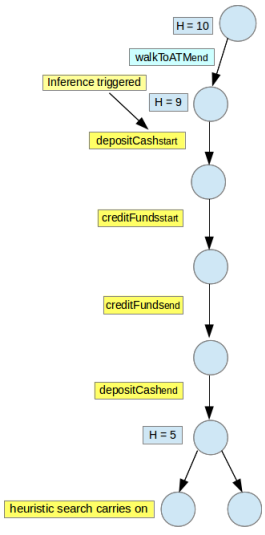

Figure 9: Progression through search space using inference.

5 Summary

We have illustrated the patterns of required concurrency that we handle, and the inferences we look to perform according to the pattern type. We have discussed how performing inference in forward search becomes a much more complex and challenging task for a planner to handle in the temporal propositional case, where actions have parameters. So far, POPF has been extended to perform the pattern detection we have described and we are currently in the process of implementing inference machinery for problems of required concurrency as they appear in the patterns presented in section 3.3. The approaches we have described are still in development and may be revised and refined as our understanding increases. Our goal is to exploit the power of inference in a forward chaining framework and to solve problems of required concurrency with less search and more inference.

References

Coles, A.; Fox, M.; Halsey, K.; Long, D.; and Smith, A. 2009. Managing concurrency in temporal planning using planner-scheduler interaction. *Artif. Intell.* 173(1):1–44.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS-10)*.

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. S. 2007. When is temporal planning really temporal? In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 1852–1859.

Fox, M., and Long, D. 2003. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)* 20:61–124.

Hoffmann, J., and Nebel, B. 2011. The FF planning system: Fast plan generation through heuristic search. *CoRR* abs/1106.0675.

Jimnez, S.; Jonsson, A.; and Palacios, H. 2015. Temporal planning with required concurrency using classical planning.

Lipovetzky, N., and Geffner, H. 2009. Inference and decomposition in planning using causal consistent chains. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*.

Long, D., and Fox, M. 2003. Exploiting a graphplan framework in temporal planning. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003), June 9-13, 2003, Trento, Italy*, 52–61.

Talukdar, A. 2016. Temporal inference in forward search temporal planning dissertation abstract. In *The 26th International Conference on Automated Planning and Scheduling*, 73–78.

Vidal, V., and Geffner, H. 2004. Branching and pruning: An optimal temporal POCL planner based on constraint programming. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, 570–577.

Vidal, V., and Geffner, H. 2005. Solving simple planning problems with more inference and no search. In *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*, 682–696.