A Limited Extension of PDDL for Planning with Non-primitive Actions

Biplav Srivastava
Email: sbiplav@in.ibm.com
IBM India Research Laboratory
Block 1, IIT Delhi, Hauz Khas, New Delhi 110016, India.

Abstract

When planning technology is considered for industrial applications, Hierarchical Task Network (HTN) planners are the popular choice because they provide mechanisms to the domain expert for encoding any available domain knowledge about the hierarchy and inter-relationship among actions. This information can be used by planning systems to potentially become more efficient as well as produce desirable plans. However, planners appearing in the international planning competitions (IPCs) have hitherto not been evaluated in domains containing non-primitive actions. The main reason has been that previous competitions had adopted the domain modeling maxim that *physics* and *not advice* should be encoded as part of the domain description. While the physics v/s debate will not end anytime soon, there is no doubt that the ability to model non-primitive actions in PDDL will help in introducing real-world domains into future competitions and making them more industry relevant.

In this paper, we propose a limited extension of PDDL to model non-primitive actions and describe a planning flow so that even without special provisions for such actions, a popular class of planners (specifically state-space planners) can still produce correct and desirable plans with little tweaking. This is made possible by a pre-processing step on non-primitive actions that creates new, primitive, *merged* actions that correspond to all the reductions of the non-primitive actions. The planner only needs to work with primitive actions and the generated plan, if using a merged action, is translated back into the primitive actions. We illustrate the approach by extending Sapa to handle non-primitive actions.

Introduction

Hierarchical Task Network (HTN) Planning (Erol1995) is a planning framework to reason with non-primitive and primitive actions. Non-primitive actions aggregate behaviour over a choice of primitive and other non-primitive actions and create a hierarchical network of sub-goals (tasks) to achieve. Actions are aggregated based on domain knowledge to reflect the domain expert's (user's) desire of acceptable solutions or provide heuristics to expedite solution finding. It is therefore no surprise that whenever planning technology has been considered for industrial applications, HTN planners have been the popular choice.

The International Planning Competitions (IPCs)(McDermott 2000; Bacchus 2000; Fox & Long 2002) have become a widely anticipated and significant event in the AI horizon. Besides the competition element (e.g., which is the fastest planner ?), each IPC has uniquely contributed to better understanding of the planning problems and solution methods - AIPS-98 saw the emergence of graphplan-based algorithms on the forefront, AIPS-00 saw the dominance of heuristic-based algorithms and AIPS-02 gave prominence to temporal-metric planning. HTN planning was considered as a possible track for the first competition(McDermott 2000) but later dropped due to lack of clarity on semantics and domain modeling philosophy. All competitions till now have adopted the domain modeling maxim that *physics* and *not advice* should be encoded as part of the domain description, i.e., the modeling of the domain should be independent of the intention of any agent in it. Though this philosophy seems reasonable from a competition standpoint because it provides no advantage to any specific type of planner, it has had the unintended effect that automated planners from the competition do not have a feature that is valuable for large-scale applications – they cannot handle non-primitive actions. It is worth noting that an HTN planner, SHOP(Nau et al 2000), has participated in the hand-coded tracks of AIPS-00 and AIPS-02 competitions but it was not evaluated for its ability to use advice. Non-primitive actions can incorporate both physics and any available advice from the domain. Consequently, when one wants to use a planner from the competition to solve problems in real applications, it is not clear which planner would be able to use advice better.

We take the position that while the physics v/s debate will not end anytime soon, the ability to model non-primitive actions in PDDL will help in introducing real-world domains into the competition and making it more industry relevant (Srivastava 2002b). Therefore, we propose a limited extension of PDDL up to level 3 to model non-primitive actions. We also describe a planning flow so that even without special provisions for such actions, a popular class of planners (specifically state-space planners) can still produce correct and desirable plans with little tweaking. This is made possible by a pre-processing step on non-primitive actions that creates new, primitive, *merged* actions that correspond to all the reductions of the non-primitive actions. The planner only needs to work with primitive actions and the generated plan, if using a merged action, is translated back into the non-primitive action. We have implemented the extensions in

Sapa and applied the resulting planner in many applications, e.g., query planning in bioinformatics(Srivastava 2002a) and web service composition.

Here is the outline of the paper: we start with a brief overview of HTN planning and propose the PDDL extensions. Next, we show how *Sapa*, a temporal planner, can be extended to support non-primitive actions. Finally, we summarize the main points and conclude.

HTN Planning

A planning problem P is a 3-tuple $\langle I, G, A\rangle$ where I is the complete description of the initial state, G is the partial description of the goal state, and A is the set of executable (primitive) actions. An action sequence S is a solution to P if S can be executed from I and the resulting state of the world contains G. A HTN planning problem(Erol1995; Kambhampati et al1998) can be seen as a planning problem where in addition to the primitive actions, the domain contains schemas which represent non-primitive (abstract and non-executable) actions and acceptable rules to reduce non-primitive actions to primitive and other non-primitive actions (hence an hierarchy of actions). The acceptable solutions to a HTN problem not only achieve the top-level goals but can also be parsed in terms of the non-primitive actions that are provided for the top-level goals(Barrett & Weld1994).

A HTN planner can transform any non-primitive action into executable actions by recursively applying the available reduction information from the schemas. We will restrict ourselves to state space planning where actions in a plan sequence occur contiguously, i.e., once two actions occur consecutively in a partial plan, no other action can subsequently come between them. In this case, one can interpret any reduction of a schema into an eventual *sequence* of primitive actions. We use this insight to pre-process (specifically, top-down parse) the schema into a set of *merged actions* that correspond to the sequential execution of the primitive actions in the final reduction.

Figure 1 gives the new planning flow. At the start, the primitive and non-primitive actions descriptions are read by a planner. The non-primitive actions are now used to generate *merged actions* which are essentially executable plan fragments. Both types of actions are together fed to a state-space planner. Though not necessary for completeness, the planner may want to differentiate between the actions in order to prefer *merged* actions because they result from user provided domain knowledge. The final plan is now post-processed so that it is in terms of the original inputs.

We now discuss this approach for PDDL(Fox & Long 2002) which is the current planning language adopted for IPC. It has 1 through 5 levels increasing in functionality: level 1 is for propositional STRIPS and ADL, level 2 allows numeric variables, level 3 supports durative actions with discrete effects while level 4 allows continuous effects, and level 5 has exogenous effects. We consider planning extensions at levels 1 through 3.

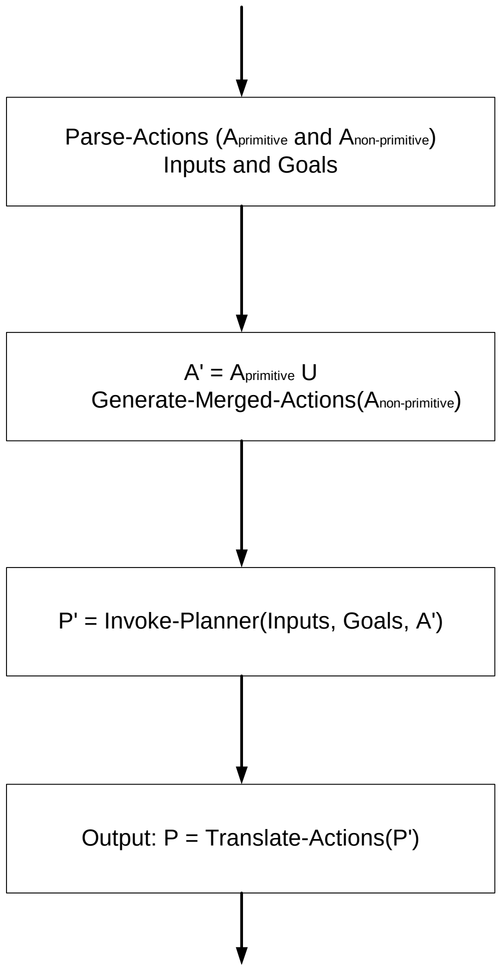
Parse-Actions ($A_{primitive}$ and $A_{non-primitive}$)
Inputs and Goals
A' = $A_{primitive}$ U
Generate-Merged-Actions($A_{non-primitive}$)
P' = Invoke-Planner(Inputs, Goals, A')
Output: P = Translate-Actions(P')

Figure 1: *Planning with proposed PDDL extensions.*

```
(:schema ⟨schema-name⟩
      :parameters
           ({⟨?var⟩ - ⟨var-type⟩})
      ;; Preconditions true at the start
      :precondition
           [(and] {⟨predicate⟩}[)]
      :effect
           [(and] {⟨predicate⟩}[)]
      :method
           [(choice] [{(sequence] { ⟨action⟩ } [)] } [)] )
```

Figure 2: *Schema specification in PDDL level 1. Fields in [ ] are optional while fields in { } are one or more.*

Extension of PDDL

We first discuss the proposed PDDL extensions. In the next section, we discuss how a non-HTN state space planner can adopt them to produce correct and desirable plans with minor tweaking.

Schema Specification

We extend PDDL to represent schemas through the $:schema$ construct which is described in Figure 2 for PDDL level 1 and Figure 3 for PDDL level 3. The $:precondition$ field is a place holder to specify necessary conditions for applying the schema. These are additional preconditions beyond those of the constituent primitive actions which should be true to apply the reductions. The $:effect$ field records the *primary effects*(Kambhampati et al1998) of the schema for which the merged action should be introduced into the plan. They take care of a basic concern in HTN planning that non-primitive actions should not be used to achieve secondary effects which will unnecessarily produce very complex plans. The $:method$ field specifies the choice in selecting a sequence of actions that are to be used for reductions. When there are more than one sequence, a *choice* delimiter is used. *We purposefully keep the specification of reductions simple so that merging of actions can be computed without any runtime information.* The $:duration$ field is applicable only at level 3 and records the minimum and maximum duration of each sequence of actions which are permissible while reducing this schema (if only maximum duration is specified, minimum is assumed 0).

As an example, in Figure 4, a schema is given in a variant of PDDL level 3. It encodes that the primary reason to make use of this schema is to prepare the data. In order to achieve the effect, there is a choice between three sequences of actions corresponding to the alternative reductions available.

Reduction of a Schema

We noted earlier that if one only considers state space planning, one can interpret reductions of a schema into an eventual *sequence* of primitive actions. Therefore one can preprocess (specifically, top-down parse) the schema into a set of *merged actions* that correspond to the sequential execution of the primitive actions in the final reduction.

```
⟨timed-predicate⟩ ::=
                        (at start ⟨predicate⟩) |
                        (at end ⟨predicate⟩) |
                        (over all ⟨predicate⟩)

(:schema ⟨schema-name⟩
      :parameters
           ({⟨?var⟩ - ⟨var-type⟩})
      ;; Static duration
      :duration
           ⟨max-dur⟩ | (⟨min-dur⟩, ⟨max-dur⟩)
      ;; Preconditions true at the start
      :precondition
           [(and] {⟨timed-predicate⟩}[)]
      :effect
           [(and] ⟨timed-predicate⟩} [)]
      :method
           [(choice] [{(sequence] { ⟨action⟩ } [)] } [)] )
```

Figure 3: *Schema specification in PDDL level 3. Fields in [ ] are optional while fields in { } are one or more.*

```
(:schema RESULT-PREPARE-SCHEMA
      :parameters
           (?d - data ?q - query)
      ;; Static duration
      :duration 4
      ;; Preconditions true at the start
      :precondition
           ()
      :effect
           (at end (prepared_data ?d))
      :method
           (choice
                 (sequence
                      (CLUSTER-DATA ?d)
                      (PREPARE-DATA ?q ?d))
                 (sequence
                      (ALIGN-DATA ?d)
                      (PREPARE-DATA ?q ?d))
                 (sequence
                      (SUMMARIZE-PUBLICATION ?d)
                      (PREPARE-DATA ?q ?d))))
```

Figure 4: *An example schema.*

Figure 5 describes a procedure to create such actions in a top-down manner. It recursively simulates the execution of each reduction and collects the specification of the top-level action. The merged actions are like primitive actions except that they only record primary effects for which they will be used during search. The duration specification of the schema is interpreted as a filtering criteria to assert that only reductions which lead to action sequences in the permissible interval are valid. Thus, if the duration of a sequence lies outside the range, no corresponding merged action is created.

The merged actions along with the original primitive actions can be fed to any state-space planner for its normal execution. Given an initial and goal state, the planner can either choose merged actions to respect user's intent or select primitive actions to account for possible incompleteness in domain knowledge.

Specification of Value Preferences

In many applications, we found that the domain expert wants to specify an preference on the values a variable can take. Akin to domain axioms, we introduce a $:prefers$ construct to allow a user to specify her preference for a variable's values. This information is used while instantiating variables of a particular type in an action. In Figure 6 for example, while considering variables of $Type_1$, $value_{11}$ is the most preferred value followed by $value_{12}$, and so on.

Soundness and Completeness

We now show that the proposed extensions does not compromise on desirable planner properties of soundness and completeness. A planner is sound if it generates executable plans and it is complete if the planner will find a solution whenever one exists. If the state-space planner is originally sound and complete, the only complication is introduced by the merged actions that are produced as a result of schema reductions.

One can be assured of soundness if, as part of the schema elicitation process, a verification procedure is implemented that ensures that (a) all the primitive actions mentioned in the schema are declared, and (b) each action sequence in the reductions lead to some executable sequence of primitive actions. We assume that such a verification procedure will exist. Completeness is guaranteed in the new planner as long as no plan without the merged actions are pruned away. We assume that the new planner will not prune such a plan but only penalize it to be further down in the search queue through heuristic adjustments.

With these precautions in place, the new planner will be both sound and complete.

Case Study: Adapting *Sapa* to Plan with Non-primitive Actions

Sapa(Do & Kambhampati2001) is a heuristic forward state space (also known as forward chaining) planner that can handle actions with durations, metric resource constraints and temporal deadlines. It starts from the initial state and applies actions as they become applicable taking into account

```
Algorithm: Generate-Merged-Actions
Input:     Schema $s$
Output:    $M$ = [] ; set of primitive actions

1. For each action sequence $L_i$ in $s$, create action $M_p$
2.    $M_p$.name = Make-unique-name($s$)
3.    $M_p$.parameters = $s$.parameters
4.    $M_p$.precondition = $s$.precondition
5.    $M_p$.duration = 0
6.    $M_p$.primary = $s$.effect
7.    Let $X$ = $M_p$
8.    For each action $a_j$ in $L_i$
9.       For each merged action $M_q$ in X
10.         If $a_j$ is non-primitive,
11.            Let $W$ = Generate-Merged-Actions($a_j$)
12.            For each (non-empty) $a_k$ in $W$
13.               $M_r$ = duplicate($M_q$) ; Make a copy
14.               $M_r$ = Merge-Action($s$, $M_r$, $a_k$)
15.               $X = X \bigcup M_r$
16.            End-for
17.         Else
18.            $M_q$ = Merge-Action($s$, $M_q$, $a_j$)
19.            $X = X \bigcup M_q$
20.         End-if
21.      End-for
22.   End-for
23.   $M = M \bigcup X$
24. End-for


Algorithm: Merge-Action
Input:     Schema $s$,
           Currently-Merged-Action $M$,
           Action $a$
Output:    $M$; the updated merged action

1. $M$.duration = $M$.duration + $a$.duration
2. If($M$.duration $\ni$ [$s$.min_dur, $s$.max_dur])
      return []; merge invalid
3. $M$.parameter = $M$.parameter $\bigcup$ $a$.parameter
4. $M$.precondition= $M$.precondition $\bigcup$ $a$.precondition
                       - $M$.effect
5. $M$.effect = $M$.effect $\bigcup$ $a$.effect
```

Figure 5: *Procedure to produce primitive (merged) actions based on reductions in a schema.*

```
(:prefers ($\langle Type_1\rangle$ $\langle value_{11}\rangle$ $\langle value_{12}\rangle$ ...)
          ($\langle Type_2\rangle$ $\langle value_{21}\rangle$ $\langle value_{22}\rangle$ ...))
```

Figure 6: *Specification of value preferences.*

```
Algorithm: Heuristic-Adjust-Minimal
Input:     Partial plan, $P$
Output:    Adjusted heuristic value of the plan

1. $length$ = Sapa-heuristic($P$)
2. $solution$ = Sapa-relaxed-plan($P$)
3. Foreach merged action, $M_i$ in the
   $solution$ of relaxed problem
4.     Foreach action, $a_j$ constituting
       the merged action
5.         If $a_j \in solution$
6.             numMergeRedundantActions ++
7.         End-if
8.     End-for
9. End-for
10. $length$ = $length$ + numMergeRedundantActions
```

Figure 7: *Heuristic adjustment for potential non-minimal plans.*

their duration and when (either start or end of the duration) each effect becomes valid. In order to guide its search, Sapa builds a temporal relaxed planning graph, and uses action and resource measures to calculate heuristic distance to the goal.

We extended *Sapa* by introducing non-primitive actions into the domain actions, A, and providing reduction schemas for them. Moreover, we allow users to give preferences over parameter values of actions and schemas. We also modify the heuristic estimates to account for merged actions in the partial plan. The modifi ed planner was applied in many applications including query planning in bioinformatics (called *SHQPlanner*(Srivastava 2002a)) which we discuss here. In this domain, the user is a biologist who can have specialized (partial) domain knowledge about how a query should be resolved, e.g., a solution for protein search may be to fetch data, merge results, run a particular application, and show the result. At PDDL level 3, we can additionally leverage the duration modeling of actions so that the user can reason about query costs.

Heuristic Adjustments: *Sapa* uses heuristics based on actions and resource usage to guide its search. With merged actions also added to the set of original primitive actions, we had to account for the fact that the merged actions signify user intent. We ensure this by increasing the heuristic estimate of a plan by some ϵ for each effect supported by a primitive action in place of an available merged action.

We also wanted to discourage plans where primitive as well as the merged actions are present to provide the same effect because the plan could possibly be non-minimal. In Figure 7, a procedure is described to capture this requirement by increasing the heuristic value of such a plan by the number of potentially redundant primitive actions.

Demonstration: We have built a small bioinformatics data integration domain containing descriptions of actions that query gene expression, protein, publication and pathway sources, and run clustering, sequence alignment and text summarization applications. Using the value preference

```
⟨A0⟩. 0.0 – RETRIEVE-DATA (pir protein )
            :duration 2.0

⟨A1⟩. 0.0 – RETRIEVE-DATA (swiss-prot protein )
            :duration 2.0

⟨A2⟩. 2.0 – ALIGN-DATA (protein )
            :duration 1.0

⟨A3⟩. 3.0 – PREPARE-DATA (q1 protein )
            :duration 3.0

⟨A4⟩. 6.0 – VISUALIZE-RESULT (q1 protein )
            :duration 1.0
```

Figure 8: *A query plan without schemas.*

```
⟨A0⟩. 0.0 – MERGED:DATA-FETCH-SCHEMA:
            %RETRIEVE-DATA%ALIGN-DATA
            (pir protein )
            :duration 3.0

⟨A1⟩. 0.0 – MERGED:DATA-FETCH-SCHEMA:
            %RETRIEVE-DATA%ALIGN-DATA
            (swiss-prot protein )
            :duration 3.0

⟨A2⟩. 3.0 – PREPARE-DATA (q1 protein )
            :duration 3.0

⟨A3⟩. 6.0 – VISUALIZE-RESULT (q1 protein )
            :duration 1.0
```

Figure 9: *A query plan with schemas. The merged actions will be replaced with the corresponding action sequences in a post-processing step.*

mechanism, the user can provide information to the planner like query data source PIR only if a query on data source SWISS-PROT has failed.

Suppose the biologist wants to retrieve aligned protein data and see the matched sequences in a viewer. We can run *SHQPlanner* as a regular planner, i.e., *Sapa*, or as a HTN planner with the proposed PDDL extensions. In Figure 8, a query plan is shown for the problem where data is retrieved from the two protein sources and then they are aligned with each other. But if the user only wants to align proteins of each source respectively, it is much simpler to realize the requirement with a suitable schema(*DATA-FETCH-SCHEMA* here). Figure 9 has the resultant plan. The query plans are generated in a few milliseconds.

Conclusion

The ability to model non-primitive actions in PDDL is very important in introducing real-world domains into future IPCs and making them more industry relevant. Hierarchical task network (HTN) planning generally looks into planning with domain knowledge encoded in schemas and primitive actions. A typical algorithm considers non-primitive actions and their reduction schemas as part of the domain specifi - cation (i.e., the set of available actions) and generalizes the planning refi nements to handle non-primitive actions. However, we wanted to avoid algorithm level dependency in order to make the planning flow general for a large class of planners.

In this paper, we propose a limited extension of PDDL with the following salient features:

- We introduce non-primitive actions into the action domain model and provide an offline semantics of the reductions so that their resulting effect can be captured as *merged* actions which is computable without any runtime information. We currently restrict the semantics to state space refi nement but it already encompasses most of the planners fi elded in previous IPCs.
- We introduce a feature so that the user can encode their preference for values that a variable in the domain description can take.
- The primitive and merged actions are input to any state space planner as the action set for regular planning. The planner may differentiate between the actions to capture the user intent (encoded in the merged actions) but *the PDDL specification does not provide any advice of how*.
- As long as reasonable discipline is maintained in schema elicitation, the planner continues to be sound and complete.
- Any merged action in the fi nal plan is will have to be translated into the corresponding sequence of primitive actions.

We demonstrated the extensions in *Sapa* and showed the benefi t of the new planner by planning in a real-world application.

References

Bacchus, F. 2000. AIPS-00 Planning Competition. *At http://www.cs.toronto.edu/aips2000/*

Barrett, A., and Weld, D. 1994. Task Decomposition via Plan Parsing. *Proc. AAAI.*

Do, B., and Kambhampati, S. 2001. *Sapa: A Domain-Independent Heuristic Metric Temporal Planner.* Proc. European Conference on Planning.

Erol, K. 1995. *Hierarchical task network planning: Formalization, Analysis, and Implementation.* Ph.D. thesis, Dept. of Computer Science, Univ. of Maryland, College Park, USA.

Fox, M., and Long, D. 2002. AIPS-02 International Planning Competition. *At http://www.dur.ac.uk/d.p.long/competition.html*.

Fox, M., and Long, D. 2002. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Available at http://www.dur.ac.uk/d.p.long/competition.html*.

Kambhampati, S., Mali, A., and Srivastava, B. 1998. Hybrid planning for partially hierarchical domains. *Proc. AAAI*.

Kambhampati, S., and Srivastava, B. 1995. Universal Classical Planner: An algorithm for unifying state space and plan space approaches. *In New Trend in AI Planning: EWSP 95, IOS Press*.

McDermott, D. 2000. The 1998 AI Planning Competition. *AI Magazine, 21(2).*

Nau, D., Cao, Y., Lotem, A., and Muoz-Avila, H. 1995. SHOP and M-SHOP: Planning with Ordered Task Decomposition. *Tech. Report CS TR 4157, University of Maryland, College Park, MD, June, 2000.*

Srivastava, B. 2002a. Using Planning for Query Decomposition in Bioinformatics. *AI Planning & Scheduling (AIPS-02) Workshop on "Is There Life Beyond Operator Sequencing? – Exploring Real World Planning"*.

Srivastava, B. 2002b. Temporal Constraints and the Physics v/s Advice Issue from a Practical Perspective. *AI Planning & Scheduling (AIPS-02) Workshop on "Temporal Planning"*.