The Use of Condition Types to Restrict Search in an AI Planner

Austin Tate, Brian Drabble & Jeff Dalton
Artificial Intelligence Applications Institute
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
United Kingdom
A.Tate@ed.ac.uk, B.Drabble@ed.ac.uk & J.Dalton@ed.ac.uk

Abstract

Condition satisfaction in planning has received a great deal of experimental and formal attention. A "Truth Criterion" lies at the heart of many planners and is critical to their capabilities and performance. However, there has been little study of ways in which the search space of a planner incorporating such a Truth Criterion can be guided.

The aim of this document is to give a description of the use of condition "type" information to inform the search of an AI planner and to guide the production of answers by a planner's truth criterion algorithm. The authors aim to promote discussion on the merits or otherwise of using such domain-dependent condition type restrictions as a means to communicate valuable information from the domain writer to a general purpose domain-independent planner [1].

Introduction to Condition Typing

Research in AI planning has introduced a range of progressively more powerful techniques to address increasingly more realistic applications (Allen, Hendler & Tate 1990). A lesson learned in the expert systems and knowledge-based systems field is that it is important to make maximum use of domain knowledge where it is available in order to address many real problems. One powerful means of using domain knowledge to restrict and guide search in a planner is to recognise explicit precondition types, as introduced into Interplan (Tate 1975) and Nonlin (Tate 1977) and subsequently used in other systems such as Deviser (Vere 1981), SIPE–2 (Wilkins 1988) O-Plan (Currie & Tate 1991) and O-Plan2 (Tate, Drabble & Kirby 1994).

An explicit account of the *Goal Structure* or *teleology* of a plan can be kept in these systems. This records the causal relationships between actions in the plan and can show the intentions of the domain writer or planner in satisfying conditions on actions. In some circumstances, such domain knowledge can be used to prune the search of a planner. The information is provided to the planner via a planner's domain description language (e.g., Task Formalism – TF – in Nonlin and O-Plan). The domain writer takes the responsibility for a deliberate pruning of the search space or for providing preferences via condition types. This caused us to adopt the term *knowledge based planning* to describe our work.

Nonlin and O-Plan TF extends the notion of a precondition on an action and mates it with a "process" oriented view of action descriptions. A TF schema description specifies a method by which some higher level action can be performed (or higher level goal achieved). Each schema is thought of as provided by its own "manager". The schema introduces lower level actions under the direction of its manager and uses that manager's own resources. The schema may say that some specific sub-action is included in order to set up for some later sub-action as part of the overall task. In TF, such internally satisfied requirements in actions are specified as **supervised** conditions. The "manager" also relies on other (normally external) agents to perform tasks that are their own responsibilities, but affect the ability of this manager to do the task. These are given as **unsupervised** conditions. There are other conditions which the "manager" may wish to impose on the applicability of particular solutions (e.g.. don't try this method for house building if the building is over five stories tall). These are termed **holds** and **usewhen** conditions in different versions of Nonlin and are now called **only_use_if** conditions in O-Plan2.

Condition typing can be used to restrict search in a planner, but there is work to be done on how far this technique can be developed. It is often difficult for a domain writer to choose the correct type for a condition to most effectively restrict the search space while not over-indulging and throwing away plans which should be considered valid in the domain. Tool sup-

[1] O-Plan2 work is supported by the US Advanced Research Projects Agency (ARPA) and the US Air Force Rome Laboratory acting through the Air Force Office of Scientific Research (AFSC) under contract F49620-92-C-0042. The project is monitored by Dr. Northrup Fowler III at Rome Laboratory.

port to aid in the reliable modelling of large domains will undoubtably be needed. In practice, we have found that condition typing is an essential aspect of encoding realistic problems to an AI planner in order to reduce search spaces to a manageable level.

Other Related Work

The concept of providing explicit domain encoder input to guide planning has its roots in early research on the Planner language family. POPLER (Davies 1973) identified the search space implications of providing only a single type of "goal" which can either already be true or which can induce subgoaling to be made true. Interplan (Tate 1975) provided a simple facility to indicate that nominated conditions should not be sub-goaled upon. That is, that no method of *achieving* them should be introduced into the plan. Nonlin (Tate 1977) provided a comprehensive set of condition types as described earlier. These were used to restrict the options considered to satisfy a condition in the Nonlin "QA Algorithm". QA was a precursor to the Truth Criterion used in many planners which use a partial order plan representation and make use of Goal Structure or causal links to direct search. See (Tate 1993) for a historical perspective.

A more general condition satisfaction approach, not using such domain knowledge, is used in TWEAK based on Chapman's formalisation of the Modal Truth Criterion (MTC) (Chapman 1987). This approach does not address search control issues. Chapman's work provides a description of the search space, but not a specification of how to control or prune search in that space.

There has been little study of ways in which the search space of a planner incorporating a Truth Criterion can be guided. Drummond (Drummond 1993) argues that there has been too much concentration on planner aspects that deal with logically or syntactically complete condition achievement, and too little attention has been paid to other capabilities of practical planners such as Nonlin, SIPE-2 and O-Plan. These other capabilities include hierarchical expansion, a simple but effective resource allocation mechanism, and explicit languages to describe and allow for the protection of the a plan's causal structure (effects/conditions).

A number of researchers have pre-analysed operator information to guide search.

Collins and Pryor (Collins & Pryor 1993) provide the first critical analysis on the use of condition types intended to filter out options that would otherwise have to be considered by a planner. They conclude that in the majority of cases such filter conditions are misused and may not have the effect intended. Their arguments assume that changes to the set of operators available might invalidate domain modelling assumptions about the use of filters(true[2]), that most providers of systems employing condition types did not fully appreciate that use of filter conditions would restrict the search space (false), and that restricting the search space using such filter conditions is not useful due to the restrictions under which they correctly apply (false, their argument assumes that hierarchical modelling is not used properly in planning or that filter conditions can be "hierarchically promiscuous"[3] – they must not be). Although the Collins and Pryor critical analysis paper is flawed in making some of these assumptions, it is none-the-less a useful document in raising the issue of the validity or otherwise of utilising condition type information to restrict search in a planner and may start wider study and debate on whether such condition types are valid and useful. Unfortunately, the work takes too simplistic a view of how condition types (filter and otherwise) are already used in planning systems today.

It is hoped that the current paper goes some way towards providing an information base on which comment, study and analysis will be possible.

O-Plan2 Domain Description Language Task Formalism

TF is used by a domain encoder to give an overall hierarchical description of an application area by specifying the activities within the domain and in particular their more detailed representation as a set of sub-activities with ordering constraints imposed. Plans are generated by choosing suitable "expansions" for activities (by refining them to a more detailed level) in the plan and including the relevant set of more detailed sub-activities described therein. Ordering constraints are then introduced to ensure that asserted effects of some activities satisfy, and continue to satisfy, conditions on the use of other activities. Other constraints, such as a time window for the activity or resource usage required, are also included in the description. These descriptions of activities form the main structure within TF - the *schema*. Schemas are also used in a completely uniform manner to describe *tasks*, set to the planning system, in the same language. Other TF structures hold global information and heuristic information about preferences of choices to be made during planning.

O-Plan2 Triangle Model of Activity

O-Plan2 uses a hierarchical model of activity which gives emphasis to an owner's perspective of how an activity is performed and the environment in which it can

[2]Tool support may help in avoiding such domain encoding errors.

[3]Hierarchical promiscuity occurs when a domain modeller confuses the levels at which effects are introduced and conditions are required. This is especially problematic for the ways in which AI planners typically handle filter conditions.

be sanctioned, resourced and used. This is reflected in the "triangle" model of an activity (see Figure 1). The vertical dimension reflects activity decomposition, the horizontal dimension reflects time. Inputs and outputs are split into three principal categories (authority, conditions/effects and resources). Arbitrarily complex modelling is possible in all dimensions. "Types" are used to further differentiate the inputs and outputs and their semantics.

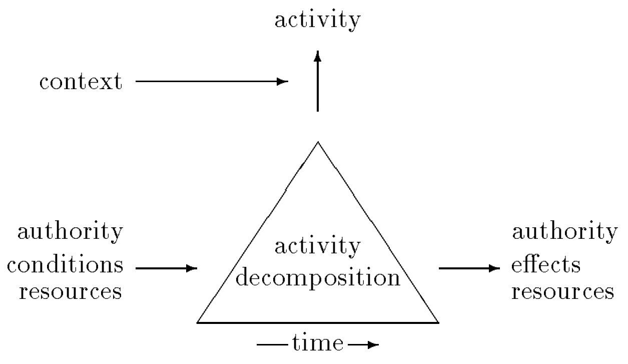

Figure 1: Triangle model of Activity

"Entry" to the model can be from any of three points in the triangle model. From the top vertex it is possible to ask for activity expansions or decompositions. From the right side of the triangle, it is possible to ask for activities satisfying or providing the output requirement (a desired effect or "goal", a required resource, or a needed authority). These two points are used mostly by our planners to date. The third point on the left side can reflect triggering conditions for an activity and will be needed when improved models of independent processes are used as in our Excalibur (Drabble 1993) extension to Nonlin. A "context" requirement permits use of each particular expansion or decomposition of an activity.

The triangle model of activity is a generalisation of process models used in many structured analysis and design techniques (SADT) such as IDEF, R-Charts, etc., and can be directly related to them.

O-Plan2 Condition Types

Condition typing allows relevant information to be kept about when, how and why a condition present in the plan has been satisfied and the way it is to be treated if the condition cannot be maintained. All condition statements appear in O-Plan2 Task Formalism action schemas. Conditions play a greater role in O-Plan than in previous planning systems since there is no *special* notion of *goal*. Nonlin (Tate 1977), NOAH (Sacerdoti 1977) and SIPE–2 (Wilkins 1988) style goal nodes in action expansions become simply **achieve** conditions in O-Plan. The **achieve** condition type is the only one on which sub-goaling is permitted.

Conditions are one of the most elaborate of all TF statements due to the variety of condition types identified as being needed for practical planning in O-Plan2. The "process" or "manager" view of hierarchical activity description used in Nonlin contributed the three basic condition types of **supervised**, **unsupervised** and **usewhen**. O-Plan research and applications experience identified the need to separate two different uses being made of the Nonlin **usewhen** condition type. This led to the introduction of **only_use_if** and **only_use_for_query**. A more flexible **achieve** condition definition was also required to remove temporal scope limitations on the ways in which earlier planners such as Nonlin could satisfy goals by adding new activities into a plan.

The O-Plan2 condition types are thus:

- **only_use_if** conditions provide an applicability check on the context in which a schema can be used. These are sometimes referred to as filter conditions.
- **only_use_for_query** conditions are used to make queries at a point in the plan to instantiate or restrict variables in a schema.
- **unsupervised** conditions must be satisfied at the required point, but it is assumed that, in circumstances in which the schema introducing such a condition is used, that the condition will have been satisfied elsewhere. Therefore, they act as a sequencing constraint.
- **supervised** conditions are satisfied directly within the schema containing them by the deliberate introduction of a suitable effect (or alternative effects) at an earlier point or by the direct inclusion of an action known to achieve the necessary effect (at some more detailed level in the action's decomposition). They may be used as a means to explicitly record a protection interval within the causal structure of a plan.
- **achieve** conditions can be satisfied by any means available to the planner including the addition of new actions into the plan.

Other condition types can be identified but the ones above have been found to be useful ways to extract knowledge from a domain writer in a communicable form that can be used to restrict search in an AI planner.

Condition typing helps direct the planning process, but it also requires that the domain encoder structures the hierarchy of the tasks or actions clearly. It forces checks to be made on processes or actions which should communicate with others – ensuring they actually do advertise their results through a common vocabulary.

O-Plan2 Plan Levels

Before describing condition types and their definitions, it is useful to describe how O-Plan2 uses hierarchical modelling levels in its operation.

Definition

Each action and effect is introduced at a single domain modelling level and higher level activities introduce activities and effects at the same or a lower level.

A plan level can be introduced for two distinct purposes:

1. For convenience of abstraction and aggregation.
2. To place an order on the commitments and constraints made during planning.

The numerical plan levels are assigned by the O-Plan2 TF compiler in quite an intuitive way. Level numbers increase as lower level, more detailed action and effect descriptions are given. However, there can be "loops" in the structure, such that some actions can expand recursively or may expand back to themselves via other schemas. In such cases, all the actions and effects in the "loops" are mapped to the same plan modelling level. The detailed way in which level numbers are assigned is as follows.

Each schema represents a way to perform the action indicated by its **expands** clause. The first word of the **expands** pattern is referred to as the *action name* of the schema. Each schema S links its action to a number of direct successor actions: the sub-actions listed in the schema's definition. These successor actions are normally at the next lower level (except when loops are involved). A further set of direct successors can be found by taking the action names of all schemas that have an **only_use_for_effects** that matches any **achieve** condition of S.

This will define a graph in which the action names are vertices and there is an edge from each action name to each of its directly reachable successors. The next step is to find the *strongly connected components* (SCCs) in this graph in order to build a new graph in which each SCC is treated as a unit. This new graph is acyclic and the level of an action can be found by taking the longest path to the SCC that contains it through this graph. This will also identify the level at which *effects* are introduced into the plan. The plan level mapper is sensitive to loops in the graph and the SCCs components represent such loops. Whenever you can get from A to B *and* from B to A in a directed graph, A and B are in the same SCC.

Condition Types for the Domain Writer

This section gives definitions of O-Plan2 condition types in terms of what information a domain writer providing a library of action or plan components can state, hopefully in an understandable way without knowledge of how the AI planner would go about using this in detail.

For each condition type used within O-Plan2 we provide below the following information:

- **Purpose**: This describes the condition type in domain terms for use by the domain encoder and describes the circumstances under which the condition type should be used.
- **Definition**: This describes the condition type in planner terms and describes in more detail how the planner goes about dealing with the condition type on behalf of the domain encoder.
- **Examples**: These clarify the use of each type.

Only_use_if

- **Purpose**: This is a filter condition on the applicability of a particular schema.
- **Definition**: It may be given on statements introduced as effects at a higher level or on the same modelling level as the schema introducing it.
- **Examples**: On static facts (those never refuted in the plan and referred to as **always** facts in O-Plan2):

  ```
  only_use_if {type_of soil} = sandy
  ```

  and on dynamic facts (whose value can change over time):

  ```
  only_use_if {apportioned_force ?regiment}
                                      = unallocated
  ```

  In the first example the condition would be used to allow a schema to be selected which was suitable for use if the soil type is sandy. In the second example, the condition would only allow the schema to be chosen if a particular force was available at this point in the plan. During the course of the plan the force's status may vary and with it the ability to use the schema.

Only_use_for_query

- **Purpose**: A query mechanism to establish current values for variables.
- **Definition**: It may be given on statements at a higher or on the same modelling level as the schema including it. There should *always* be an answer for such a query when it is evaluated at an appropriate level.
- **Examples**: On static facts:

  ```
  only_use_for_query {country_of ?city} = ?country
  ```

  and on dynamic facts:

  ```
  only_use_for_query {position_of ?robot}
                                         = ?location
  ```

  The first example would allow the country in which a city is located to be looked up. The second example allows the dynamic lookup of the position of the robot.

Unsupervised

- **Purpose**: Specifies a scheduling constraint on the schema which is (normally) satisfied externally. Exceptionally, it may also specify an internal ordering requirement within the schema making use of actions introduced for other reasons.
- **Definition**: It may be given on statements at the same or a higher modelling level if the condition is satisfied externally to the sub-actions of the schema or at the same or lower modelling level if the condition is satisfied from the sub-actions within the schema.
- **Example**:

```
unsupervised {status ground_buffer} = empty at 2
```

This would make a sub-action number 2 introduced by the schema occur after some other action in the plan which empties the ground_buffer.

Supervised

- **Purpose**: To protect an intended effect of some earlier sub-activity up to the point required.
- **Definition**: It may be given on statements at the same or on a lower modelling level as the schema including it.
- **Example**:

```
supervised {status ground_buffer} = full
                                   at 3 from 2
```

This would protect the ground_buffer as being full between the end of a schema sub-action number 2 (say where some data was captured into the ground_buffer) to the beginning of a later schema sub-action number 3 where the data might be used.

Achieve

- **Purpose**: To allow a condition to be satisfied by the optional inclusion of sub-activities.
- **Definition**: Specifies a requirement which the schema writer is optionally prepared to meet by adding new structure into the plan at the same or lower modelling level as the schema including the condition.
- **Example**:

```
achieve {in_position ?tool} = workbench at 1
```

This allows the required tool to be moved to the required place if it is not already there.

The following table summarises the ways in which each O-Plan2 condition type may be satisfied.

Condition	Levels Considered			Type
only_use_if	above	same		EXTERNAL
only_use_for_query	above	same		EXTERNAL
unsupervised	above	same		EXTERNAL or
		same	below	INTERNAL
supervised		same	below	INTERNAL
achieve		same	below	OPTIONAL

Condition Type Correspondence to Nonlin, SIPE-2 and ACT

Nonlin (Tate 1977) was the first Edinburgh planner to use the Task Formalism (TF) language and made use of **supervised**, **unsupervised** and **usewhen** condition types. It handled achievable conditions by including goal nodes in the action expansion.

The SIPE–2 planner (Wilkins 1988) also includes support for a number of condition types and is converging on similar types to those available in O-Plan2. A development of the SIPE–2 domain description language to link to work on the PRS (Procedural Reasoning System) reactive execution support system (Georgeff 1986) is now underway to create a shared domain description language called ACT. The following sections compare O-Plan2 condition type usage with the those used in Nonlin, SIPE–2 and ACT.

O-Plan2	Nonlin	SIPE-2	ACT
only_use_if	usewhen	precondition	precondition
only_use_for_query	none	none	setting
unsupervised	unsupervised	external	wait-until
supervised	supervised	protect-until	require-until
achieve at N	none	none	none
achieve after	goal	goal	achieve

only_use_if This is the same as Nonlin's **usewhen** (originally called **holds**). It is the same as a **precondition** in either SIPE–2 or ACT. The ***already** condition in later releases of Deviser (Vere 1981) performs the same function.

only_use_for_query In Nonlin and SIPE–2, such conditions were modelled as **usewhen** conditions or **precondition**s respectively and not treated separately. Nonlin or SIPE–2 treated a query condition in the same way as an **only_use_if** filter condition, except that it was assumed that variables would be bound by satisfying it. This incorrectly limits the range of legitimate solutions. As in O-Plan2, ACT separates query type conditions for clarity – calling them the **setting**.

unsupervised Later releases of SIPE–2 allow an **external** condition to give this capability. In ACT, this is called **wait-until**.

supervised The same as a **protect-until** in SIPE–2 and **require-until** in ACT.

achieve at N This imposes no restriction on the temporal scope of any activity inserted in the plan to satisfy the condition. There is no equivalent in Nonlin, SIPE–2 or ACT all of which make such temporal scope restrictions and thus restrict the domains which can be modelled.

achieve at N after <time point> This is an **achieve** with a temporal restriction on the points within the plan from which a contributor may be chosen to satisfy the condition. This is a more general way to describe Nonlin, NOAH and SIPE–2 goal nodes which appear in the expansion/decomposition part of their operator schemas. For these systems, the <time point> is restricted to be after the start of the time range of the expansion of the schema containing the condition. In ACT this is called **achieve** and has the same fixed restriction on temporal scope as Nonlin, NOAH and SIPE–2.

Summary

In large realistic domains, we believe that significant domain knowledge must be made available to a planner in order to reduce search spaces to a manageable level. One important way in which this can be done effectively is to get a domain writer to provide information about a domain from which we can extract instructions to the planning system about how to satisfy and maintain conditions required in the plan.

Condition types can be a valuable aid to providing knowledge about a domain to a domain-independent planner. Condition typing can successfully restrict the search for a plan, but there is work to be done on how far this technique can be developed. It is often difficult for a domain writer to choose the correct type for a condition to most effectively restrict the search space while not over-indulging and throwing away plans which should be considered valid in the domain. Improved planning knowledge capture aids now under development may assist in this process.

One aim of this paper has been to seek to separate the domain writer oriented description of condition types from the mechanisms used by a planner to satisfy, maintain and re-satisfy conditions.

By making this information more widely available, the authors aim to promote discussion on the merits or otherwise of using such domain-dependent condition type restrictions as a means to communicate information from the domain writer to a general purpose domain-independent planner. The control of planner search via condition types is worthy of a serious study in its own right, and could form an ideal Ph.D. topic.

Acknowledgements

We are grateful for discussions on condition types with Drew McDermott, Nancy Lehrer, Glen Reece, Mark Drummond, Subbarao Kambhampati, Dan Weld, Craig Knoblock and Louise Pryor, as well as from the comments of the reviewers and conversations with other members of the AI planning community. Sorry we had to be ruthless to achieve the final page count. The work reported here has benefited from input by talented researchers who have participated in the O-Plan and related projects. Thanks to David Wilkins who helped in checking the comparison to SIPE–2 and ACT usage of condition types. Errors remain ours.

References

Allen, J., Hendler, J. & Tate, A. 1990. *Readings in Planning*, Morgan-Kaufmann.

Chapman, D. 1987. Planning for Conjunctive Goals, *Artificial Intelligence*, Vol. 32, pp 333-377.

Collins, G. & Pryor, L. 1993. On the Misuse of Filter Conditions: A Critical Analysis, in *Current Trends in AI Planning*, (eds. Backström, C. & Sandewall, E.), IOS Press.

Currie, K.W. & Tate, A. 1991. O-Plan: the Open Planning Architecture, *Artificial Intelligence* Vol 51, No. 1, pp 49-86, North-Holland.

Davies, D.J.M. 1973. POPLER 1.5 Reference Manual, Theoretical Psychology Unit Report no.1., Department of Artificial Intelligence, University of Edinburgh.

Drabble, B. 1993. Excalibur: A Program for Planning and Reasoning with Processes, *Artificial Intelligence*, Vol. 62 No. 1, pp 1-40.

Drummond, M.E. 1993. On Precondition Achievement and the Computational Economics of Automatic Planning, in *Current Trends in AI Planning*, (eds. Backström, C. & Sandewall, E.), IOS Press.

Georgeff, M.P. & Lansky, A.L. 1986. Procedural Knowledge, in *Proceedings of the IEEE*, Special Issue on Knowledge Representation, Vol. 74, pp 1383-1398.

Sacerdoti, E. 1977. *A Structure for Plans and Behaviours*, Artificial Intelligence Series, North Holland.

Tate, A. 1975. Using Goal Structure to Direct Search in a Problem Solver. Ph.D. Thesis, University of Edinburgh.

Tate, A. 1977. Generating Project Networks, in Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-77), Cambridge, MA, USA.

Tate, A. 1993. The Emergence of "Standard" Planning and Scheduling System Components, in *Current Trends in AI Planning*, (eds. Backström, C. & Sandewall, E.), IOS Press.

Tate, A., Drabble, B. & Kirby, R. 1994. O-Plan2: an Open Architecture for Command, Planning and Control, in *Intelligent Scheduling*, (eds. Fox, M. & Zweben, M.), Morgan Kaufmann.

Vere, S. 1981. Planning in Time: Windows and Durations for Activities and Goals, *IEEE Transactions on Pattern Analysis and Machine Intelligence* Vol. 5.

Wilkins, D. 1988. *Practical Planning*, Morgan-Kaufmann.