CIFE CENTER FOR INTEGRATED FACILITY ENGINEERING

Knowledge Based Strategy Generalization

Tony Confrey and Barbara Hayes-Roth

TECHNICAL REPORT
Number 26

April 1990

Stanford University

If you would like to contact the authors please write to:

c/o CIFE, Civil Engineering,
Stanford University,
Terman Engineering Center
Mail Code: 4020
Stanford, CA 95305-4020

Abstract

WATCH is a system that performs knowledge level learning within the BB1 blackboard system environment. WATCH learns strategic knowledge by observing or *"watching"* the actions an expert uses to solve a problem. By its very nature however, the strategies WATCH finds are very problem specific and cannot be used to solve a very large class of problems. This paper describes MetaWATCH, an enhancement to the WATCH system that compares multiple WATCH output strategies in the context of the problem they solve and finds the common abstraction. This allows it to learn strategies applicable to a wider range of problems.

1 Introduction

WATCH [2] is a system which performs knowledge level learning within the BB1 [1] blackboard system environment. WATCH learns strategic knowledge by observing or *"watching"* the actions an expert uses to solve a problem. By its very nature however, the strategies WATCH finds are very problem specific and cannot be used to solve a very large class of problems. This paper describes MetaWATCH, an enhancement to the WATCH system that compares multiple WATCH output strategies in the context of the problem they solve and finds the common abstraction. This allows it to learn strategies applicable to a wider range of problems.

The motivation behind this work is to develop a tool that can be used to acquire the strategic knowledge required to control the operation of BB1 applications. It can thus be characterized as an experiment in both machine learning and knowledge acquisition.

This paper first describes strategic knowledge, what it is and how it is used. The environment in which MetaWATCH learns is then detailed. The WATCH system is then examined after which the operation of MetaWATCH is fully described. Finally a set of validating experiments performed on the system are listed as are the observations they give rise to. The paper ends with conclusions and some suggestions for future developments of the MetaWATCH system.

2 Strategic Knowledge

Strategic knowledge is the knowledge an expert uses to direct the course of problem solving. It is the ability to decide what actions need to be executed to accomplish a task and the order in which they should be performed to achieve an optimal solution.

Within an application strategic knowledge is used to control the flow of program execution and as such is often hard coded and treated as an implementation detail. For many knowledge based systems this hard coding takes the form of rule clustering, clause ordering within rules, conflict resolution strategies and other such techniques. Hard coding restricts these systems to using a single control strategy for all problems. However, often the problem solving strategy required depends on the problem being solved. For this reason making control

knowledge explicit can be of great benefit. BB1, described later, allows for dynamic control and makes the control knowledge explicit.

In many knowledge based systems, strategic knowledge dictates the best path to a solution. While the systems could perform without this knowledge the problem may become intractable or result in a less than optimal solution. Within BB1 knowledge about problem solving strategies is implemented as control and thus I will use the terms strategic and control knowledge interchangeably.

Strategic knowledge is one of the more difficult types of knowledge to acquire. It is often a lot easier for an expert to describe his domain knowledge, i.e. the objects he works with, their attributes, the relationships between them, the actions that can be performed on them and the effects of those actions, than to formalize how he actually solves problems within this domain. For this reason it is best to acquire strategic knowledge in the context of a problem solving session. WATCH attempts to learn the strategies used by an expert by examining the set of actions used to solve a problem within the context of the applicable domain knowledge.

3 The Learning Environment

This section describes the environment in which the WATCH systems operate (i.e. WATCH and MetaWATCH) and the ways in which this environment supports the learning task.

3.1 BB1

BB1 is a domain independent architecture based on the blackboard model. In this model, multiple, independent knowledge sources operate on objects stored in globally-accessible data structures called blackboards.

The objects on these blackboards are frames whose attribute slots can describe any property, behavior. These objects can be linked with user definable links along which inheritance paths can be defined.

Within the BB1 framework knowledge sources are also stored as objects on a blackboard. There are two types of knowledge source(KS). *Domain knowledge sources* perform problem solving actions. *Control knowledge sources* form a control plan which dictates

the order of firing of the domain KS's and can be used to implement arbitrarily complex problem solving strategies. Since these KS's can be operated upon in the same manner as any other blackboard object, control can be changed dynamically.

Every KS has preconditions that describe the context in which it should be considered and trigger conditions that describe the actual conditions under which it can be applied. On each execution cycle KS's whose trigger conditions are satisfied, in the context of its preconditions, become executable. The BB1 scheduler ranks the executable KS's in accordance with the control plan and the top ranked KS is executed (also called as "fired"). The actions associated with that KS, which may be domain or control actions, are then performed. These actions may alter domain objects or the control plan. The cycle of triggering, ranking and executing KS's then repeats.

3.2 Action description language

BB1 supports the use of languages that specialize BB1 for a particular class of problems. These languages formalize a set of primitive operators that facilitate solving the particular type of problem. There are a number of advantages to the use of these languages. They provide an underlying conceptual representation of the concepts involved in the domain and they facilitate reasoning by providing an unambiguous language for describing strategies and actions. However, from the point of view of WATCH, the most important properties of these languages are 1) to limit the experts actions to a finite set with which we can reason and 2) to provide us with a hierarchy showing the relations between these actions.

ACCORD is a domain-independent language that specializes BB1 for a class of arrangement problems that use constraints to position objects in a metric space. ACCORD supports a method for incremental assembly of objects to solve these arrangement/assembly problems. ACCORD provides a type hierarchy describing actions, events, objects, object modifiers and states and a linguistic framework for describing instantiations of these types.

To solve a problem within ACCORD the problem solver defines one or more *partial arrangements* each comprising a subset of the objects and constraints in the problem specification. One object is chosen as the *anchor,* and the other objects, the *anchorees,* are positioned relative to it. There are a number of actions which can be performed

on these objects to position them (see figure1). These actions apply constraints between objects to reduce their sets of legal locations. For example anchorees can be *anchored* to the anchor or *yoked* to another anchoree. Constraints are applied to reduce the legal locations for objects until a single set of feasible positions are found or objects locations have been reduced as much as possible. The results from different partial arrangements can then be integrated to form a complete solution.

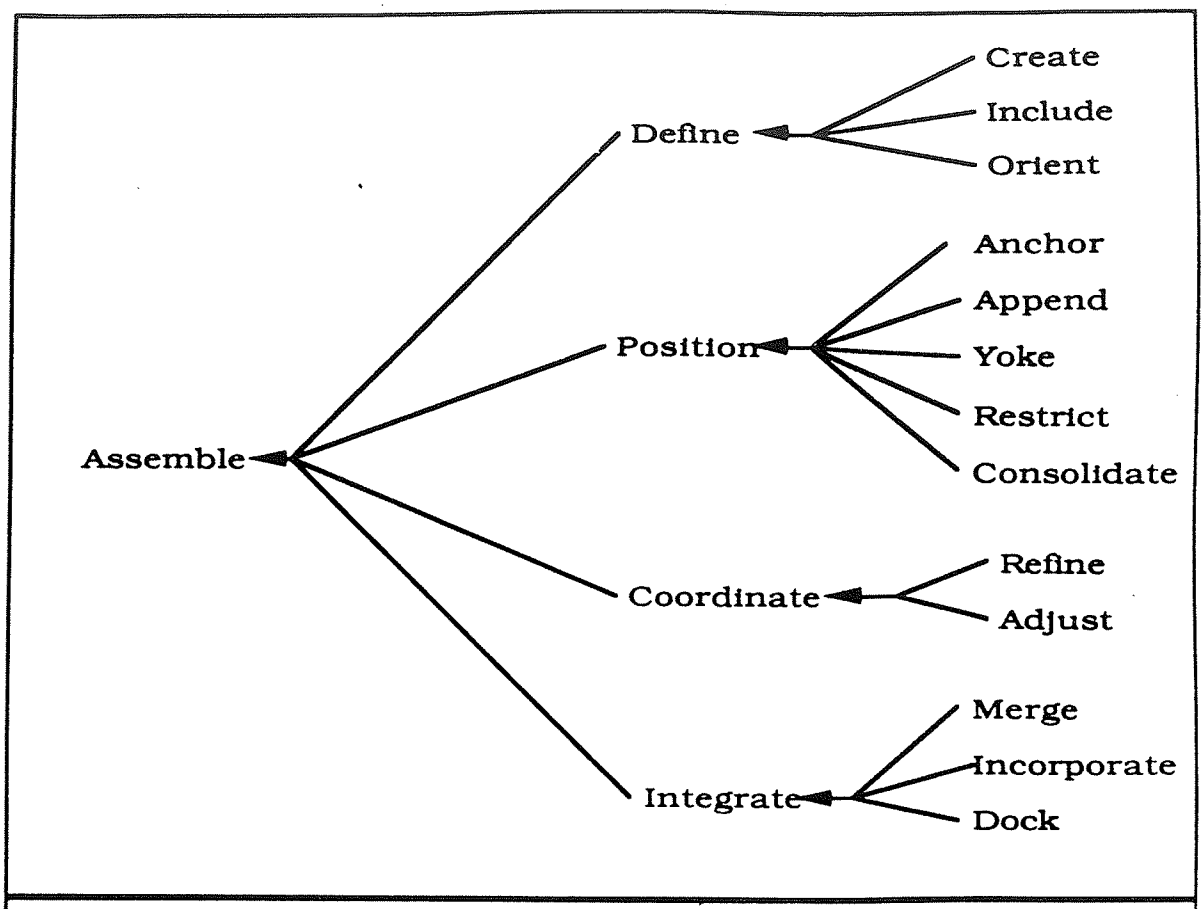
3.3 Applications

At the time of writing there are two significant applications that run within BB1 and make use of the ACCORD language: SightPlan [5] and PROTEAN [6]. WATCH and MetaWATCH have been applied to learn strategies within both these domains as described later.

SightPlan is an application that performs the two-dimensional spatial layout of large civil engineering sites. PROTEAN models the three-dimensional conformations of proteins in accordance with biochemical constraints. In spite of the apparent differences in the domains, both programs operate in a very similar manner. They both position the objects of interest in accordance with various constraints to arrive at a solution in which the locations of all objects have been determined. For clarity only examples from the SightPlan domain will be used to illustrate the various points.

3.4 Strategies within BB1

Strategies within the BB1 framework are implemented using what is called the control plan. The control plan is composed of a set of control KS's which assign ratings to domain KS's and thus control the flow of problem solving actions. The typical form for a strategy to take is that of a skeletalplan. Skeletalplans implement hierarchical strategies. Hierarchical strategies have a tree-like form that unfolds as the strategy is executed (see figure 2). The actions that are actually executed are represented as leaf nodes of the tree. There is a one to one correspondence between the objects in a skeletal-plan and the nodes in a hierarchical strategy.



- **Assemble** <partial-arrangement>
  - **Define** <partial-arrangement>
    - **Create** <partial-arrangement>
    - **Include** <object> *in* <partial-arrangement>
    - **Orient** <partial-arrangement> *about* <object>
  - **Position** <objects> *in* <partial-arrangement> *with* <constraints>
    - **Anchor** <object> *to* <object> *in* <partial-arrangement> *with* <constraints>
    - **Yoke** <object> *and* <object> *in* <partial-arrangement> *with* <constraints>

Figure 1: The ACCORD action hierarchy. These are all the actions defined by the ACCORD framework. Each action has a template defining the objects it operates on. Some sample templates are shown.

Given the dynamic nature of control within BB1 other types of strategy can also be implemented. These include goal-directed reasoning and opportunistic focusing. Goal-directed reasoning entails identifying and performing actions in order to achieve a state in which other, desirable actions, can be performed. These other actions may be desirable in themselves or because of their effect, e.g. in leading to a goal state. Opportunistic focusing allows an application to notice unusual or anomalous conditions and to temporarily focus its

attention on the anomaly to some end, returning later to complete its original goal. All of these reasoning methods can be integrated within the BB1 architecture. [4]

Hierarchical strategies will now be described in more detail since, for reasons described later, WATCH is limited to recognizing only this type of strategy. Throughout the following description, strategies are expressed using the ACCORD language and take examples from the SightPlan domain.

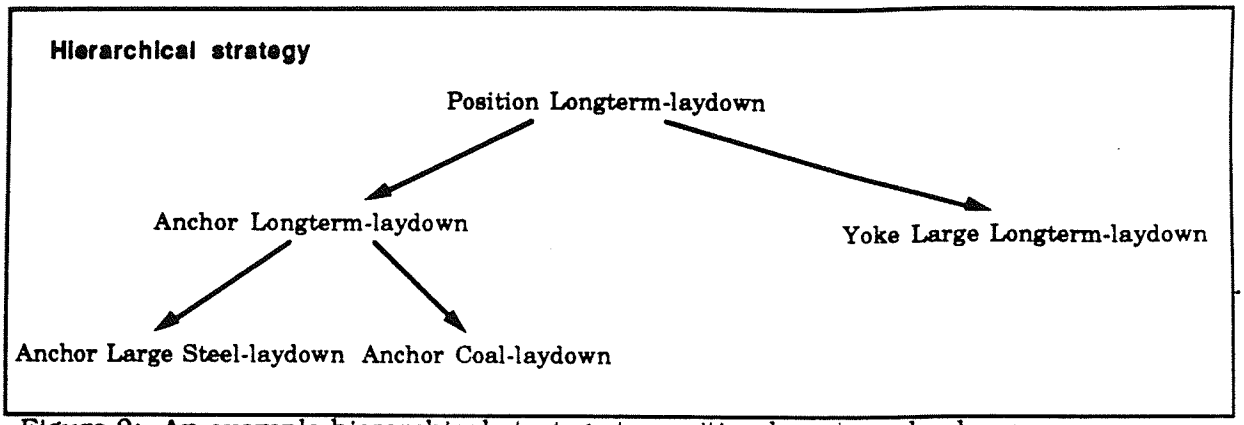


Figure 2: An example hierarchical strategy to position long-term laydown areas

3.5 Hierarchical strategies

As mentioned previously, hierarchical plans have a tree-like structure. Moving down the tree, at each level the problem is decomposed into smaller and smaller subproblems until the leaf nodes are reached. The leaf nodes describe individual actions. The strategy is only as detailed as the actions specified by the leaf nodes. If more than one action matches that prescribed by a leaf node then these actions cannot be distinguished and their execution order is essentially random. In BB1 this means the actions rate equally against the criteria specified by the control plan and will be executed in the order they became executable. If none of the possible actions correspond to a particular leaf node then the node perscribes no actions and is passed over.

One useful feature provided by ACCORD to distinguish between actions prescribed by one node of a strategy is the use of modifiers. A modifier assigns a value to a feature of an object or action. The value can be used to order actions. For example the ACCORD action sentence: "Position LARGE laydown areas" uses the modifier "LARGE" to order the set of possible positioning actions in order of size, the

largest first. Laydowns are areas on a building site used to store materials.

Given this background the hierarchical strategy in figure 2 will be used to provide an illustration. The strategy will be applied to a problem in which the objects to be positioned are: Steel-laydown-1, Steel-laydown-2, Coal-laydown-1.
The following sequence of steps will be prescribed by the strategy to solve the problem:
1) Execution starts at the top node: "Position Long-term Laydown", this is subdivided into Anchoring the laydowns and Yoking the laydowns.
2) Since the Anchor action is first, the laydowns are first anchored. Anchoring is subdivided into anchoring the steel laydown and anchoring the coal laydown.
3) "Anchor Large Steel-laydown" is a leaf node (or "focus") so any possible actions matching its description will be executed. If in the example Steel-laydown-1 is the larger, it will be anchored first. Then Steel-laydown-2 will be anchored. If there are no more possible actions that match this focus the next one is executed.
4) The Coal-laydown will be anchored in accordance with the action of the next focus.
5) Finally the action "Yoke the long-term" laydowns is reached. Note that this again is a focus so its actions will be performed. All possible Yoke laydowns actions will be executed, in order of the size of the object they yoke.

4 WATCH

This section describes the WATCH application, how it operates and what its limitations and biases are. It also gives an example of inductive generalization with the ACCORD framework.

4.1 Overview of WATCH

WATCH is a BB1 application that learns strategic knowledge for use within other BB1 applications. When using WATCH, the BB1 application under consideration is run with all the domain knowledge available but with no control knowledge i.e. no control KS's. A domain expert provides the application control knowledge by choosing which of the domain KS's should be executed on each BB1 cycle. By

doing this the expert is choosing the order in which actions should be performed to solve the problem. WATCH performs inductive generalization on the resulting action sequence to infer the strategy used.

To perform inductive generalization WATCH has available to it the ACCORD language hierarchy, the domain knowledge and some problem specific knowledge. The domain knowledge includes such information as the objects under consideration in this domain, their attributes and the relationships between them. These will take the form of object hierarchies on a BB1 blackboard. The problem specific knowledge includes the actual set of objects within the problem being solved, the values of their attributes and the constraints they have with respect to each other.

4.2 Assumptions

WATCH makes a number of assumptions about the problem solving strategy being used. These assumptions are:

1) The expert is consistent in using a single strategy to solve the problem. This strategy may not be well formed in the experts mind, but if he/she switches between two or more strategies WATCH will become confused and make no sense of the control structure.

2) The strategy being used is hierarchical. As described later in this section, WATCH acts by inductively generalizing the action sequence up the ACCORD hierarchy (This hierarichy is illustrated in appendix 1). This generalization method will only describe strategies that are hierarchical in nature. See [7] for a technique that allows WATCH to recognize opportunistic interruptions in a hierarchical strategy.

3) The experts strategy can be implemented using the actions provided by the ACCORD framework. The basic assumption about the use of languages in BB1 is that the application will be able to solve problems using the actions provided it by the language, this assumption must hold for the strategies sought by WATCH.

4.3 How WATCH works

WATCH sequentially examines the sequence of problem solving actions and builds a tree of possible generalizations of various subsequences. A set of heuristics select the subsequences of actions to be generalized. When all actions have been examined the possible hierarchical strategies are identified and ranked according to a

second set of heuristics. The best set are chosen and the modifiers that explain the ordering of actions in each sub-sequence are inserted. The strategies are finally written out as a set of BB1 skeletalplans.

Each of these operations will now be examined in greater detail.

4.3.1 Inductive generalization

As mentioned previously ACCORD serves as a generalization language. WATCH steps through the action sequence trying to find similarities between sets of adjacent actions. Some types of similarities looked for are: sets of actions that all involve a particular object, or sets of actions that all apply the same constraints, or all use the same ACCORD verb. Figure 3 shows an example of inductive generalization up the ACCORD hierarchy.

Anchor fabrication-yard-1 *to* power-plant-1 *in* partial-arrangement-1 *with* constraint-set-fabyard-power-plant.

Yoke fabrication-yard-1 *and* construction-offices-1 *in* partial-arrangement-1 *with* constraint-set-fabyard-offices.

Generalizing up the ACCORD class hierarchy could give:

Securely **Position** fabrication-yard-1 *in* partial-arrangement-1 *with* strong constraint-set

Figure 3: Generalizing two ACCORD sentences up the ACCORD hierarchy. Bold words are actions, Underlined are modifiers, italics are ACCORD template words and normal font are domain objects.

Every ACCORD object or action can be generalized up one or more levels of the ACCORD hierarchy and can have one or more modifiers applied to it. Since every sentence in ACCORD has an action and many objects there are many possible generalizations for any two sentences. The fact that WATCH also has to choose which of the possible subsequences in the action sequence should be generalized together means that WATCH faces a combinatorial problem of extreme proportions. Figure 4 gives an example of two possible generalizations of three sequential actions.

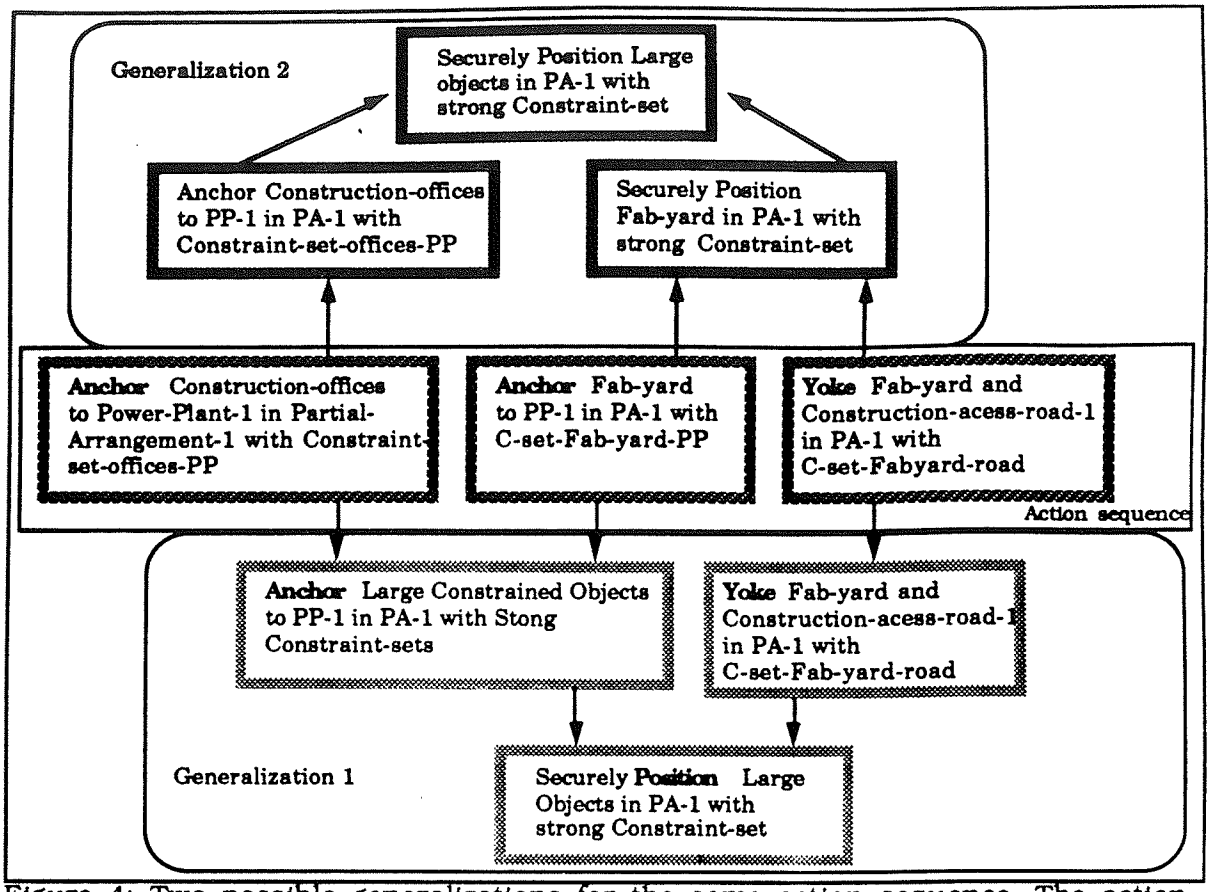


Figure 4: Two possible generalizations for the same action sequence. The action sequence is the center set of actions from left to right, upward arrows lead to one generalization tree, downward arrows lead to another.

4.3.2 Generalization heuristics

In an attempt to cope with this combinatorial explosion of alternative generalizations WATCH uses a number of heuristics to prune the generalization tree. Most of these heuristics relate to the general goal of creating a well formed hierarchical strategy. Some examples of this kind of heuristic are "Prefer longer sequences of statements" (because the fewer generalizations you have to make the better) and "Prefer generalizing over fewer variables" (because if everything is generalized together detail is lost).

Some of the heuristics are specific to the ACCORD language such as: "Avoid generalizing actions". This is true for ACCORD because the actions do the work and so should be kept as close to the original sequence as possible. This may not be true of other languages.

3.3.3 Ranking strategies

When all the actions have been considered, the generalization trees that account for all the actions performed are identified and ranked according to a second set of heuristics. These heuristics, similar to the first set, are a measure of the "well-formedness" of the hierarchical plan specified by the generalization tree. Two examples are: "Avoid poorly grouped strategies" and "Prefer less generalization". The best two or three strategies are chosen for output.

4.3.4 Identifying the appropriate modifiers

For each of the strategies selected, WATCH inserts any modifiers that explain the ordering of generalized objects or actions. These modifiers model the heuristics the expert uses to order a set of similar actions.

4.4 Biases and Limitations of WATCH

The major limitation imposed on WATCH is its lack of knowledge about BB1 state information. BB1 applications are normally very state dependant - the KS to be fired is chosen on each BB1 cycle according to the state of the system at that time. The only state information available to WATCH is the actual action sequence performed and information about alternative actions on each cycle. The lack of state information makes it impossible for WATCH to find goal-directed plans.

WATCH is also biased by its use of a language, in this case ACCORD. It cannot find strategies that cannot be expressed in ACCORD. Without the language framework, however, WATCH would have no basis for generalizing action sequences.

WATCH has no knowledge of the use of a language. For example WATCH does not know the general sequence of problem solving actions in ACCORD i.e. define partial-arrangements each with an anchor etc. Such knowledge could be used within its generalization phase to help choose the subsequences of actions to be generalized.

Finally WATCH is very much biased by the problem solving session on which it is run. This is because the same strategy can result in very different action-sequences when applied to different problems. Thus the strategy WATCH produces is tailored to solving one particular problem and may not work with problems containing

different objects or constraints. This arises from the fact that WATCH can only observe the experts strategy in action on one problem and if the problem does not fully exercise all branches of the strategy it is impossible for WATCH to correctly deduce those branches. This severely limits WATCH's utility as a knowledge acquisition tool. It is toward this limitation that the MetaWATCH system is addressed.

5 MetaWATCH

This section describes MetaWATCH. The motivation behind the system is given and its operation is detailed.

5.1 Introduction

MetaWATCH is a BB1 application that examines multiple sets of WATCH output in the context of the problems they solve to incrementally converge on a common "super-strategy". This super-strategy is closer to the experts intended strategy and can be used to solve a wider range of problems. It is output as a BB1 skeletalplan.

5.2 Motivation

As described previously the strategies produced by WATCH are very problem specific. The basic premise of MetaWATCH is that given multiple problem solving sessions we are more likely to see all branches of the experts strategy fully exercised and that this can make the learning process more accurate.

5.3 Operation

5.3.1 Input

The input to MetaWATCH is the ACCORD language blackboards and all the domain knowledge used by WATCH. Also for each problem solving session examined by WATCH, MetaWATCH is given:

- The problem specific knowledge used by WATCH.
- The state information used by WATCH. This consists of some BB1 agenda information recorded during this session detailing the action sequence and information about alternative actions at each cycle.
- The best two or three strategies found by WATCH during this session.

5.3.2 Incremental Convergence on a Super-strategy

MetaWATCH operates as follows:

- All strategies found by WATCH from the first problem solving session are compared to all strategies found by WATCH from the second session.
- For each pair that have a common super-strategy that super-strategy is generated. Any such super-strategy will solve a set of problems of which the two test cases are elements.
- Correspondences are then sought between all such super-strategies and the WATCH output strategies for the next problem.
- Again all common super-strategies are found. These super-strategies can be used to solve a set of problems of which the three problems examined thus far are elements.

The above sequence iterates over all the sets of WATCH input, of which there are an arbitrary number. In each case the super-strategy generated will be the most common generalization of its component strategies. In the worst case MetaWATCH incrementally converges on a strategy that only solves the problems seen so far in the same manner as the expert did. In the best case we find the actual strategy the expert has been using which can then be used to solve any problems to which he would apply the same problem solving strategy.

There are a number of points to be made about this procedure. First, all WATCH strategies explain the action sequence observed for the given problem. If a strategy cannot be generalized to a more general strategy that will explain the other problems examined then it will be termed an incorrect strategy for this set of problems. This means it cannot be an instantiation of the more general strategy the expert is using to solve all the problems. Figure 5 gives examples of correct and incorrect strategies. An important point to be mentioned here is that MetaWATCH can only work with the strategies found by WATCH. If WATCH does not list an instatiation of the correct one among its output it is impossible for MetaWATCH to find it. It is true that WATCH can find all possible strategies explaining the action-sequence however given the huge number of such strategies it is possible that the correct one will have been rejected during pruning.

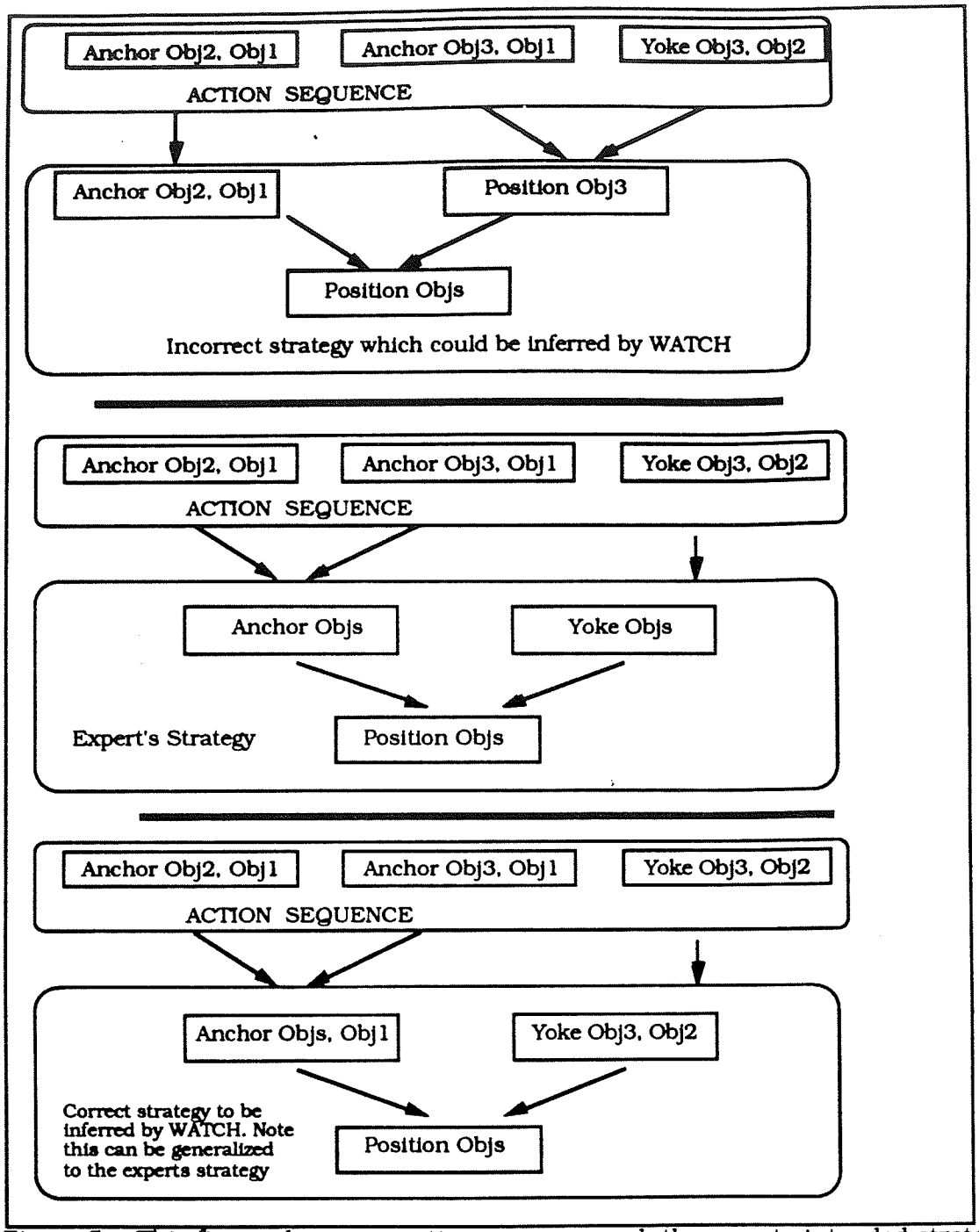


Figure 5: This figure shows an action sequence and the experts intended strategy. It can be seen that WATCH can come up with numerous strategies that explain the action sequence, however of the 2 shown only the lower one is an instantiation of the experts strategy. The other one is termed incorrect.

It may seem that this approach will lead to a combinatorial explosion of the number of strategy comparisons required. For example given 4 problem solving sessions for which WATCH finds 4 possible

strategies each, the worst case number of strategy comparisons is 4^4. However we are assuming the expert is using the same basic strategy to solve all the problems, there is only one instantiation of this strategy when applied to each problem, therefore only one of the WATCH outputs can be correct. Experience has shown that WATCH comes up with many spurious strategies very specific to the problem solved. These strategies do not match with others and so early on in the comparison process the true strategy is found. Subsequent comparisons then explore this strategies branches in greater detail.

The convergence described is expected to occur after only a few iterations thus the number of problem solving sessions used will be low. So while in the worst case the problem is combinatoric, in practice it is manageable.

Another approach to comparing strategies is a version space approach, in which the most general and most specific versions of the strategy are stored [8]. Correct strategies are merged with the most specific version, to generalize it. Incorrect strategies are merged with the most general version, to specialize it. As more strategies are examined the true strategy is converged upon. There is one difficulty with this approach: Our version space has many dimensions, the strategy trees can have any branching factor and each node of the tree is composed of a sentence whose components can all be generalized up one of more levels of a generalization hierarchy. A version space approach could work if the task was to narrow in on the specific action for an individual node. However it is difficult to see how two instances could be said to bound the space of possible structures for a strategy tree.

5.3.3 Comparing Two Strategies

This section describes in detail the process of comparing two strategies to determine whether or not they have a common generalization. The technique is essentially a walk through the strategy trees in conjunction with a stepping through the corresponding actions in each action sequence. See figure 10 for an algorithm that describes the matching process.

The question to be asked when comparing two strategies is whether or not they could both be instantiations of the same super-strategy as applied to different problems. The test of this is to determine whether the same set of actions are generated when the first strategy is applied to the second problem. Similarly for the second

strategy applied to the first problem. The difficulty with this approach is that the two strategies have been generated from different WATCH runs so they will not necessarily have the same branches in common. This is because, as mentioned previously, different problems exercise different branches of the experts strategy. The best that can be done is to compare across the strategy branches which they have in common or in which one strategy is more general than the other.

This is best illustrated with examples. The strategies shown in figure 6 were generated by WATCH from the action sequences listed, they give a positive comparison. Figure 8 shows a third strategy and action sequence which will be compared to strategy 2 to give an example of a negative comparison.

First the comparison of the strategies in figure 6. Starting at the top node, MetaWATCH steps through the two trees comparing nodes and noting the actions indicated by the leaf nodes, building up the corresponding super-strategy as it goes. In this comparison the top two nodes are the same, so that node is copied straight across to the new strategy. The next nodes compared are "Anchor Steel-laydown" verses "Anchor Constrained Lt-laydown". From the ACCORD hierarchy it is known that Steel-laydown is a type of Lt (Long-term) laydown, so the second node is more general than the first. This means they may match, one being a generalization of the other.

MetaWATCH has to determine if the two subtrees rooted by these nodes could be derived from one, more general, subtree. To do this all the actions from the first action sequence specified by the more specific node - " Anchor Steel-laydown"- are stepped through. Using the BB1 saved state information MetaWATCH then checks to see whether any actions specified by the more general subtree were possible at this point in this problem i.e. were there any actions of the form " Anchor Lt-laydown" possible? This state information is not shown in the figure but for this example it is assumed that no such anchor actions were possible. This means that if at this point the expert strategy was to "Anchor Lt-laydown", in the context of the first problem it would appear as just an "Anchor Steel-laydown" action, so the subtrees match. The more general subtree is then taken from the second strategy and copied onto the new super-strategy. It is assumed that the children nodes in the more general subtree contain necessary detail. All the "Anchor Lt-laydown" actions in the

second action sequence are then stepped through and the comparison proceeds to the next node.

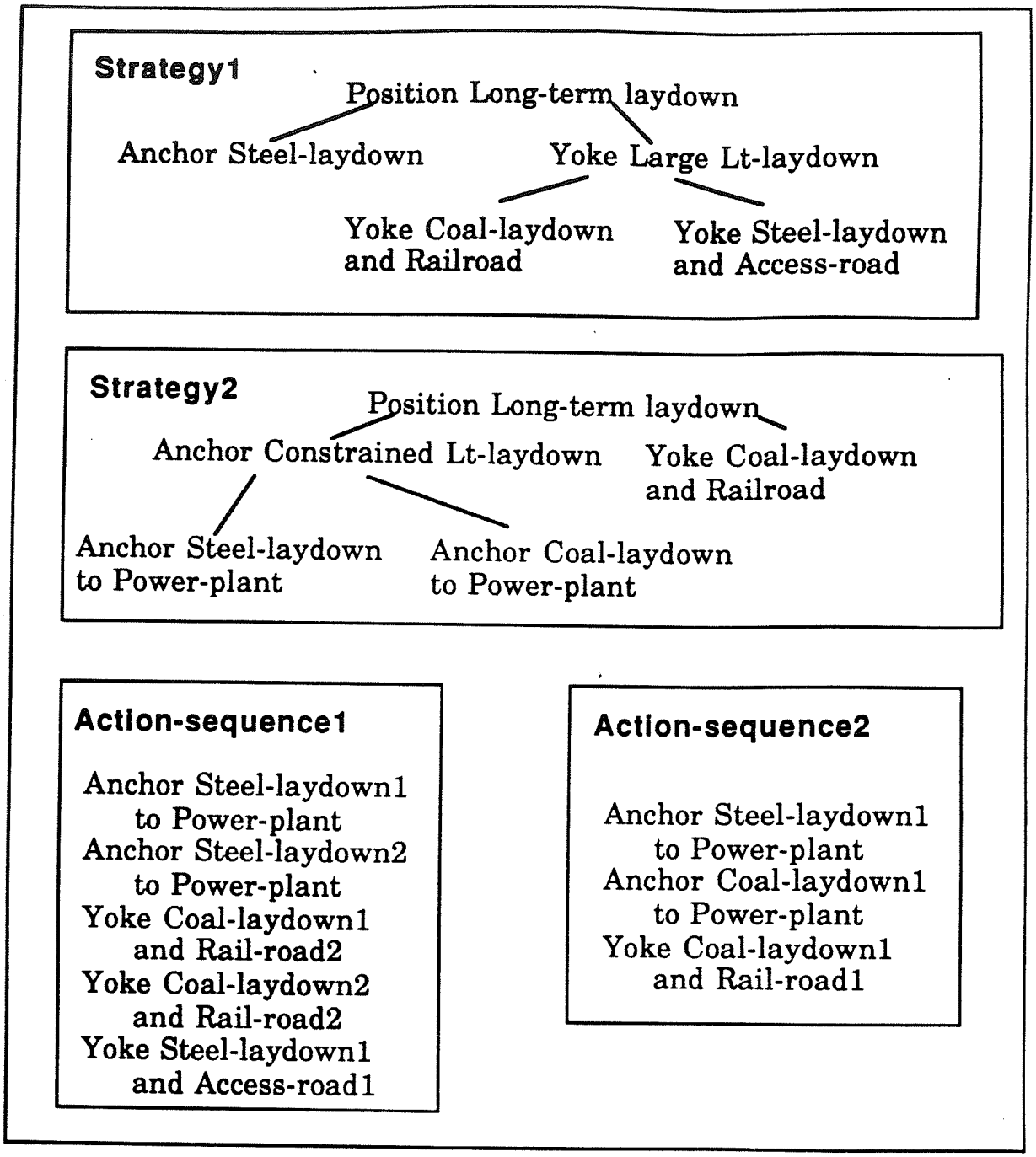


Figure 6: Two strategy trees and the action sequences from which WATCH derived them. See text for an explanation.

The next nodes to be compared are the "Yoke Large Lt-laydown" and "Yoke Coal-laydown and Railroad" nodes. Again these may match but one - the "Yoke Large Lt-laydown" - is more general. In a manner similar to the previous check the actions proscribed by the more specific node in its action sequence are stepped through. Whether or not any actions prescribed by the more general node were possible at this point is then checked, if not the nodes match. Again for the purposes of this example it is assumed that they match. The more general subtree is then copied across, completing the comparison. The resulting super-strategy is shown in figure 7. Note that while neither of the WATCH strategies could explain both the action sequences, this new super-strategy can.

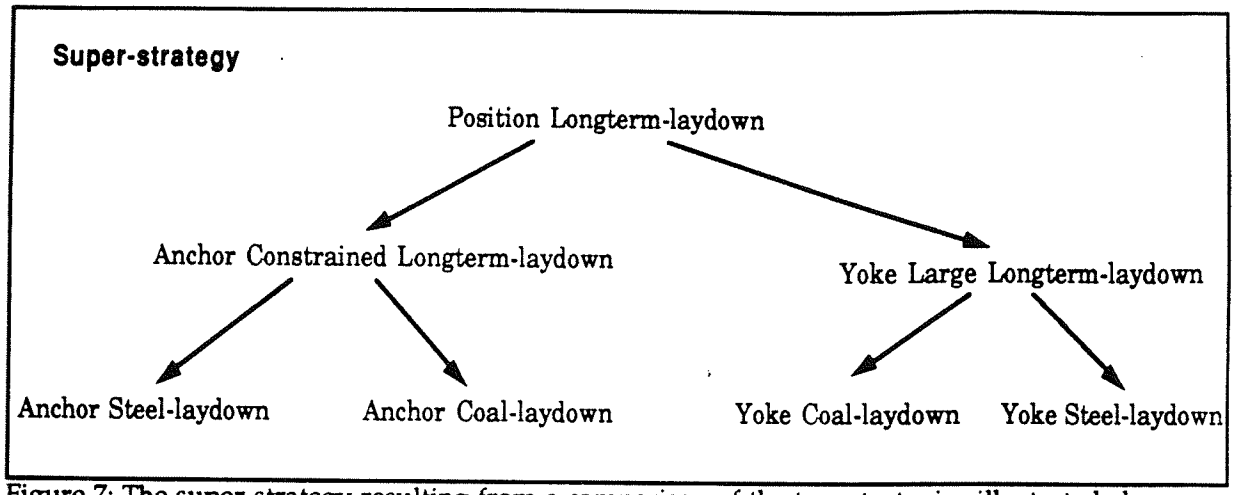


Figure 7: The super-strategy resulting from a comparison of the two strategies illustrated above.

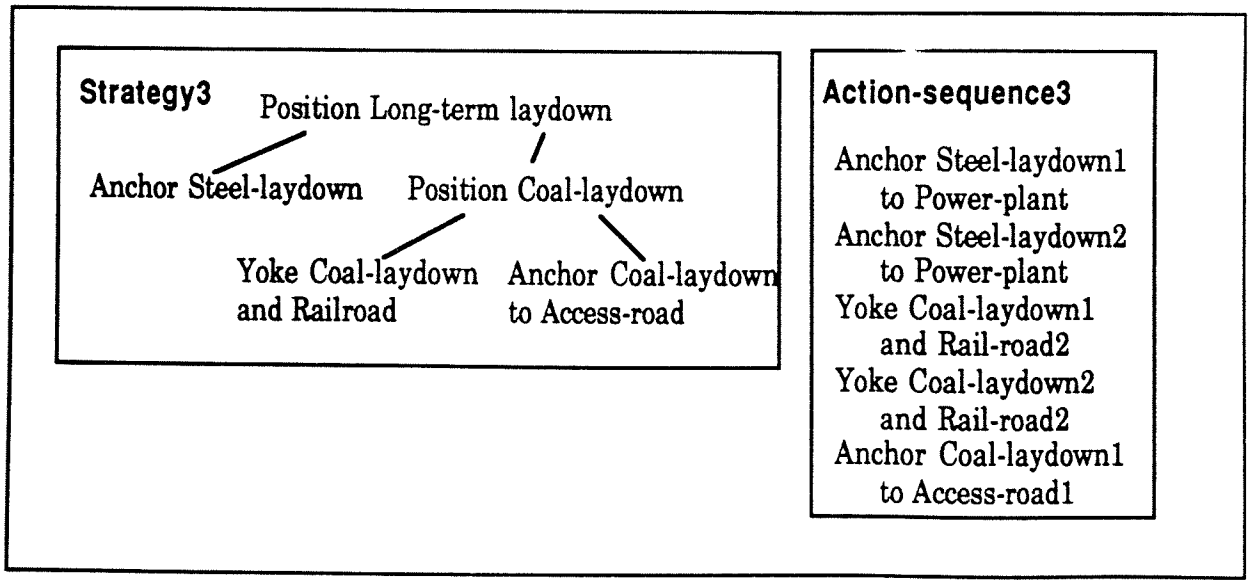


Figure 8 : The strategy 3 has been derived by WATCH from action-sequence 3.

The comparison of Strategy 3 in figure 8 and Strategy 2 in figure 6 will now be performed. Firstly, as before, the top nodes are matched. The next nodes to be matched are the "Anchor Steel-laydown" node of strategy 3 and the "Anchor Constrained Lt-laydown" node of strategy 2. As before the former is a more specific version of the latter, so we cannot rule out the possibility of the nodes matching. Since "Anchor Steel-laydown" is the more specific node the actions it dictates in Action-sequence3 are stepped through. We then check to see if there are any actions that match the more general node that could have been executed in problem 3 at this time. In this example the action "Anchor Coal-laydown1 to Access-road1" matches the more general node. This means that strategy 2 would perscribe a different ordering on the problem solving actions than strategy 3 does. Thus the strategies don't match, i.e. could not both be instantiations of a more general strategy.

This method while resulting in a more widely applicable super-strategy does not loose any of the detail of either component strategy. Thus while this paper mentions *converging* on the required strategy and *generalizing* a pair of strategies, the actual term used should be a combination of the two. The strategies become more widely applicable while at the same time becoming more detailed. The only time MetaWATCH loses detail from the originals is when two matching strategies have the order of some subtrees swapped, but either order is possible. MetaWATCH would decide the expert did these actions in an arbitrary order and would generalize up the language hierarchy to their common ancestor. For example if the strategy of figure 7 above was compared with one the same in every way except that the "Anchor steel-laydown" and "Anchor coal-laydown" foci were swapped, the new super-strategy generated would use "Anchor Constrained Longterm-laydown" as the focus. If the order of foci is superfluous to insist upon any one ordering would be wrong.

There are many other circumstances that can arise in comparing two strategies, two such circumstances are outlined below and the methods MetaWATCH uses to deal with them discussed.

One problem arises when two nodes being compared are different, yet neither of the nodes can be said to be more general. This arises when one node has a more general action and the other deals with more general objects, or both deal with one type of object more

generally. In this case MetaWATCH generates an action sentence that contains the common superset of the two and compares both against this supersentence in the manner detailed previously (checking for possibly executable actions etc). If both match, the supersentence is used in the corresponding strategy node. See figure 8 for an example.

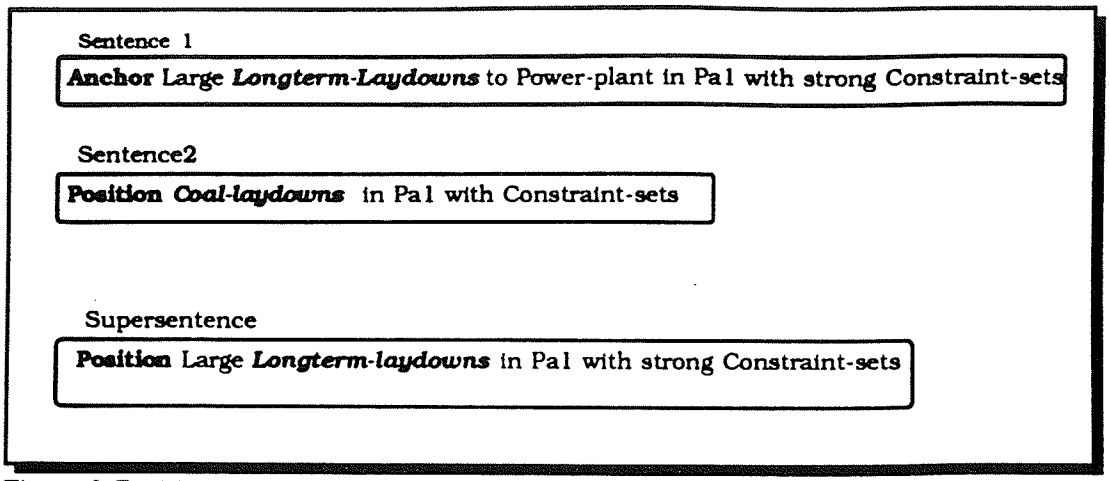


Figure 8: Position, in sentence 1, is the more general action but Longterm-laydown, in sentence 2, is the more general object, thus neither sentence can be said to be more general. For comparison purposes MetaWATCH generates the common supersentence.

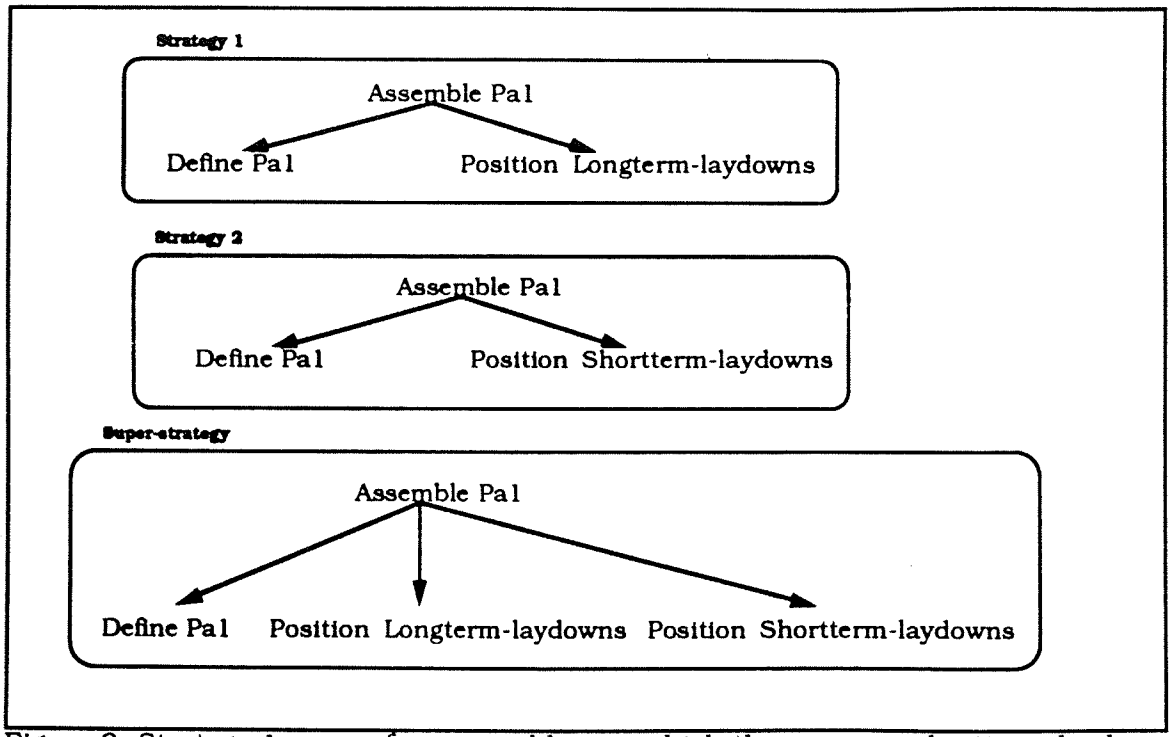


Figure 9: Strategy 1 comes from a problem in which there are no shortterm laydowns, strategy 2 from a problem with no longterm laydowns. The super-strategy will deal with both laydown types, but has no information about which the expert would do first so the order is arbitrary.

Another such problem occurs when the two strategies being compared have distinct subtrees neither of which would make sense when applied to the other problem. This arises when both strategies explore totally different branches of the experts strategy. See for example figure 9. In this case MetaWATCH inserts both subtrees into the super-strategy being generated. There is, however, the problem of ordering the subtrees. MetaWATCH looks for other information to determine the ordering. In the absence of any other information an arbitrary order is assumed in lieu of more information from subsequent problems.

5.3.4 Dealing with modifiers

As mentioned previously modifiers model the heuristics used by the expert to order a set of similar actions. In a skeletalplan type strategy this means the heuristics used to order the set of actions prescribed by the action sentence of a strategy node. So if, in BB1, an action sentence prescribes a set of anchoring actions and the expert likes to anchor large things first, the modifier- "LARGE" - will be applied to the anchored objects.

WATCH, as one of the final steps in its operation, inserts modifiers into the action sentences of the nodes in the strategies it has developed. WATCH seeks modifiers that explain the observed ordering of any objects or actions that have been generalized. All such modifiers are inserted into the sentence.

The problem with this approach is that often the generalized elements of the action sentence have been derived from only two or three lower level (on the language hierarchy) elements. Thus many of the modifiers WATCH finds to order the set of actions are spurious and do not accurately model the experts heuristics. For example if the expert prefers "operating on large objects first" it may just happen that the ordering of the objects by size corresponds with the ordering of the objects by most constrained and by most time-costly. WATCH will insert all three modifiers into the action sentence. MetaWATCH can identify some of these spurious modifiers by comparing the sets of modifiers across multiple problems. Thus it can more accurately model the experts heuristics.

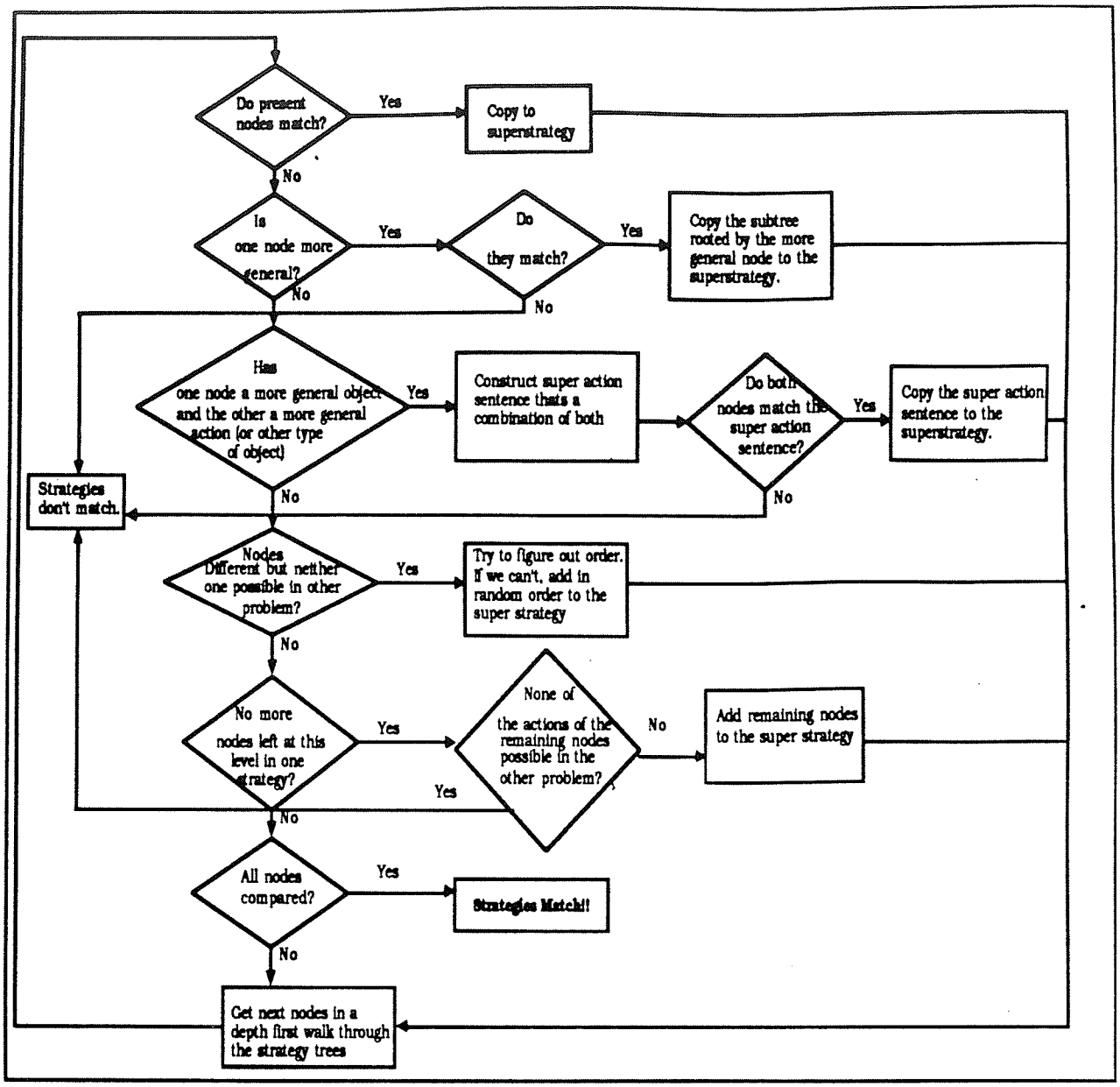


Fig 10: The algorithm used to compare two strategies. See text for an explanation of how some of the steps are implemented.

The modifiers are one of the last things dealt with by MetaWATCH. When a common super-strategy has been found for two strategies each of the matching nodes of these strategies are examined and the common modifiers extracted. Since WATCH includes all modifiers that correctly order the actions specified by the nodes, the correct modifiers - if any - must be included in the sets listed by the nodes on both strategies. So any modifiers that are not common to both nodes must be spurious and are removed by MetaWATCH. Figure 11

shows an example of MetaWATCH converging on the correct modifier by comparing successive sets of modifiers inserted by WATCH in different problem solving sessions.

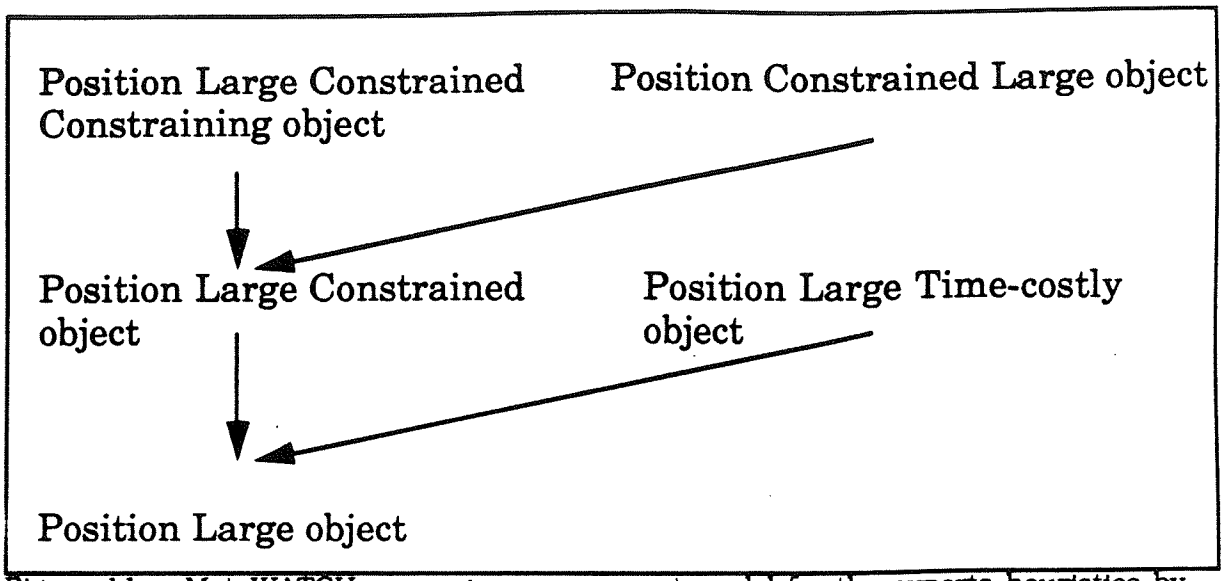


Figure 11: MetaWATCH converges on a correct model for the experts heuristics by comparing the modifiers proposed by WATCH for the action sentence of one node across three problems.

5.3.5 Choosing a strategy

Given that MetaWATCH has compared a set of WATCH outputs taken from a number of problem solving sessions and added the appropriate modifiers to each action sentence, the one problem that can remain is that there may be more than one possible super-strategy that will explain the observed actions in all the training sessions.

There are a number of approaches to be taken here. Adding more training instances may discriminate between the remaining alternatives. Since MetaWATCH is a knowledge acquisition tool it may suffice to simply ask the expert to indicate which he/she considers closer to the correct strategy. It will certainly be a lot easier to pick one refined strategy from a number of alternatives than describe the strategy from scratch. Finally a set of heuristics, such as those used by WATCH, could be used to choose the best among the remaining possibilities. These heuristics should choose a compromise between an overly general and an overly specific strategy. See figure 12.

The present MetaWATCH assumes the appropriate number of training instances are available to converge on one strategy and if this is not true it will output all alternative super-strategies found.

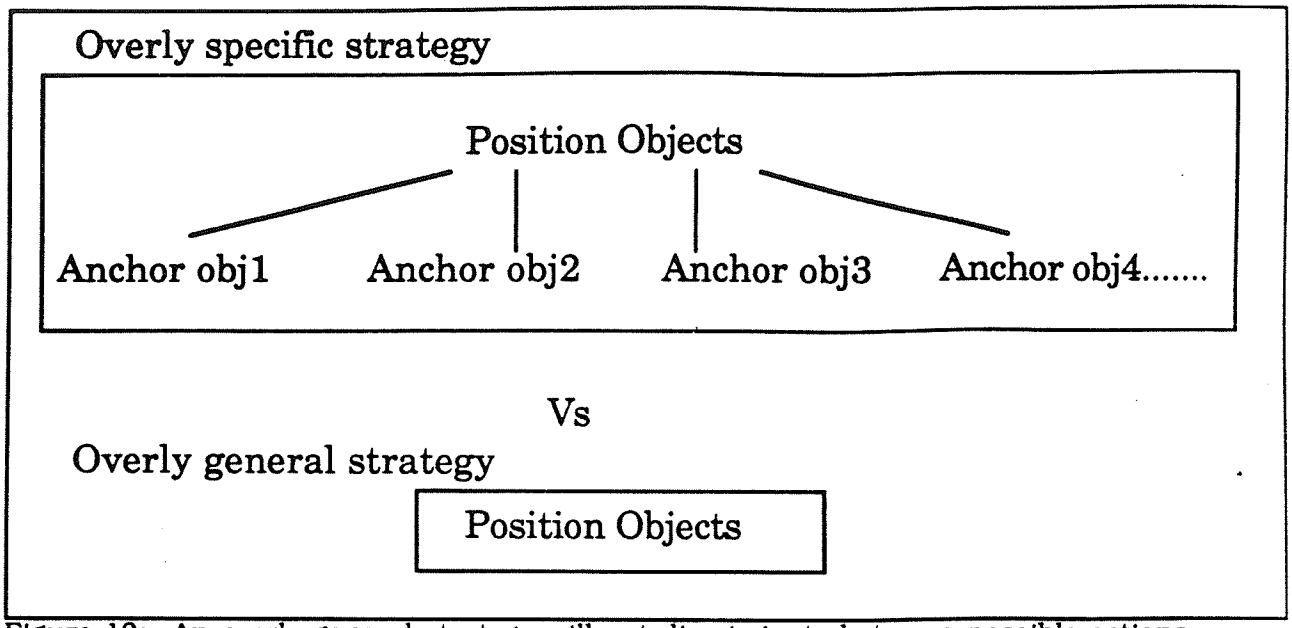


Figure 12: An overly general strategy will not discriminate between possible actions and is little better than no strategy at all. An overly specific strategy cannot handle problems which contain objects not previously seen.

6 Testing and validation

This section describes a set of test cases which have been used to validate MetaWATCH, the results of that validation and some observations.

6.1 Introduction

This paper presents a lot of ideas that will work on the idealized examples described. However for MetaWATCH to be of any use as a Knowledge Acquisition tool it has to learn strategies under more realistic conditions.

Throughout the design and implementation of MetaWATCH it was tested and refined using a set of WATCH outputs from examples in the Protean domain. The examples used for validation were taken from the SightPlan domain. This tested the assumption that MetaWATCH can learn strategies from multiple domains.

To obtain the test cases WATCH was run on a set of SightPlan outputs for which the control strategy was known. In this way the assumption that a single consistent strategy was being used across problems was known to be true. The known control strategy provides a gold standard against which the effectiveness of MetaWATCH's learning can be measured.

To obtain the test cases a single realistic strategy from SightPlan was used to solve three distinct problems. The appropriate action sequences and domain knowledge were then passed to WATCH which learned a set of possible strategies for each problem. The three sets of WATCH output were then input to MetaWATCH with the goal of finding the correct strategy.

6.2 The Test Cases

This section describes in detail the strategies used to test MetaWATCH. The strategies are listed in appendix 2. WATCH output a set of possible strategies for each SP problem solving session run through it. For the sake of brevity only the strategies used to generate the final superstrategy are shown. The failed comparisons gave rise to some of the observations noted in a subsequent section and are described where appropriate.

The strategy trees shown in the appendix are taken from screen dumps of the BB1 display. For each strategy a listing of the strategy nodes along with the action they perform is included. The nodes and links in each display are listed by BB1 in the order created and may not reflect the actual ordering or execution. In cases where this may cause confusion, the correct order of execution will be outlined in the description of each strategy - see next subsection. It is intended that the reader refer to the appendix while reading the following descriptions.

6.2.1 The SightPlan Strategy

This is a strategy that has actually been implemented and used within the SightPlan application. It will be termed the AM1 strategy since it has been used to layout the American1 power plant. It has two main parts: the AE-Layout and the CM-Layout.

The AE-Layout is the Architect-Engineer Layout. This section of the strategy defines the partial-arrangement (pa) and positions all the physical site objects within this pa. Note the order of execution of the Define actions is to Create the pa, do all the Includes and then Orient the pa. The Position branch needs some more explanation. The first action is to Yoke all the physical site objects to subareas. The next two foci: POSITION-CONSTR-FACILITIES and UPDATE-OCC-SPACE are done in parallel i.e. actions from both may be interspersed.

The CM-Layout is the Construction-Manager Layout. This section of the strategy includes laydown areas in the pa and positions them. In a similar manner to the AE-Layout the laydowns are first Yoked with subareas and then Positioned in parallel with Updating occupied space. The UPDATE-OCC-SPACE foci is used by both parts of the strategy.

6.2.2 WATCH Skeletalplan1

The first problem solved by SightPlan using the AM1 strategy did not have any laydown areas to position, thus the second part of the SP strategy, the CM-layout, was not exercised. The output from WATCH therefore does not include this part of the strategy at all.

Observations on this strategy:

- WATCH found the wrong overall structure for the strategy but the one found is functionally equivalent to the one sought. The YOKE-FAC&AREA actions were grouped with the DEFINE-AE-PA instead of with POSITION-CONSTR-FACILITIES.
- The DEFINE-AE-PA subtree was found correctly albeit without the correct amount of detail within the Include actions. (Node 81006 in the strategy)
- The YOKE-FAC&AREA was noted but the problem didn't exercise the action fully and so a more specific version of the action was found. (Node 81007 in the strategy)
- WATCH didn't at all notice the interleaved UPDATE-OCC-SPACE but found the POSITION-CONSTR-FACILITIES NODE. (Node 81003 in the strategy). Lots of spurious detail was included to try and order the actions within this subtree.

6.2.3 WATCH Skeletalplan 2

The second problem solved by SP using the AM1 strategy was much smaller than the first and consequently has less spurious detail. It also did not position any laydown areas and so ignored the CM-layout.

Observations on this strategy:

- This strategy contains the same structural mistake as the previous one.
- It has the same DEFINE-AE-PA branch. (Node 48256)
- This problem exercised the YOKE-FAC&AREA focus differently than the previous problem. As a result in this strategy the node describes an action whose first object is more specific but whose second object is more general.(Node 48255)
- The POSITION-CONSTR-FACILITIES node found is more specific than that found in the previous problem. (Node 482510)

6.2.4 The first MetaWATCH superstrategy

The strategy found by MetaWATCH by comparing the two WATCH strategies described comes closer than either of them to the original AM1 strategy. However since both of the strategies compared have the same structural mistake the MetaWATCH strategy has it also.

Some observations:

- MetaWATCH knows nothing of the CM-LAYOUT so it is not mentioned in the strategy.
- The MetaWATCH strategy, by combining the two previous strategies gets the YOKE-FAC&AREA node completely correct. (Node 10)
- One of the POSITION-CONSTR-FACILITIES nodes compared is more general than the other so the subtree below it is used in the MetaWATCH strategy. However all the spurious detail is copied along with it. (Node 13).

6.2.5 WATCH Skeletalplan3

The problem this strategy is derived from did not contain any construction facilities and so doesn't exercise the POSITION-CONSTR-FACILITIES node. It does however contain laydown areas and so exercises the CM-LAYOUT part of the AM1 strategy.

Observations:

- In the same way as for all previous strategies the structure of the strategy tree for the first part of the strategy is incorrect.
- In this strategy WATCH misses out completely on the POSITION-AE-OBJECTS subtree.
- The CM-LAYOUT subtree is found with reasonable precision. Again the parallel foci are ignored and some spurious detail is inserted.

6.2.6 The MetaWATCH output strategy

Since the third WATCH output was the final one, MetaWATCH outputs the result of its comparison of this strategy and the previously described intermediate MetaWATCH superstrategy.

Observations:

- Since the first subtree was correct previously, apart from the problem with structure, it is the same in this strategy.
- In constructing this output MetaWATCH tries to compare the POSITION-AE-OBJECTS subtree of its previous strategy with the CM-LAYOUT subtree of the new strategy. These are found to be incompatible with each other. However, as described previously - in section 3.4.3.3, it is found that both subtrees should be in the final strategy. Without any extra information MetaWATCH has no idea in which order the subtrees should be inserted and puts them in in the order compared. This by chance is the correct ordering.
- All the spurious foci from the first strategy are included in this output strategy because we have found nothing to contradict them. Recall that MetaWATCH never loses detail from a strategy unless it has explicit knowledge that the ordering of actions was arbitrary.

6.3 Observations

The major assumption made in developing the MetaWATCH system was that an instantiation of the correct strategy will be contained within each set of WATCH output. It was found that this was not the case for all the examples tested.

The WATCH strategies tend to be grouped around a common theme with only minor variations. The reason for this is that WATCH makes early assumptions about the groupings of subsequences of the action sequence. If these assumptions are wrong all the output strategies will be wrong.

Even given the correct initial assumptions about subsequences of actions the output strategies invariably have associated spurious detail. This can be seen in all three of the WATCH strategies detailed.

It was also noted that there were some errors in the way MetaWATCH operates

- The assumption that when comparing two subtrees the more general one should be used was found to be false. Often it is found that the less general subtree has detail that should also be taken into consideration. For example in comparing nodes 81003 and 482510 from skeletalplans 1&2 the latter is the less general node but contains foci that have detail not covered by the former.
- MetaWATCH only ever compares pairs of subtrees, there was a need in some of the strategies tested, but not considered here, to compare one subtree in one strategy against two or more in another strategy.

It can be seen however from the test cases reviewed that MetaWATCH works well within its assumptions and can produce strategies that closely model the one used to solve a set of problems.

7 Conclusions

Strategies can be learned accurately by using multiple problem solving sessions to more fully exercise all branches of the strategy tree. MetaWATCH takes a first step toward the goal of performing knowledge-based strategy comparison. It works well if its assumptions about the initially inferred strategies, i.e. the WATCH output strategies, hold.

However it was seen that in many cases these assumptions - specifically that WATCH always outputs an instantiation of the correct strategy - don't hold.

At present MetaWATCH is extremely sensitive to any small differences between strategies. For example if one of the spurious foci in Skeletalplan1 had conflicted with the details of one of the other skeletalplans they would not have matched. WATCH cannot be expected to discover the correct instantiation of the experts strategy with complete accuracy. To be practical MetaWATCH needs to use a more heuristic strategy comparison.

8 Extensions to the work on MetaWATCH

The type of heuristic strategy comparison referred to in the previous section could be achieved using a set of heuristics in conjunction with MetaWATCH. The present MetaWATCH system

could be used to match and discover the inconsistences in a pair of strategies. If inconsistences exist MetaWATCH could point out exactly where they are and the form they take. The heuristics could then be used to decide if the strategies were indeed closely related and if so how they could be combined. This would allow the system to ignore spurious detail of the type seen in the previous section but still find the general features in common between strategies.

For it to be of use as a knowledge acquisition tool the strategies learned by MetaWATCH should be applicable to a wider set of problems than those examined by WATCH. However it is obvious that one strategy will not work for all possible problems within a domain. Thus MetaWATCH requires knowledge of the class of problems to which the strategy learned can be applied. At present MetaWATCH has no such knowledge. Deriving this knowledge is a non trivial task and would require much experience using the system.

Given more knowledge about types of strategies, classes of problems and the differences between them MetaWATCH could be expanded to perform an interactive style of learning more in line with the needs of a knowledge acquisition system. It could do this by posing problems for the expert to solve. These problems could be designed to explore more fully various branches of the strategy trees under consideration and distinguish between various plausible alternative structures for these branches. Such problems could also be used to explore the limits on the range of applicability of a particular strategy.

References

[1] Hayes-Roth, B.: **BB1: An Architecture for Blackboard systems that Control, Explain and Learn about their own Behavior**, Tech. Report HPP-84-16, Stanford University, 1984.

[2] Harvey, J.: **WATCH - Inductive Learning of Control Abstractions**, Unpublished report 1987.

[3] Hayes-Roth, B., Hewett, M., Johnson, M.V., Garvey, A.: **ACCORD: A Framewrok for a Class of Design Tasks**, Tech. Report KSL 88-19 1988.

[4] Johnson, M.V., Hayes-Roth, B.: **Integrating Diverse Reasoning Methods in the BB1 Blackboard Control Architecture**, Tech. Report KSL 86-76 1986.

[5] Levitt, R.E., Tommelein, I., Hayes-Roth, B., Confrey, T: **SIGHTPLAN: A Blackboard Expert System for Constraint-Based Spatial Reasoning About Construction Site Layout**, ASCE 1989?

[6] Hayes-Roth, B., Buchanan, B., Lichtarge, O., Hewett, M., Altman, R., Brinkley, J., Cornelius, C., Duncan, B., Jardetzky, O: **PROTEAN: Deriving protein structure from constraints**, Proceedings of the Fifth National Conference on Artificial Intelligence, 1986.

[7] Gans, A: **NEWWATCH: Learning Interrupted Strategies by Observing Actions**, M.S.A.I. Practicum paper, Stanford 1989.

[8] Mitchell, T., Utgoff, P., Banerji, R: **Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics**, Machine Learning, Volume II, Chapter 6.

Appendix 1 - The SightPlan object hierarchy

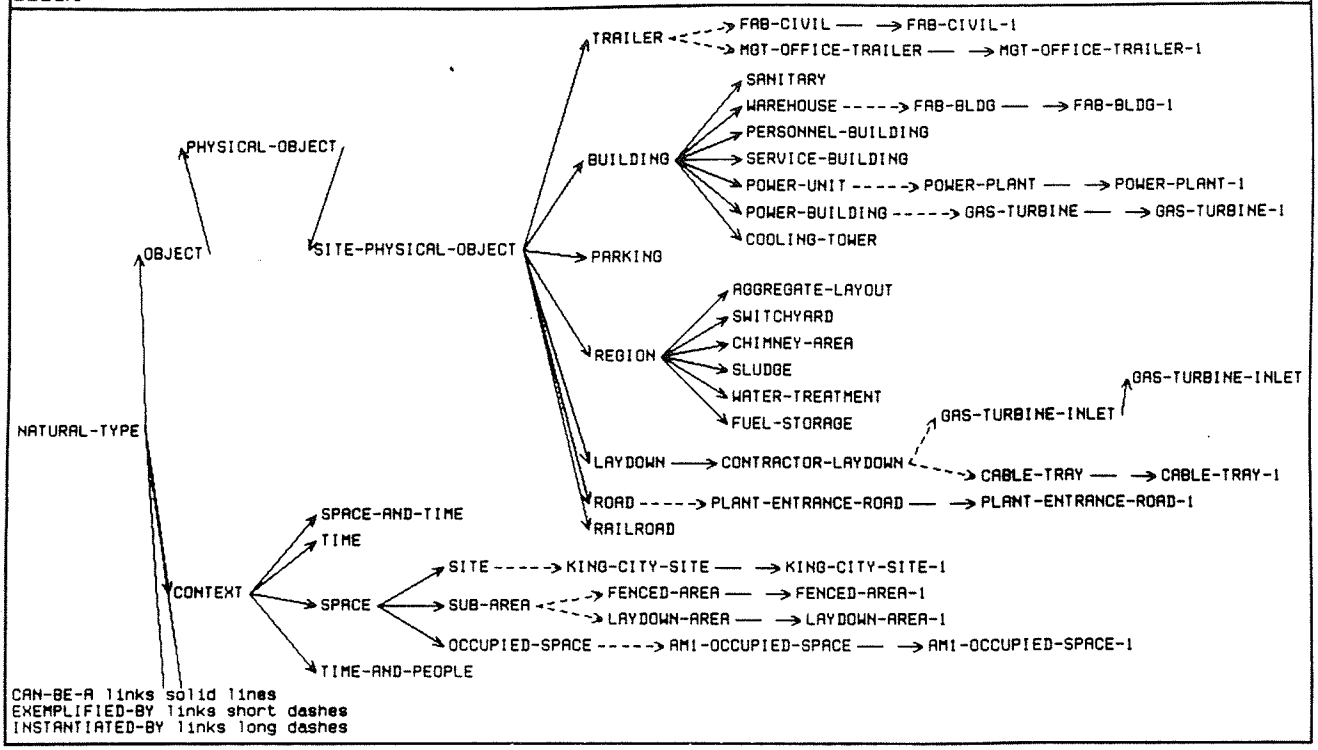
This appendix shows the SightPlan object hierarchy. This hierarchy is used by both WATCH and MetaWATCH to generalize objects to a common superobject:

HELP Info File Edit New Rename Delete Print Link Attribute Inheritance Data Display

Displaying short object names on a horizontally-oriented graph.

Sorry, can't do (BEND-EDGE (CAN-BE-A (CONCEPT.NATURAL-TYPE.CONSTRAINT CONCEPT.CONSTRAINT.STATE-CONSTRAINT))) yet.
[13:58 Finished printing User-specified Rectangle on printer Imagen-1 of Imagen-1]

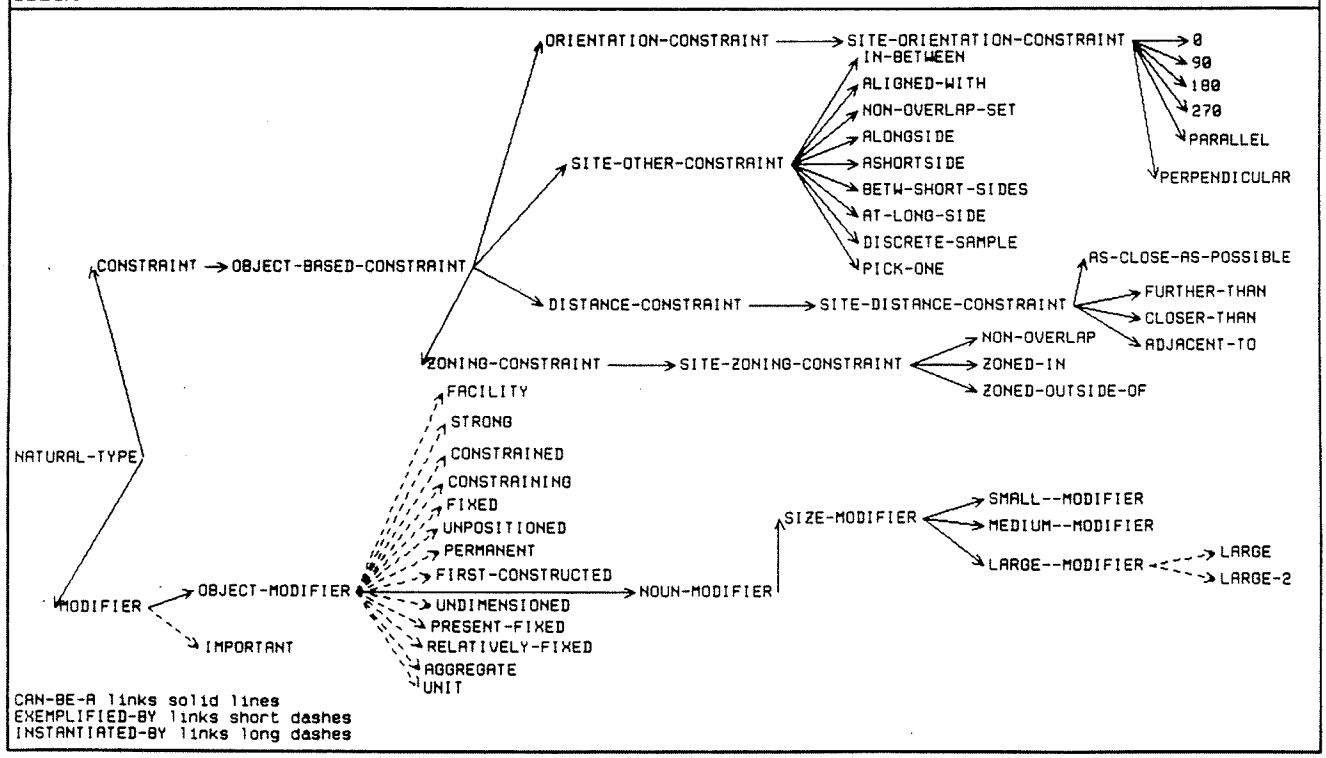
BBEdit



HELP Info File Edit New Rename Delete Print Link Attribute Inheritance Data Display

> [Resume]

BBEdit



Appendix 2 - Test cases

Each of these sets of output consists of a screen dump of the hierarchy of a skeletalplan and a description of the actions of each of its nodes.

Contents:

The SightPlan skeletalplan. This is the goal strategy which we are trying to learn.

The first skeletalplan learned by WATCH by examining set of actions used to solve a problem with the SightPlan skeletalplan.

A second skeletalplan learned by WATCH from the actions used to solve a second problem with the same skeletalplan.

The intermediate strategy MetaWATCH found by comparing the above two WATCH skeletalplans.

A third WATCH output skeletalplan.

The final MetaWATCH output - the result of comparing all three WATCH output strategies.

-*- Mode:Text; Fonts:(HL6 HL6B HL7B) -*-

SKELETALPLAN1 OBJECT DESCRIPTIONS

(STRATEGY810002
(ASSEMBLE PA1))

(STRATEGY810003
(SECURELY POSITION UNIT AGGREGATE LARGE RELATIVELY-FIXED PRESENT-FIXED UNDIMENSIONED PERMANENT UNPOSITIONED FIXED CONSTRAINING FACILITY SITE-PHYSICAL-OBJECT IN PA1 WITH OBJECT-BASED-CONSTRAINT))

(STRATEGY810004
(ASSEMBLE PA1))

(STRATEGY810005
(YOKE TRAILER AND SITE-PHYSICAL-OBJECT IN PA1 WITH SITE-DISTANCE-CONSTRAINT))

(STRATEGY810006
(DEFINE PA1))

(STRATEGY810007
(YOKE SITE-PHYSICAL-OBJECT AND FENCED-AREA-1 IN PA1 WITH IMPORTANT ZONED-IN))

(STRATEGY810008
(YOKE NATURAL-TYPE AND FAB-BLDG-1 IN PA1 WITH OBJECT-BASED-CONSTRAINT))

(STRATEGY810009
(SECURELY POSITION UNIT AGGREGATE LARGE RELATIVELY-FIXED PRESENT-FIXED UNDIMENSIONED PERMANENT UNPOSITIONED FIXED CONSTRAINING CONSTRAINED FACILITY FAB-BLDG-1 IN PA1 WITH SITE-DISTANCE-CONSTRAINT))

(STRATEGY810010
(YOKE MGT-OFFICE-TRAILER-1 AND NATURAL-TYPE IN PA1 WITH OBJECT-BASED-CONSTRAINT))

(STRATEGY810011
(SECURELY POSITION UNIT AGGREGATE LARGE RELATIVELY-FIXED PRESENT-FIXED UNDIMENSIONED PERMANENT UNPOSITIONED FIXED CONSTRAINING CONSTRAINED FACILITY MGT-OFFICE-TRAILER-1 IN PA1 WITH OBJECT-BASED-CONSTRAINT))

(STRATEGY810012
(CREATE PA1 FOR KING-CITY-SITE))

(STRATEGY810013
(INCLUDE NATURAL-TYPE IN PA1))

(STRATEGY810014
(ORIENT PA1 ABOUT POWER-PLANT-1))

(STRATEGY810015
(YOKE FENCED-AREA-1 AND FAB-BLDG-1 IN PA1 WITH ZONED-IN-103-FENCED))

(STRATEGY810016
(YOKE FENCED-AREA-1 AND MGT-OFFICE-TRAILER-1 IN PA1 WITH ZONED-IN-105-FENCED))

(STRATEGY810017
(YOKE FAB-BLDG-1 AND AM1-OCCUPIED-SPACE-1 IN PA1 WITH NON-OVERLAP-SET-103-OCCUPIED))

(STRATEGY810018
(YOKE MGT-OFFICE-TRAILER-1 AND FAB-BLDG-1 IN PA1 WITH CLOSER-THAN-103-105))

(STRATEGY810019
(YOKE GAS-TURBINE-1 AND FAB-BLDG-1 IN PA1 WITH CLOSER-THAN-103-GT))

(STRATEGY810020
(ANCHOR FAB-BLDG-1 TO POWER-PLANT-1 IN PA1 WITH AS-CLOSE-AS-103-PLANT))

(STRATEGY810021
(YOKE MGT-OFFICE-TRAILER-1 AND AM1-OCCUPIED-SPACE-1 IN PA1 WITH NON-OVERLAP-SET-105-OCCUPIED))

(STRATEGY810022
(YOKE MGT-OFFICE-TRAILER-1 AND FAB-BLDG-1 IN PA1 WITH CLOSER-THAN-103-105))

(STRATEGY810023
(YOKE PLANT-ENTRANCE-ROAD-1 AND MGT-OFFICE-TRAILER-1 IN PA1 WITH ADJACENT-TO-105-ROAD))

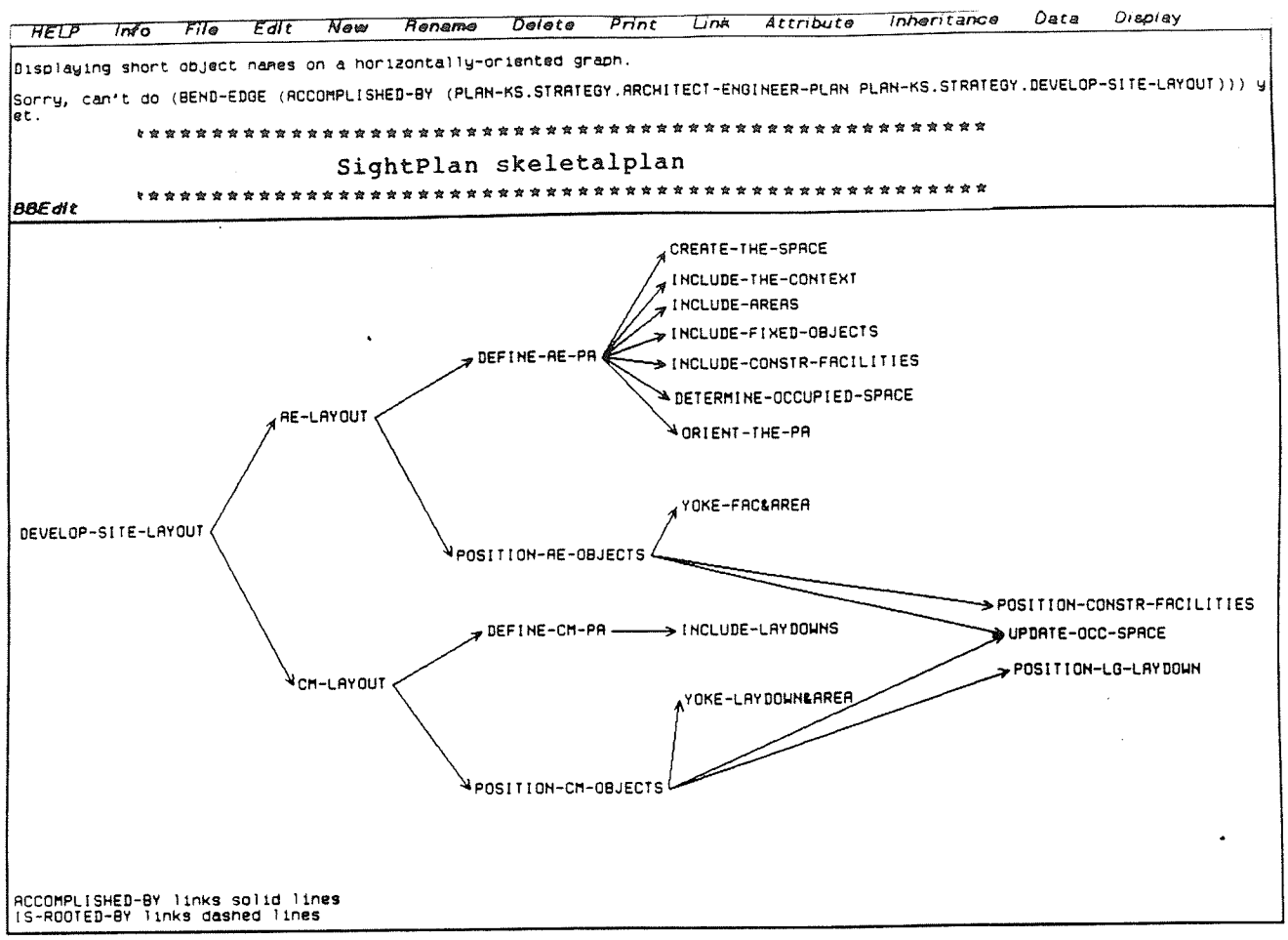
(STRATEGY810024
(YOKE MGT-OFFICE-TRAILER-1 AND FAB-BLDG-1 IN PA1 WITH CLOSER-THAN-103-105))

(STRATEGY810025
(ANCHOR MGT-OFFICE-TRAILER-1 TO POWER-PLANT-1 IN PA1 WITH PARALLEL-1-105))

(STRATEGY810026
(YOKE MGT-OFFICE-TRAILER-1 AND FAB-BLDG-1 IN PA1 WITH CLOSER-THAN-103-105))

(STRATEGY810027
(YOKE GAS-TURBINE-1 AND MGT-OFFICE-TRAILER-1 IN PA1 WITH AS-CLOSE-AS-105-GT))

(STRATEGY810028
(YOKE MGT-OFFICE-TRAILER-1 AND FAB-BLDG-1 IN PA1 WITH CLOSER-THAN-103-105))



(**DEVELOP-SITE-LAYOUT**
(**ASSEMBLE** PARTIAL-ARRANGEMENT))

(**CM-LAYOUT**
(**LAYOUT** DONE BY CONSTRUCTION MANAGER))

(**AE-LAYOUT**
(**LAYOUT** DONE BY ARCHITECT ENGINEER))

(**POSITION-CM-OBJECTS**
(**POSITION** LAYDOWN AREAS))

(**DEFINE-AE-PA**
(**DEFINE** PARTIAL-ARRANGEMENT))

(**DEFINE-CM-PA**
(**DEFINE** PARTIAL-ARRANGEMENT))

(**POSITION-AE-OBJECTS**
(**POSITION** CONSTRUCTION FACILITIES))

(**INCLUDE-THE-CONTEXT**
(**INCLUDE** CONTEXT 'SOLUTION.PARTIAL-ARRANGEMENT))

(**INCLUDE-AREAS**
(**INCLUDE** SUB-AREA 'SOLUTION.PARTIAL-ARRANGEMENT))

(**CREATE-THE-SPACE**
(**CREATE** PARTIAL-ARRANGEMENT FOR CONCEPT))

(**ORIENT-THE-PA**
(**ORIENT** PA1 ABOUT FIRST-CONSTRUCTED POWER-UNIT))

(**INCLUDE-FIXED-OBJECTS**
(**INCLUDE** FIXED OBJECT 'SOLUTION.PARTIAL-ARRANGEMENT))

(**YOKE-FAC&AREA**
(**YOKE** LARGE FACILITY SITE-PHYSICAL-OBJECT AND LARGE-2
SUB-AREA 'SOLUTION.PAR[illegible]L-ARRANGEMENT WITH IMPORTANT
CONSTRAINT))

(**YOKE-LAYDOWN&AREA**
(**YOKE** LARGE CONTRACTOR-LAYDOWN AND LARGE-2
SUB-AREA 'SOLUTION.PARTIAL-ARRANGEMENT WITH
IMPORTANT CONSTRAINT))

(**POSITION-LG-LAYDOWN**
(**POSITION** LARGE CONTRACTOR-LAYDOWN IN PA1
WITH IMPORTANT CONSTRAINT))

(**DETERMINE-OCCUPIED-SPACE**
"Determine the space that is occupied by the fixed objects on
the site; that is, roads, railroads and permanent facilities.")

(**UPDATE-OCC-SPACE**
"Keep track in a separate object of which spaces on the site
are occupied")

(**INCLUDE-LAYDOWNS**
(**INCLUDE** LAYDOWNS IN SOLUTION.PARTIAL-ARRANGEMENT)

(**INCLUDE-CONSTR-FACILITIES**
(**INCLUDE** FACILITY SITE-PHYSICAL-OBJECT IN
SOLUTION.PARTIAL-ARRANGEMENT))

(**POSITION-CONSTR-FACILITIES**
(**POSITION** LARGE FACILITY SITE-PHYSICAL-OBJECT IN
PA1 WITH IMPORTANT CONSTRAINT))

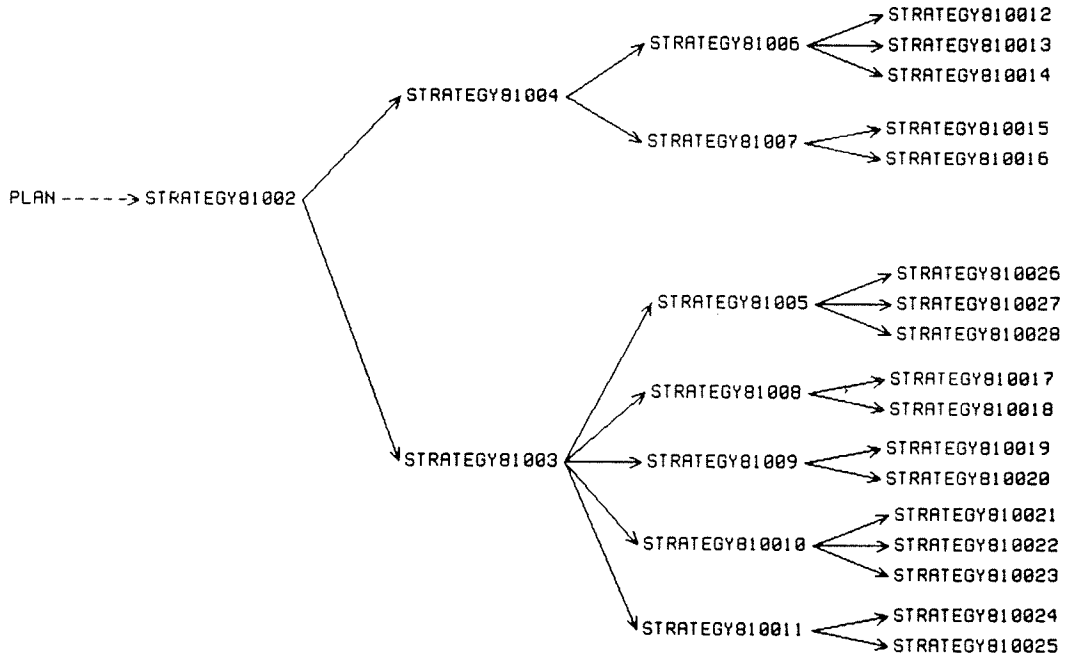
HELP Info File Edit New Rename Delete Print Link Attribute Inheritance Data Display

```
Displaying short object names on a horizontally-oriented graph.
[13:47 Finished printing Bbedit Frame 2 on printer Imagen-1 of Imagen-1]
[13:48 Finished printing User-specified Rectangle on printer Imagen-1 of Imagen-1]
```

BBEdit

- Mode:Text; Fonts:(HL6 HL6B HL7B) --

SKELETALPLAN1



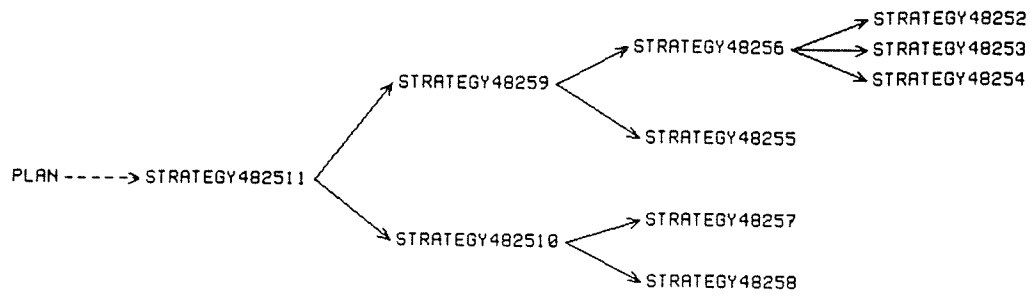

```
ACCOMPLISHED-BY links solid lines
IS-ROOTED-BY links dashed lines
```

SKELETALPLAN2 OBJECT DESCRIPTIONS

HELP Info File Edit New Rename Delete Print Link Attribute Inheritance Data Display

Displaying short object names on a horizontally-oriented graph.

BBEdit



ACCOMPLISHED-BY links solid lines
IS-ROOTED-BY links dashed lines

($STRATEGY48252

(**CREATE** PA1 FOR KING-CITY-SITE))

($STRATEGY48253

(**INCLUDE** NATURAL-TYPE IN PA1))

($STRATEGY48254

(**ORIENT** PA1 ABOUT POWER-PLANT-1))

($STRATEGY48255

(**YOKE** PARKING-1 AND SUB-AREA IN PA1 WITH IMPORTANT SITE-ZONING-CONSTRAINT))

($STRATEGY48257

(**YOKE** NATURAL-TYPE AND FAB-SHOPS-1 IN PA1 WITH IMPORTANT OBJECT-BASED-CONSTRAINT))

($STRATEGY48258

(**ANCHOR** FAB-SHOPS-1 TO POWER-PLANT-1 IN PA1 WITH AS-CLOSE-AS-103-PLANT))

($STRATEGY48256

(**DEFINE** PA1))

($STRATEGY48259

(**ASSEMBLE** PA1))

($STRATEGY482510

(SECURELY **POSITION** UNIT AGGREGATE LARGE RELATIVELY-FIXED PRESENT-FIXED UNDIMENSIONED PERMANENT UNPOSITIONED FIXED CONSTRAINING CONSTRAINED FACILITY SITE-PHYSICAL-OBJECT IN PA1 WITH OBJECT-BASED-CONSTRAINT))

($STRATEGY482511

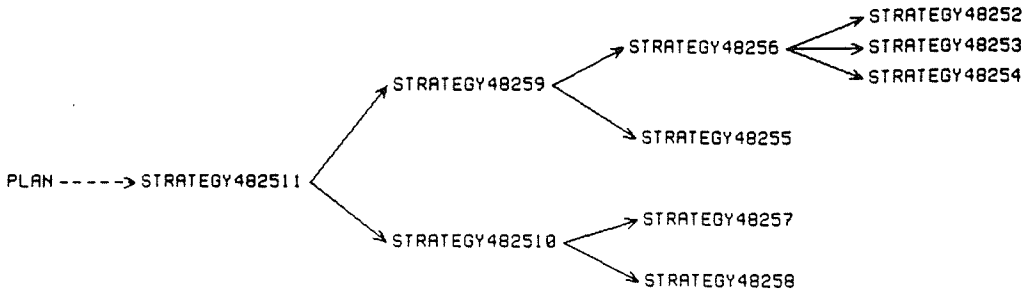
(**ASSEMBLE** PA1))

SKELETALPLAN2 OBJECT DESCRIPTIONS

HELP Info File Edit New Rename Delete Print Link Attribute Inheritance Data Display

Displaying short object names on a horizontally-oriented graph.

BBEdit



ACCOMPLISHED-BY links solid lines
IS-ROOTED-BY links dashed lines

($STRATEGY48252
(**CREATE** PA1 FOR KING-CITY-SITE))

($STRATEGY48253
(**INCLUDE** NATURAL-TYPE IN PA1))

($STRATEGY48254
(**ORIENT** PA1 ABOUT POWER-PLANT-1))

($STRATEGY48255
(**YOKE** PARKING-1 AND SUB-AREA IN PA1 WITH IMPORTANT SITE-ZONING-CONSTRAINT))

($STRATEGY48257
(**YOKE** NATURAL-TYPE AND FAB-SHOPS-1 IN PA1 WITH IMPORTANT OBJECT-BASED-CONSTRAINT))

($STRATEGY48258
(**ANCHOR** FAB-SHOPS-1 TO POWER-PLANT-1 IN PA1 WITH AS-CLOSE-AS-103-PLANT))

($STRATEGY48256
(**DEFINE** PA1))

($STRATEGY48259
(**ASSEMBLE** PA1))

($STRATEGY482510
(SECURELY **POSITION** UNIT AGGREGATE LARGE RELATIVELY-FIXED PRESENT-FIXED UNDIMENSIONED PERMANENT UNPOSITIONED FIXED CONSTRAINING CONSTRAINED FACILITY SITE-PHYSICAL-OBJECT IN PA1 WITH OBJECT-BASED-CONSTRAINT))

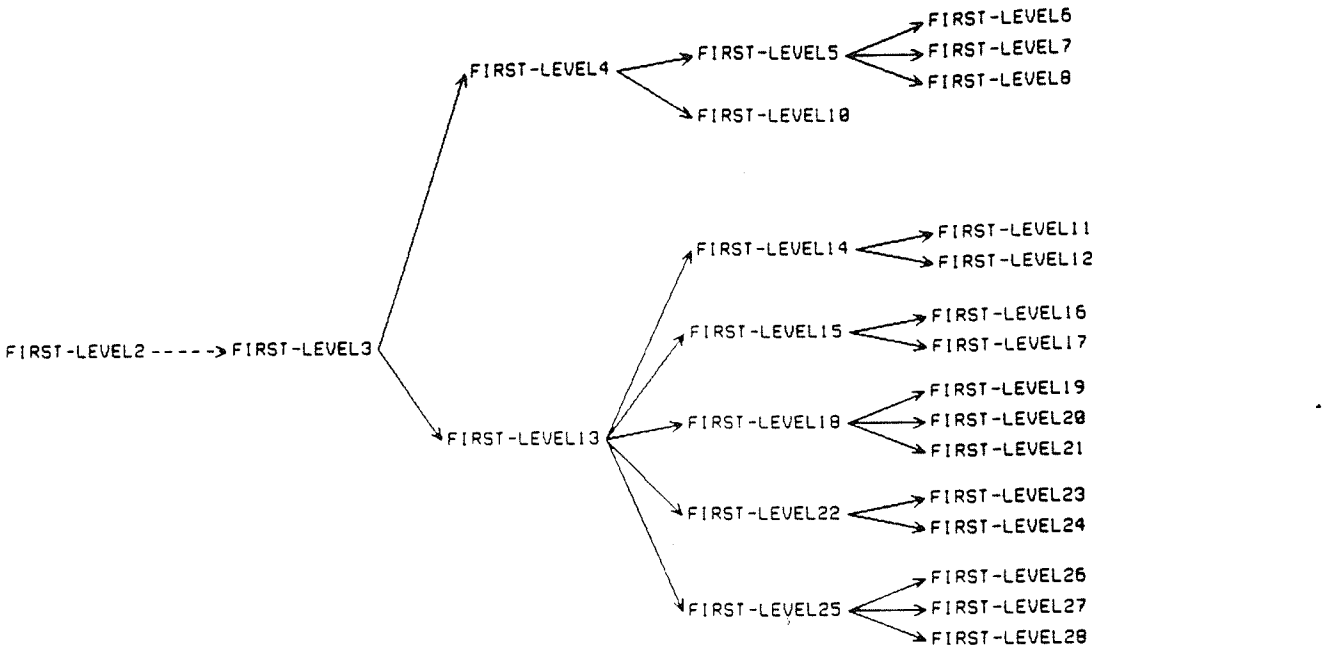
($STRATEGY482511
(**ASSEMBLE** PA1))

HELP Info File Edit New Rename Delete Print Link Attribute Inheritance Data Display

Displaying short object names on a horizontally-oriented graph.

****
First MetaWATCH Strategy - combining skeletalplans 1&2
****

BBEdit



ACCOMPLISHED-BY links solid lines
IS-ROOTED-BY links dashed lines

FIRST METAWATCH STRATEGY OBJECT DESCRIPTIONS

```
(FIRST-LEVEL3
      (ASSEMBLE PA1))

(FIRST-LEVEL4
      (ASSEMBLE PA1))

(FIRST-LEVEL5
      (DEFINE PA1))

(FIRST-LEVEL6
      (CREATE PA1 FOR SITE.CONTEXTS.SITE))

(FIRST-LEVEL7
      (INCLUDE NATURAL-TYPE IN PA1))

(FIRST-LEVEL8
      (ORIENT PA1 ABOUT POWER-PLANT-1))

(FIRST-LEVEL10
      (YOKE SITE-PHYSICAL-OBJECT AND SUB-AREA
      IN PA1 WITH IMPORTANT ZONED-IN-CONSTRAINT))

(FIRST-LEVEL11
      (YOKE FENCED-AREA-1 AND FAB-BLDG-1 IN PA1 WITH
      ZONED-IN-103-FENCED))

(FIRST-LEVEL12
      (YOKE FENCED-AREA-1 AND MGT-OFFICE-TRAILER-1
      IN PA1 WITH ZONED-IN-105-FENCED))

(FIRST-LEVEL13
      (POSITION UNIT AGGREGATE LARGE RELATIVELY-FIXED
      PRESENT-FIXED UNDIMENSIONED PERMANENT
      UNPOSITIONED FIXED CONSTRAINING FACILITY
      SITE-PHYSICAL-OBJECT IN PA1 WITH
      OBJECT-BASED-CONSTRAINT))

(FIRST-LEVEL14
      (YOKE NATURAL-TYPE IN PA1 WITH
      OBJECT-BASED-CONSTRAINT))

(FIRST-LEVEL15
      (POSITION FAB-BLDG-1 IN PA1 WITH
      SITE-DISTANCE-CONSTRAINT))

(FIRST-LEVEL16
      (YOKE GAS-TURBINE-1 AND FAB-BLDG-1
      IN PA1 WITH CLOSER-THAN-103-GT))

(FIRST-LEVEL17
      (ANCHOR FAB-BLDG-1 TO POWER-PLANT-1 I
      N PA1 WITH AS-CLOSE-AS-103-PLANT))

(FIRST-LEVEL18
      (YOKE MGT-OFFICE-TRAILER-1 AND NATURAL-TYPE
      IN PA1 WITH OBJECT-BASED-CONSTRAINT))

(FIRST-LEVEL19
      (YOKE MGT-OFFICE-TRAILER-1 AND
      AM1-OCCUPIED-SPACE-1 IN PA1 WITH
      NON-OVERLAP-SET-105-OCCUPIED))

(FIRST-LEVEL20
      (YOKE MGT-OFFICE-TRAILER-1 AND FAB-BLDG-1
      IN PA1 WITH CLOSER-THAN-103-105))

(FIRST-LEVEL21
      (YOKE PLANT-ENTRANCE-ROAD-1 AND
      MGT-OFFICE-TRAILER-1 IN PA1 WITH
      ADJACENT-TO-105-ROAD))

(FIRST-LEVEL22
      (SECURELY POSITION UNIT AGGREGATE LARGE
      RELATIVELY-FIXED PRESENT-FIXED UNDIMENSIONED
      PERMANENT UNPOSITIONED FIXED CONSTRAINING
      CONSTRAINED FACILITY MGT-OFFICE-TRAILER-1 IN
      PA1 WITH OBJECT-BASED-CONSTRAINT))

(FIRST-LEVEL23
      (YOKE MGT-OFFICE-TRAILER-1 AND FAB-BLDG-1
      IN PA1 WITH CLOSER-THAN-103-105))

(FIRST-LEVEL24
      (ANCHOR MGT-OFFICE-TRAILER-1 TO POWER-PLANT-1
      IN PA1 WITH PARALLEL-1-105))

(FIRST-LEVEL25
      (YOKE TRAILER AND SITE-PHYSICAL-OBJECT IN PA1
      WITH SITE-DISTANCE-CONSTRAINT))

(FIRST-LEVEL26
      (YOKE MGT-OFFICE-TRAILER-1 AND FAB-BLDG-1
      IN PA1 WITH CLOSER-THAN-103-105))

(FIRST-LEVEL27
      (YOKE GAS-TURBINE-1 AND MGT-OFFICE-TRAILER-1
      IN PA1 WITH AS-CLOSE-AS-105-GT))

(FIRST-LEVEL28
      (YOKE MGT-OFFICE-TRAILER-1 AND FAB-BLDG-1
      IN PA1 WITH CLOSER-THAN-103-105))
)
```

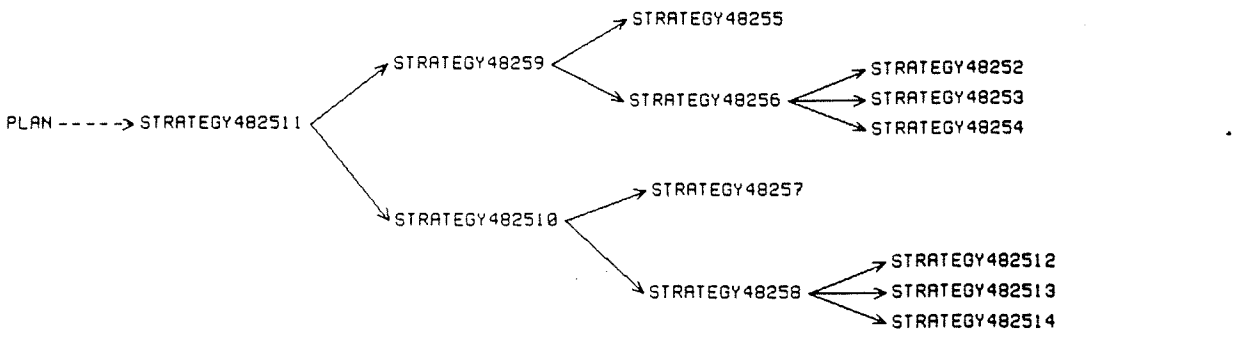
-*- Mode:Text; Fonts:(HL6 HL6B HL7B) -*-

SKELETALPLAN3 OBJECT DESCRIPTIONS

HELP Info File Edit New Rename Delete Print Link Attribute Inheritance Data Display

Displaying short object names on a horizontally-oriented graph.

BBEdit



ACCOMPLISHED-BY links solid lines
IS-ROOTED-BY links dashed lines

(STRATEGY48259
(ASSEMBLE PA1))

(STRATEGY482510
(ASSEMBLE PA1))

(STRATEGY482511
(ASSEMBLE PA1))

(STRATEGY482512
(YOKE LAYDOWN-AREA AND SPACE WITH
OBJECT-BASED-CONSTRAINT))

(STRATEGY482513
(YOKE LAYDOWN-AREA AND NATURAL-TYPE WITH
OBJECT-BASED-CONSTRAINT))

(STRATEGY482514
(ANCHOR CABLE-TRAY-1 TO POWER-PLANT-1 WITH
AS-CLOSE-AS-102-PLANT))

(STRATEGY48252
(CREATE PA1 FOR KING-CITY-SITE))

(STRATEGY48253
(INCLUDE NATURAL-TYPE IN PA1))

(STRATEGY48254
(ORIENT PA1 ABOUT POWER-PLANT-1))

(STRATEGY48255
(YOKE PARKING-1 AND SUB-AREA IN PA1 WITH
IMPORTANT SITE-ZONING-CONSTRAINT))

(STRATEGY48257
(INCLUDE SUB-AREA))

(STRATEGY48258
(POSITION LAYDOWN-AREA))

(STRATEGY48256
(DEFINE PA1))

META-WATCH OUTPUT SKELETAL-PLAN OBJECT DESCRIPTIONS

(FIRST-LEVEL3
(ASSEMBLE PA1))

(FIRST-LEVEL4
(ASSEMBLE PA1))

(FIRST-LEVEL5
(DEFINE PA1))

(FIRST-LEVEL6
(CREATE PA1 FOR SITE.CONTEXTS.SITE))

(FIRST-LEVEL7
(INCLUDE NATURAL-TYPE IN PA1))

(FIRST-LEVEL8
(ORIENT PA1 ABOUT POWER-PLANT-1))

(FIRST-LEVEL10
(YOKE SITE-PHYSICAL-OBJECT AND SUB-AREA
IN PA1 WITH IMPORTANT ZONED-IN-CONSTRAINT))

(FIRST-LEVEL11
(YOKE FENCED-AREA-1 AND FAB-BLDG-1 IN PA1
WITH ZONED-IN-103-FENCED))

(FIRST-LEVEL12
(YOKE FENCED-AREA-1 AND MGT-OFFICE-TRAILER-1 IN
PA1 WITH ZONED-IN-105-FENCED))

(FIRST-LEVEL13
(POSITION UNIT AGGREGATE LARGE RELATIVELY-FIXED
PRESENT-FIXED UNDIMENSIONED PERMANENT UNPOSITIONED
FIXED CONSTRAINING FACILITY SITE-PHYSICAL-OBJECT IN PA1
WITH OBJECT-BASED-CONSTRAINT))

(FIRST-LEVEL14
(YOKE NATURAL-TYPE IN PA1 WITH
OBJECT-BASED-CONSTRAINT))

(FIRST-LEVEL15
(POSITION FAB-BLDG-1 IN PA1 WITH
SITE-DISTANCE-CONSTRAINT))

(FIRST-LEVEL16
(YOKE GAS-TURBINE-1 AND FAB-BLDG-1 IN PA1
WITH CLOSER-THAN-103-GT))

(FIRST-LEVEL17
(ANCHOR FAB-BLDG-1 TO POWER-PLANT-1 IN PA1 WITH
AS-CLOSE-AS-103-PLANT))

(FIRST-LEVEL18
(YOKE MGT-OFFICE-TRAILER-1 AND NATURAL-TYPE IN PA1
WITH OBJECT-BASED-CONSTRAINT))

(FIRST-LEVEL32
(YOKE LAYDOWN-AREA AND NATURAL-TYPE WITH
OBJECT-BASED-CONSTRAINT))

(FIRST-LEVEL19
(YOKE MGT-OFFICE-TRAILER-1 AND AM1-OCCUPIED-SPACE-1 I
N PA1 WITH NON-OVERLAP-SET-105-OCCUPIED))

(FIRST-LEVEL20
(YOKE MGT-OFFICE-TRAILER-1 AND FAB-BLDG-1 IN PA1 WITH
CLOSER-THAN-103-105))

(FIRST-LEVEL21
(YOKE PLANT-ENTRANCE-ROAD-1 AND MGT-OFFICE-TRAILER-1
IN PA1 WITH ADJACENT-TO-105-ROAD))

(FIRST-LEVEL22
(SECURELY POSITION UNIT AGGREGATE LARGE RELATIVELY-FIXED
PRESENT-FIXED UNDIMENSIONED PERMANENT UNPOSITIONED FIXED
CONSTRAINING CONSTRAINED FACILITY MGT-OFFICE-TRAILER-1 IN
PA1 WITH OBJECT-BASED-CONSTRAINT))

(FIRST-LEVEL23
(YOKE MGT-OFFICE-TRAILER-1 AND FAB-BLDG-1 IN PA1 WITH
CLOSER-THAN-103-105))

(FIRST-LEVEL24
(ANCHOR MGT-OFFICE-TRAILER-1 TO POWER-PLANT-1 IN PA1
WITH PARALLEL-1-105))

(FIRST-LEVEL25
(YOKE TRAILER AND SITE-PHYSICAL-OBJECT IN PA1 WITH
SITE-DISTANCE-CONSTRAINT))

(FIRST-LEVEL26
(YOKE MGT-OFFICE-TRAILER-1 AND FAB-BLDG-1 IN PA1 WITH
CLOSER-THAN-103-105))

(FIRST-LEVEL27
(YOKE GAS-TURBINE-1 AND MGT-OFFICE-TRAILER-1 IN PA1 WITH
AS-CLOSE-AS-105-GT))

(FIRST-LEVEL28
(YOKE MGT-OFFICE-TRAILER-1 AND FAB-BLDG-1 IN PA1 WITH
CLOSER-THAN-103-105))

(FIRST-LEVEL29
(INCLUDE SUB-AREA IN PA1))

(FIRST-LEVEL30
(POSITION LAYDOWN-AREA IN PA1 WITH
OBJECT-BASED-CONSTRAINT))

(FIRST-LEVEL31
(ASSEMBLE PA1))

(FIRST-LEVEL33
(YOKE LAYDOWN-AREA AND NATURAL-TYPE WITH
OBJECT-BASED-CONSTRAINT))

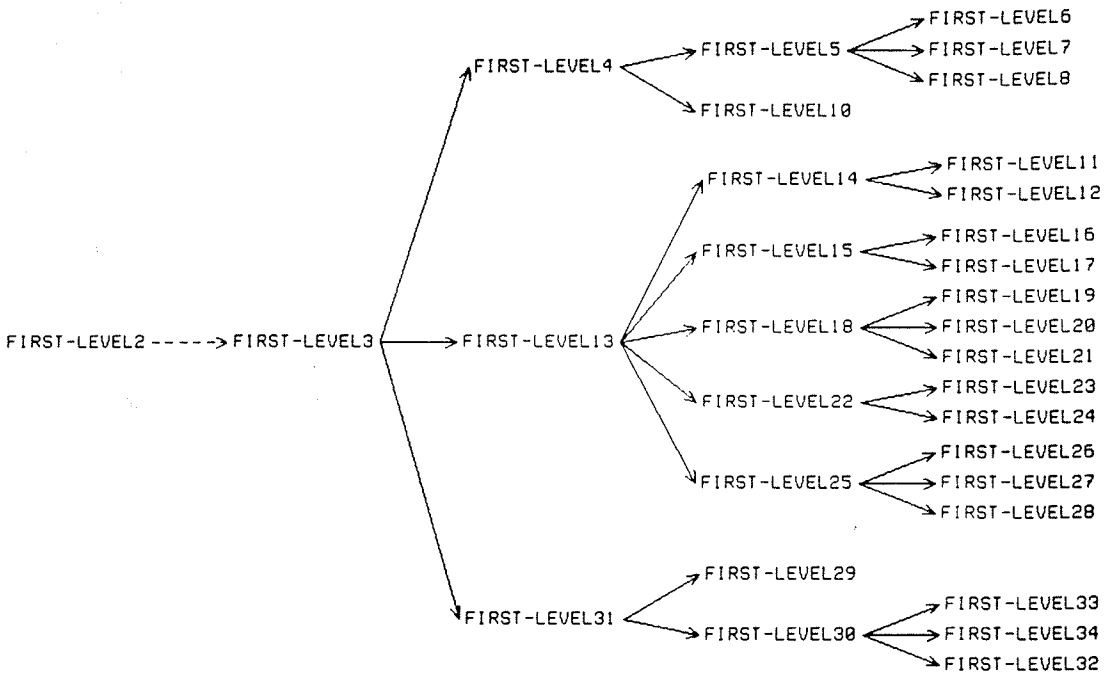
(FIRST-LEVEL34
(ANCHOR CABLE-TRAY-1 TO POWER-PLANT-1
WITH AS-CLOSE-AS-102-PLANT))

HELP Info File Edit New Rename Delete Print Link Attribute Inheritance Data Display

splaying short object names on a horizontally-oriented graph.

```
**********************************************************************
   MetaWATCH output skeletalplan - combining skeletalplans 1,2&3
**********************************************************************
```

3Edit



COMPLISHED-BY links solid lines
-ROOTED-BY links dashed lines