Hierarchical Planning

Hierarchical Task Networks: Angelic Semantics and HLA Descriptions
Abstract

In this text I attempt to explain the basics of hierarchical planning as embodied in hierarchical task networks, including structure, abstract algorithm behaviour and computational properties. Furthermore, I explore some of the more advanced features of HTN planning such as semantics, HLA descriptions and angelic search. Included is an elaboration on the properties of angelic semantics and why angelic semantics is superior to demonic semantics for HTN planning. Finally, I discuss some of the advantages and disadvantages of HTN planning and what they are mainly used for today (using concrete examples).
Table of Contents

1.1 Introduction ................................................................................................................................. 4

2.1 A Brief Introduction to HTN Planning .......................................................................................... 5
  2.1.1 Introduction to HTN .................................................................................................................. 5
  2.1.2 Basic Structure .......................................................................................................................... 5
  2.1.3 Basic Requirements for HTN .................................................................................................... 6
  2.1.4 An Example of Inventing Plans using HTN, The Milk Carton Mission ..................................... 7

2.2 HTN Semantics and HLA Descriptions ......................................................................................... 8
  2.2.1 Demonic Determinism and Semantics ....................................................................................... 8
  2.2.2 Angelic Determinism and Semantics ......................................................................................... 9
  2.2.3 Reachable Sets .......................................................................................................................... 10
  2.2.4 HLA Descriptions ................................................................................................................... 12
  2.2.5 Computational Properties of Angelic Search .......................................................................... 13

2.3 Discussing Hierarchical Planning ................................................................................................ 14
  2.3.1 Usage of Hierarchical Planning Today .................................................................................... 14
  2.3.3 Advantages of HTN .................................................................................................................. 14
  2.3.4 Disadvantages of HTN ............................................................................................................. 14

2.4 Summary ....................................................................................................................................... 15

2.5 References ..................................................................................................................................... 16
1.1 Introduction

In our daily life, we constantly plan ahead and work out in what order we want to do things. It may be simple tasks like fetching a drink from the refrigerator, or more advanced tasks like purchasing the drink at a supermarket. Regardless of the simplicity of the task, humans are generally believed to plan on a higher level of detail (Marthi, Russel & Wolfe, 2007), meaning we might plan going to the supermarket before heading to the university for afternoon classes instead of planning a sequence of more detailed actions like "rise from bed", "put on clothes", "exit through the door" and so on. Hierarchical Task Network planning works in similar ways, taking advantage of higher level planning in order to handle larger problems and tasks. HTN planners are common when inventing plans for factories or vacations where a limited amount of resources are available, and you have surely (even if you may not have realized it) come across a HTN based planner while surfing the internet.

In this text, I will attempt to provide an easy to understand perspective on HTN planning as a whole, but also explain some of the more detailed features of the semantics and HLA descriptions involved in an HTN planning system. In order to achieve this, I have studied papers and books by, among others, Peter Norvig, Stuart Russel, Jason Wolfe and Bhaskara Marthi whom are some of the leading names in the field. The text starts with a short introduction to classical planning and the basic structure and requirements of hierarchical task networks. The latter sections of the text discuss angelic, demonic semantics, HLA descriptions, computational aspects and finally some of the advantages and disadvantages of HTN planning as well as presenting some common areas where HTN planners are used today.
2.1 A Brief Introduction to HTN Planning

2.1.1 Introduction to HTN

As in biological life, in the field of artificial intelligence, inventing plans to get from a starting state to a goal state is very important for many types of intelligent agents. There are a number of different techniques agents can use to plan their courses of action, each technique varying in requirements, time/space limitations and results. To be able to plan a course of action agents need, among other things, to have access to information that states where the agent is now, what the agent can do and what goal the agent has. Additionally, the agent needs to know what the requirements are to perform an action as well as what the effect its actions are intended to have upon the world. Depending on what kind of environment (deterministic or non-deterministic) the agent operates in, the outcome of actions may or may not be clear.

Hierarchical planning works in a similar way to how it is believed that human planning works (Marthi, Russel & Wolfe, 2007) in regards to that it plans using high-level actions. For example, when planning to go to a restaurant, we plan what time we want to go and what restaurant we'd like to eat at. It is believed that we don't plan using simpler actions such as moving our legs, opening the front door or putting on the seatbelt in the car and so on, as such a plan would surely be harder to invent. The most appealing aspect of HTN planning is the high-level reasoning that I will explain further in the following sections, it is the high-level pruning of plans and commitment to probable successful plans using HLA precondition-effect descriptions that are some of the keys to why HTN planning is so computationally viable and widely used today.

2.1.2 Basic Structure

Hierarchical Task Networks (from here on abbreviated as HTN) is a hierarchical planning technique where actions are divided into different levels, or hierarchies. There are different types of actions residing within these hierarchies, one of them being High Level Actions (from here on abbreviated as HLA). These HLAs could be any action that you can divide into smaller actions, called refinements. These refinements may be sequences of actions or even other HLAs. Refinements in turn are broken down into implementations (also called primitive actions) which refers to any action that has no further refinements or implementations.
Above is a proposed example of a HTN structure. Imagine an example where we would like to travel from our apartment to the railway station. In such an example, you could think of the top HLA (in bold letters) as **GoToStation**, which will allow us to travel to the railway station given a set of preconditions. Using the HLA will result in a set of **effects** upon the world. A simple precondition for our HLA GoToStation could be that we are currently not at the station already, neatly pseudo-coded (not)AtStation). By using GoToStation, we assume that we safely arrive at the station, so the effect of the HLA should be AtStation. A HLA's preconditions and effects are determined by the preconditions and effects of its refinements and implementations. There are certain rules for which preconditions and effects are used by the HLA, as I will explain in 2.2.1 and 2.2.2. The HLA GoToStation branches into two more HLAs, for our example we will call them **DriveCar** and **TakeBus**. These in turn branch into refinements (which as I've mentioned can also be HLAs), which then branch into implementations. The implementations are contained in the lowest level of the hierarchy, and may be actions like "TurnRight", "GoForward" or any other primitive action.

### 2.1.3 Basic Requirements for HTN

HTN assumes full observability and determinism in the environment in which it operates. This makes HTN planners unsuitable for environments where the entire domain is not visible or in environments where the outcome of actions are not clear. Furthermore, HTN planners work best in static environments, unless they are equipped with replanning features (also termed "repairing", Russel & Norvig, 2010) like SIPE-2 for example. Also, it is important that any HLA that claims to reach the goal must indeed do so for the planner to be able to construct a reliable plan. This is called the downward refinement property (Russel & Norvig, 2010), and is vital to the success and computational efficiency of HTN planning.
2.1.4 An Example of Inventing Plans using HTN, The Milk Carton Mission

This section will discuss how a simple HTN planner generally would go about making a plan for a certain situation. Imagine a scenario in which our intelligent agent has been trusted with the important task of delivering a carton of milk from the dairy section of a supermarket to the checkout counter. In order to do so, we arm our agent with the incredibly well-defined hierarchical structure in Table 1 below. It is common to have an implementation (such as "CheckGoal") of the top HLA (in our case Maneuver) that checks whether or not the goal state has been reached by having the goal state as preconditions, and with an empty list of effects. What this does is when the goal state is reached, we return the plan and abort further iterations (Russel & Norvig, 2010).

<table>
<thead>
<tr>
<th>Level name</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top</td>
<td>&quot;Maneuver&quot;</td>
</tr>
<tr>
<td>HLA</td>
<td>&quot;GoToCheckout&quot;</td>
</tr>
<tr>
<td>Refinement</td>
<td>&quot;WalkRight&quot;</td>
</tr>
<tr>
<td>Implementation</td>
<td>&quot;TurnRight&quot;, &quot;GoForward&quot;</td>
</tr>
</tbody>
</table>

Tabell 1. Example of a hierarchy for the Milk Carton World.

If we assume hierarchical search is used, the algorithm uses any available precondition-effect data for the HLAs to generate an approximate high-level plan, pruning away plans that are believed not to reach the goal, the plan is then refined into implementations in order to execute it. In our example in Figur 3 below, if the planning algorithm decides that GoToCheckout probably reaches the goal state, it provides a plan which involves its implementations TurnRight and GoForward.

Let's say we also had the HLA "PutDownMilk". Had we used a classical search like breadth-first, it would have calculated all possible plans using PutDownMilk as well before returning a correct plan. The beauty of HTN when using hierarchical search is that if the effects of PutDownMilk does not seem to achieve the goal, we don't even consider using that action in a plan. This saves a lot of computational resources in large problem spaces. However, if there's more than one implementation (also termed primitive action) or refinement for a HLA that is believed to reach the goal, both implementations are computed as separate possible plans. This becomes problematic when using hierarchical search to compute plans for large problems with many HLAs, as HLAs often have more...
than one implementation, and thus cause a higher time/space complexity than needed. In section 2.2.2 I explain how we solve this problem using so called angelic search.

2.2 HTN Semantics and HLA Descriptions

2.2.1 Demonic Determinism and Semantics

I mentioned that there were certain rules governing what preconditions and effects a HLA would copy from its implementations. In a system operating through demonic determinism (Russel & Norvig, 2010) achieved by demonic semantics where the adversary (not the agent) makes the choices, the precondition-effect data of a HLA is determined by its implementations through a specific set of rules. For negative effects, it’s enough that any implementation of the HLA has that effect in order for the HLA to have that effect as well. For positive effects, all implementations have to have the positive effect in order for the HLA to use that effect. Consider the aforementioned example where we would like to travel to the railway station from our apartment.

<table>
<thead>
<tr>
<th>Preconditions</th>
<th>Level</th>
<th>Name</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>(not)AtStation</td>
<td>HLA</td>
<td>GoToStation</td>
<td>AtStation</td>
</tr>
<tr>
<td>(not)AtStation</td>
<td>Refinement</td>
<td>DriveCar</td>
<td>AtStation</td>
</tr>
<tr>
<td>(not)AtStation</td>
<td>Refinement</td>
<td>TakeBus</td>
<td>AtStation</td>
</tr>
</tbody>
</table>

Figur 3. The Station Situation.

We introduced the HLA GoToStation, which allowed us to go to the station provided that we were not already there. We also had its refinements, DriveCar and TakeBus. All of them require that we aren’t already at the station in order to travel there. As you can see, the preconditions and effects of the HLA all abide by the rules of demonic semantics. Problems arise, however, if we assume that taking the bus requires money for the ticket.

<table>
<thead>
<tr>
<th>Preconditions</th>
<th>Level</th>
<th>Name</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>(not)AtStation, Cash</td>
<td>HLA</td>
<td>GoToStation</td>
<td>AtStation, (not)Cash</td>
</tr>
<tr>
<td>(not)AtStation</td>
<td>Refinement</td>
<td>DriveCar</td>
<td>AtStation</td>
</tr>
<tr>
<td>(not)AtStation, Cash</td>
<td>Refinement</td>
<td>TakeBus</td>
<td>AtStation, (not)Cash</td>
</tr>
</tbody>
</table>

Figur 4. The Station Situation extended.

What happens now is that whenever we want to use the HLA GoToStation we’re required to have cash to use it, and whenever we use GoToStation we lose the cash as an effect. The reason for this is
that because the adversary (and not the agent) choose what implementations to use, we have to account for all possible choices. In our example, we show that this strict set of rules is not very handy because cash is required to use GoToStation even if we use the refinement DriveCar (and its implementations) even though DriveCar itself does not require cash. The solution to this is to remove the HLA GoToStation and reason on a lower level by making DriveCar and TakeBus HLAs on their own.

<table>
<thead>
<tr>
<th>Preconditions</th>
<th>Level</th>
<th>Name</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>(not)AtStation, Cash</td>
<td>HLA</td>
<td>GoToStation</td>
<td>AtStation, (not)Cash</td>
</tr>
<tr>
<td>(not)AtStation</td>
<td>HLA</td>
<td>DriveCar</td>
<td>AtStation</td>
</tr>
<tr>
<td>(not)AtStation, Cash</td>
<td>HLA</td>
<td>TakeBus</td>
<td>AtStation, (not)Cash</td>
</tr>
</tbody>
</table>

Figur 5. The Station Situation restructured.

This is a workaround to our problem, it enables us to consider DriveCar and TakeBus independently of eachother without a common HLA parent connecting them as refinements. The disturbing downside to this "solution" is that in the end it rather dramatically increases the time/space complexity because we now have two HLAs to consider instead of one. Imagine a problem involving, say 1000+, operations and it's easy to see why we're well on our way to digging our own computational grave instead of gracefully dancing through the computational meadows.

### 2.2.2 Angelic Determinism and Semantics

Another solution to the very same problem we just encountered is using angelic semantics. In a system operating through angelic semantics, because the agent makes the choices, it is enough if just one implementation of a HLA is able to reach the goal state, (Russel & Norvig, 2010). In order to achieve angelic determinism, using angelic semantics, we allow HLAs to inherit positive effects from implementations, just like negative effects in demonic semantics, as long as any implementation has that effect. (Marthi, Russel & Wolfe, 2007). Angelic semantics also introduce the feature of uncertainty in effects among implementations. Basically, if we consider the HLA GoToStation, in the above demonic semantics example we planned as if we lost our money when using the HLA even though we may have chosen the implementation where we drove the car. If we were using angelic semantics instead, we could express that an effect might happen, but only if the agent chooses the implementation featuring that effect. What this does for our station situation example in section 2.2.2 is that the HLA effects can be described as "might lose money" depending
on whether the implementation of TakeBus is chosen or not. We can illustrate this by using the same

<table>
<thead>
<tr>
<th>(not)AtStation, (maybe)Cash</th>
<th>HLA GoToStation</th>
<th>AtStation, (maybeNot)Cash</th>
</tr>
</thead>
<tbody>
<tr>
<td>(not)AtStation</td>
<td>Refinement</td>
<td>DriveCar AtStation</td>
</tr>
<tr>
<td>(not)AtStation, Cash</td>
<td>Refinement</td>
<td>TakeBus AtStation</td>
</tr>
<tr>
<td>(not)AtStation, Cash</td>
<td>Refinement</td>
<td>TakeBus AtStation</td>
</tr>
</tbody>
</table>

in 2.1.2.

Figur 6. The Station Situation with angelic semantics.

As you can see in Figure 6, if we use angelic semantics, the HLA GoToStation will possibly require cash, but it depends on what implementation the agent decides on. This makes it possible to reason using GoToStation directly, but we’re not necessarily required to have cash to do so. In other words, we retain the high-level planning that makes HTN planning so great, but we don’t limit ourselves by having HLAs with preconditions that are too strict. Furthermore, when using angelic semantics, HLAs with uncertain effects can greatly reduce the time/space complexity because it allows for the usage of angelic search, which computational properties I will elaborate on in section 2.2.4. The way angelic semantics are able to handle HLAs with more than one implementation is one of the reasons that makes it the preferred alternative over demonic semantics in HTN planning, but there is a downside to building in too much uncertainty in HLAs as well.

2.2.3 Reachable Sets

Every HLA has a state range, or **reachable set** (Russel & Norvig, 2010). This reachable set represents the scope, or reach, of all implementations of a HLA from its starting state.

Figur 7. Illustration of a HLAs reachable sets.
In Figure 7 above, the reachable set of a HLAs two implementations are illustrated by purple and green zones. If the desired goal state is intersecting a HLAs reachable set, that HLA is considered a viable plan. A HLA reaches the goal if any implementations reachable set intersects the goal. In other words, the reachable set of a HLA can be thought of as all of its implementations reachable sets combined. One of the problems with making HLAs too general in angelic semantics (if you compare demonic and angelic semantics, when using demonic semantics we were forced to use more specific HLAs) is that the reachable sets of a HLA becomes hard to calculate. If the HLA has too many implementations with uncertain effects it may result in the reachable set being impossible to determine, such a reachable set is also called a **wiggly set** (Russel & Norvig, 2010). If this is the case, we attempt to compute an approximation of the reachable set. Previously, I have presented reachable sets in a way that assumes HLAs either reach the goal or they don’t. This is not always the case, however, as it is only natural that checking whether or not a HLA actually does reach the goal state becomes complicated if the reachable set is wiggly. There are two steps on the way to dealing with this. One way is to define optimistic and pessimistic ways of calculating an approximation of the reachable set, leading to **pessimistic** and **optimistic descriptions** (Russel & Norvig, 2010) of a HLA. A pessimistic approximation is designed to understate the reachable set, while an optimistic approximation wants to overstate the set. Following this, we use the optimistic and pessimistic approximations to reason about the actual reachable set of the HLAs.

<table>
<thead>
<tr>
<th>Approximation</th>
<th>Pessimistic</th>
<th>Optimistic</th>
<th>Estimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>Reaches goal</td>
<td></td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>Don’t know</td>
<td></td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>Doesn’t reach goal</td>
<td></td>
</tr>
<tr>
<td>TRUE</td>
<td>FALSE</td>
<td>Impossible</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 8. Reasoning about Reachable Sets.*

The way we reason about the reachable set using approximations is illustrated in Figure 8. As you see in row 1 of Figure 8, we know that the HLA reaches the goal if the pessimistic and optimistic approximations reach the goal state. In row 2 however, if the optimistic approximation reaches the goal state, but the pessimistic approximation does not, we can’t be sure whether the plan works or not. Furthermore, if both approximations are false like in row 3, the HLA does not reach the goal. Additionally, given the nature of pessimistic and optimistic approximations, it’s impossible to have pessimistic approximation reaching the goal state while optimistic approximation does not. Another way of picturing how approximation in reachable sets work is by using bars. In the figure below, I've illustrated the same situation as Figure 8, row 2.

<table>
<thead>
<tr>
<th>Optimistic Approximation</th>
<th>TRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pessimistic Approximation</td>
<td>FALSE</td>
</tr>
<tr>
<td>Actual Reachable Set</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 9. An alternative way of picturing uncertain approximation.*
As Figure 9 shows, whenever pessimistic approximation does not reach the goal, but optimistic does, we can’t possibly tell whether or not the HLA will reach the goal or not. Whenever this happens, we move on to the final step of the solution, which is refining the plan further until we’re able to tell if the HLA reaches the goal or not.

### 2.2.4 HLA Descriptions

We've discussed some ways to describe HLAs that have wiggly sets, but in addition to that it's important for all HLAs to have proper descriptions in order to be able to use them for planning purposes. HLAs with descriptions that support inference and excluding of plans are said to have **complete descriptions**. For example, if we would have a HLA called "OpenFrontDoor" with a complete description in our railway station example, we could exclude any plans made using only that HLA. The reason being that no matter how many times you open the front door, it's impossible for that HLA alone to reach a goal state. In other words, any HLA that has preconditions-effects that can be used for reasoning has a complete description.

Furthermore, HLAs may have **sound descriptions**. Soundly described HLAs allow plans to reach goal states anywhere within the scope of the starting state and reachable set of the HLA. I prefer to think of this set of states as a python list.

Start S0 [S1, S2, S3, S4] S5 Goal

S0 represents our starting state, and S5 our goal state. If a HLA has a sound description, it means that we have access to any state between the starting state and goal state at any given time during the planning process. Think of our example where we wanted to travel to the railway station. If we imagine that we had a sound description for GoToStation, it would mean that we could use GoToStation to go anywhere between our apartment and the station, for example if we decided we made our goal to stop at the supermarket located halfway to the railway station. It's easy to imagine this if we drive our car, but if we take the bus it's not certain that there would be a bus stop at the supermarket. Sound descriptions are easy to work with, but in reality it's not so easy to achieve sound descriptions for HLAs (Marthi, Russel & Wolfe, 2007). Because of this, sound-intersecting descriptions are used instead.

**Sound-intersecting descriptions** work similarly to sound descriptions, but has a set of sets of states, compared to sound descriptions set of states (I imagined this being a python list) from the starting state to the reachable set. In a HLA that is soundly described, you can reach any state between the starting state and the reachable set. In a HLA with a sound-intersecting description, you can reach at least one state per set between the starting state and the reachable set. One of the ways to imagine this difference is using nested lists of states which a HLA can reach.

Start S0 [ [S1,S2] , [S3,S4] ] S5 Goal
In the sound-intersecting description, in each set, at least one of the two states can be reached, but not both simultaneously. In other words, we can reach either S1 or S2 and either S3 or S4. If we apply this description to our GoToStation HLA, one way to explain it is to imagine that we listen to music at some points during the trip. Say that S1 represents a state where you listen to music, and S2 represents a state where you don't. Since we can either listen to music or not listen to music, it's impossible for both states to be reached at any one time, and thus only one state is available to us.

2.2.5 Computational Properties of Angelic Search

While the computational efficiency of hierarchical search is significantly better than a classical search like breadth-first, angelic search in turn is more efficient than hierarchical search. The exact space/time complexities of angelic search varies depending on what type of HLA descriptions are used. Just to give you an idea of the difference in efficiency, what can be said about angelic search in comparison to breadth-first search is that breadth-first search is "horrifyingly exponential" (A. Jönsson, class lecture, September 21, 2011) while angelic search is close to linear (Russel & Norvig, 2010). If you were to compare breadth-first, hierarchical search and angelic search you would find that breadth-first search computes all refinements and implementations of a HLA in order to find out whether or not a goal state has been reached, hierarchical search reason on a higher level and prunes away some plans based on HLA descriptions. Hierarchical search, however, operate only with demonic semantics which lack the capability to handle uncertainty in HLA effects like angelic semantics. This results in more specific HLAs, which means planning on a slightly lower level, which ultimately greatly increases the time/space complexity of the operation. In a worst case scenario, such as a domain where you use one HLA with one implementation for example, however, angelic search is no better than hierarchical search (Alechina, 2010). The reason for this is that if we are unable to take advantage of angelic semantics ability to handle uncertainty in effects through the agents choice of implementations, angelic search loses its advantage over hierarchical search. Overall however, it is still better, because it is seldom the case when planning in real domains that we only have one implementation per HLA. Furthermore, angelic search is generally close to linear, but not strictly linear. The more general HLAs are used, the faster angelic search can find a plan. However, if HLAs are too general, it will result in the dreaded wiggly sets, which ultimately will increase the time/space complexity due to the fact that HLAs must be refined further and taken down to a lower level in order to determine if that HLA reaches a goal state or not.
2.3 Discussing Hierarchical Planning

2.3.1 Usage of Hierarchical Planning Today

Hierarchical planners are commonly used for inventing plans involving resource management and scheduling. One example is O-Plan (developed by Bell & Tate) that have been used by Hitachi to plan their assembly-line production, involving more than 2000 different operations (Russel & Norvig, 2010). Further applications of O-Plan involve house construction, software development, satellite planning and control and logistics. SIPE-2 is another HTN planner that has in-built plan critics and replanning features (Onder, 2010), its applications include military operation planning, mobile robot action planning and production line scheduling. There are also examples where HTN planning has been used in game industry. For example, in the real-time strategy game Warcraft II, HTN is was adapted to handle build orders issued by the player within the game (Brickman & Joshi, 2009). Another example of HTN in games is the first-person shooter game Unreal Tournament where it is used to pick strategies for computer controlled enemies (Gorniak & Davis, 2007).

2.3.3 Advantages of HTN

The main advantage of HTN planners is their ability to reason on a higher level of abstraction compared to classical planning. As aforementioned, in a standard scenario, HTN planners using angelic search have the ability to invent a plan using HLAs alone without considering the implementations of the plan on beforehand which is very effective when considering the computational aspect. This enables HTN planners with angelic search to handle larger search spaces than classical search methods such as breadth-first search. Additionally, hierarchical plans are generally easy for humans to understand (Russel & Norvig, 2010). A possible explanation for this is that humans are believed to plan on a higher level of abstraction, much like HTN planners do. HTN planners can also be combined with scheduling or resource handling in order to come up with plans that not only work but also are the most cost-effective possible for a specific domain.

2.3.4 Disadvantages of HTN

The general main disadvantage of HTN planners is that many of them require a deterministic environment and are unable to handle uncertain outcomes of actions. There are however examples of HTN planners able to handle non-deterministic environments, such as SIPE-2, a HTN planner using replanning features. Replanning or repairing features are however sub-optimal when viewed from a computational aspect, even if a planner is able to repair a plan during execution, it may hold up production or construction while replanning occurs. Furthermore, it is sometimes hard to compute exact reachable sets, leading to wiggly sets, which complicate the planning process.
2.4 Summary

In this report I have explored the basics of HTN planning, involving structure, search and some of the requirements for HTN planning to work. Additionally, I have elaborated on some of the details in HTN planning, namely angelic semantics, reachable sets and HLA descriptions, as well as reasoned about why angelic semantics are more suitable for HTN planning than demonic semantics. In the final part of the report, I have provided examples of some of the areas and assignments that HTN planners are used in today, as well as some of the advantages and disadvantages that HTN brings.
2.5 References


Paper ID: UCB/EECS-2007-89


Brickman, N. & Joshi, N. (2010). HTN Planning and Game State Management in Warcraft II. Project report for University of California at Santa Cruz.
Retrieved 2011-09-12 from http://users.soe.ucsc.edu/~nishant/CS244.pdf


O-Plan Applications
http://www.aiai.ed.ac.uk/oplan/oplan/applications.html

SIPE-2
http://www.ai.sri.com/~sipe/