A Parametric Hierarchical Planner for Experimenting Abstraction Techniques

Giuliano Armano, Giancarlo Cherchi, Eloisa Vargiu
Department of Electrical and Electronical Engineering
University of Cagliari
I-09123, Cagliari, Italy
{armano, cherchi, vargiu}@diee.unica.it

Abstract

This paper presents a parametric system, devised and implemented to perform hierarchical planning by delegating the actual search to an external planner (the "parameter") at any level of abstraction, including the ground one. Aimed at giving a better insight of whether or not the exploitation of abstract spaces can be used for solving complex planning problems, comparisons have been made between instances of the hierarchical planner and their non hierarchical counterparts. To improve the significance of the results, three different planners have been selected and used while performing experiments. To facilitate the setting of experimental environments, a novel semi-automatic technique, used to generate abstraction hierarchies starting from ground-level domain descriptions, is also described.

1 Introduction

There is experimental evidence that humans repeatedly use abstractions while solving different kinds of problems [Stillings *et al.*, 1987], thus justifying the research in the field of automated hierarchical planning.

It is apparent that abstraction is usually not effective on simple problems, due to the overhead introduced by the need of going back and forth across abstract spaces while performing the search. In other words, enforcing abstraction on simple problems may end up by wasting computational resources. Yet, under certain assumptions abstraction can significantly reduce the search time when applied to complex problems [Knoblock, 1991].

In order to investigate the impact of abstraction mechanisms on the search complexity, we devised and implemented a hierarchical wrapper able to embody any domain-independent planner provided that a compliance with the STRIPS subset of PDDL 1.2 standard [McDermott *et al.*, 1998] is ensured. The embodied planner is exploited at any level of the hierarchy, each level being characterized by its own definitions. A suitable decoupling between levels is guaranteed by using domain-specific rules that establish the correspondence between entities belonging to a level and its superior. Translation rules are given in a PDDL-like format, explicitly defined to support abstractions.

Experiments have been performed on some classical planning domains, widely acknowledged by the planning community –although not specifically tailored for abstraction. To better assess the significance of the results, three different planners have been used while performing experiments.

The remainder of this paper is organized as follows: first, some relevant work on planning by abstraction is briefly recalled, to give the reader a better insight of the issues deemed relevant. Then, the overall architecture of the system is illustrated, including the semi-automatic technique we developed for generating abstract spaces. Subsequently, experiments are described and results are discussed. Finally, conclusions are drawn and future work is outlined.

2 Related Work

Building an ordered set of abstractions for controlling the search has proven to be an effective approach for dealing with the complexity of planning tasks. This technique requires the original search space to be mapped into corresponding abstract spaces, in which irrelevant details are disregarded at different levels of granularity.

Two main abstraction mechanisms have been studied in the literature: action- and state-based. The former combines a group of actions to form macro-operators [Korf, 1987]. The latter exploits representations of the domain given at a lower level of detail; its most significant forms rely on (i) relaxed models, obtained by dropping operators' applicability conditions [Sacerdoti, 1974], and on (ii) reduced models, obtained by completely removing certain conditions from the problem space [Knoblock, 1994]. Both models, while preserving the provability of plans that hold at the ground level, perform a "weakening" of the original problem space, thus suffering from the drawback of introducing "false" solutions at the abstract levels [Giunchiglia and Walsh, 1990].

As for Knoblock's abstraction hierarchies, each predicate is associated with a unique level of abstraction –

according to the constraints imposed by the ordered monotonicity property [Knoblock, 1994]. Any such hierarchy can be obtained by progressively removing certain predicates from the domain (or problem) space.

From a general perspective, let us assume that abstractions might occur on types, predicates, and operators. Relaxed models are a typical example of predicate-based abstraction, whereas macro-operators are an example of operator-based abstraction.

3 System Architecture

The system has been called *HW[]*, standing for (parametric) *Hierarchical Wrapper*. Note that square brackets are part of the name, indicating the ability to embed an external planner; being *P* any such planner, the notation *HW[P]* shall be used to denote an instance of *HW[]* able to exploit the planning capabilities of *P*.

Figure 1 illustrates the architecture of the system, focusing on its main components, i.e., an engine and the embedded planner. The former is devoted to controlling the communication between adjacent levels, whereas the latter is exploited to perform planning at any given level of abstraction.

Any domain-independent planner can be embodied within the system, provided that compliance with the STRIPS subset of the PDDL 1.2 standard is ensured.

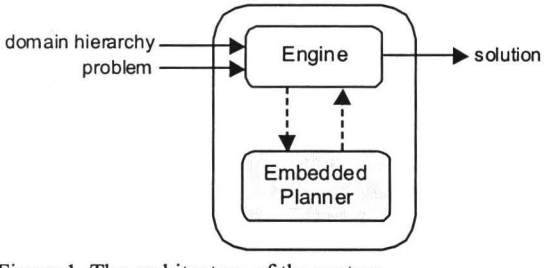


Figure 1. The architecture of the system.

Although the system supports a multiple-level hierarchy, for the sake of simplicity, in the following we assume that only one abstract level exists, giving rise to a two-level (i.e., ground and abstract) hierarchical description.

3.1 The Planning Algorithm

Once instantiated with an external planner *P*, *HW[P]* takes as inputs a ground-level problem and a structured description of the corresponding domain, including a set of rules to be used while mapping ground into abstract states and vice-versa. In fact, to perform planning at different levels of abstraction, the engine of *HW[]* must operate bi-directional translations (upwards and downwards) to permit communication between adjacent levels.

To find a solution of a given problem, first the engine of *HW[P]* translates the *init* and *goal* sections from the ground to the abstract level. *P* is then invoked to search for an abstract solution. Subsequently, each abstract operator is refined by repeatedly invoking *P*. The refinement of an abstract operator is performed by activating *P*, at the ground level, on the goal obtained by translating downward its effects. Note that the initial state of each refinement depends on the previous refinement; hence, refinements must be performed according to the order specified by the abstract plan. To avoid incidental deletion of subgoals already attained during previous refinements, they are added to the list of subgoals that results from translating downward the effects of the current abstract operator to be refined.

When the attempt to refine the current abstract solution fails *P* is invoked to find the next abstract solution, [1] unless the number of abstract solutions found so far exceeds a given threshold (*m*). If no abstract solution could be successfully refined, to ensure the completeness of the algorithm, an overall search is performed at the ground level. The whole process ends when a ground solution is found or the overall search fails.

3.2 Extending PDDL for Dealing with Abstraction

A problem and its corresponding domain are described in accordance with the standard PDDL 1.2 syntax, using the "*define problem*" and "*define domain*" statements, respectively. To describe how bi-directional communication occurs between adjacent levels an extension to the standard PDDL has been devised and adopted.

More precisely, the syntactic construct "*define hierarchy*" has been introduced. It encapsulates an ordered set of domains, together with a corresponding set of mappings between adjacent levels of abstraction. Since the mappings are given in term of types, predicates and operators, three subfields have been introduced (i.e., `:types`, `:predicates`, and `:actions`), to represent the abstraction over such dimensions. The general form of the construct is:

```
(define (hierarchy <name>)
  (:domains <domain-name>+)
  (:mapping (<src-domain> <dst-domain>)
   [:types <types-def>]
   [:predicates <predicates-def>]
   [:actions <actions-def>])*)
```

The following notation is adopted in the `:types` field to represent a clause for mapping types:

```
(abstract-type ground-type)
```

It specifies that `ground-type` becomes `abstract-type` while performing upward translations. To disregard a type, the following notation must be used:

[1] Due to the limitations of most of the existing planners, the process of incrementally querying for another solution may be simulated by preliminarily querying for m abstract solutions, to be released incrementally on demand.

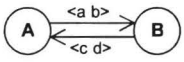
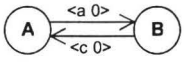
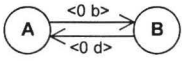
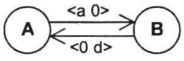
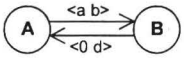
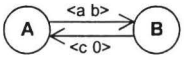
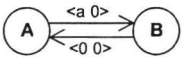
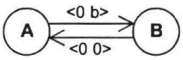
Type	Node relationships	Supporting Evidence	Action
1		(i) if a = c and b = d there is no supporting evidence for assuming that A usually precedes B in a plan, and vice-versa;	remove both edges.
		(ii) if a > c there is a high likelihood that A precedes B;	remove top edge.
		(iii) if c > a, there is a high likelihood that B precedes A.	remove bottom edge.
2		(i) if a > c there is a high likelihood that A precedes B;	remove top edge.
		(ii) if c > a, there is a high likelihood that B precedes A.	remove bottom edge.
3		A (B) negates one or more preconditions required by B (A).	remove both edges.
4		B negates one or more preconditions required by A.	remove bottom edge.
5		B negates one or more preconditions required by A.	remove both edges.
6		A and B are usually complementary or loosely-coupled actions.	remove both edges.
7		A precedes B with high likelihood.	remove top edge.
8		A negates one or more preconditions required by B.	remove the top edge.

Table 1. Heuristics for pruning the operators' graph.

```
(nil ground-type)
```

Moreover, the following notation is adopted in the `:predicates` field to represent a clause for mapping predicates:

```
((abstract-predicate ?p11 ?p21 ...)
   (ground-predicate ?p12 - t12 ?p22 - t22 ...))
```

It specifies that `ground-predicate` must be preserved while going upward and vice-versa. Note that, if no differences exist in mapping a predicate between adjacent levels, the corresponding clause can be omitted.

To disregard a predicate while performing upward translations, the following notation is used:

```
(nil
 (ground-predicate  ?p12 - t12 ?p22 - t22 ...))
```

It specifies that `ground-predicate` is not translated into any abstract-level predicate.

In addition, `abstract-predicate` can be expressed as a logical combination of some ground-level predicates.

To describe how to build the set of operators for the abstract domain, in the `:actions` field, four kinds of mapping can be expressed:

- an action remains unchanged or some of its parameters are disregarded;
- an action is removed;
- an action is expressed as a combination of actions belonging to the ground domain;
- a new operator is defined from scratch.

3.3 Generating Abstractions

To facilitate the setting of abstract spaces, as an alternative to the hand-coded approach used in [Armano *et al.*, 2003], a novel semi-automatic technique for generating abstraction hierarchies starting from ground-level domain descriptions has been devised and adopted.

From our particular perspective, performing abstraction basically involves executing two steps: (i) searching for macro-operator schemata through a priori or a posteriori analysis, (ii) selecting some of the schemata evidenced so far and translating them into abstract operators.

In this subsection, we concentrate on the task of finding macro-operator schemata throughout an a-priori analysis performed on the given domain and problem, rather than adopting the a-posteriori technique illustrated in [Armano and Vargiu, 2001], aimed at finding macro-operator schemata according to a post-mortem analysis performed on plan "chunks".

Step (i) is performed by an algorithm for building and then pruning a directed graph, whose nodes represent operators and whose edges represent relations between effects of the source node and preconditions of the destination node. In particular, for each source node A and for each destination node B, representing operators defined in the given domain, the corresponding edge is labeled with a pair of non-negative numbers, denoted by <*a b*>. The pair accounts for how many predicates A can establish (*a*) and negate (*b*) that are also preconditions of B. It is worth noting that source and destination node may coincide, thus giving rise to a self-reference.

Pruning is performed according to the domain-independent heuristics reported in Table 1. Note that the pruned graph does not contain edges labeled <0 0>, the corresponding operators being completely independent.

At this point, the most promising macro-operator schemata can be easily extracted from the pruned graph, each path being related with a candidate macro-operator.

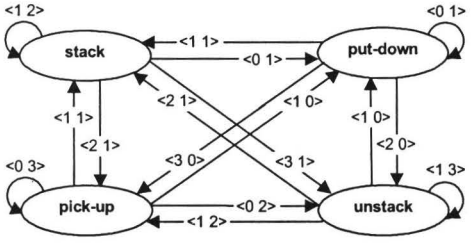


Figure 2. The directed graph (before pruning), representing static relations between operators of the blocks-world domain.

As an example, let us consider the well-known *blocks-world* domain, encompassing four operators: *stack*, *pick-up*, *unstack*, *put-down*. The corresponding graph is shown in Figure 2. Bearing in mind that the same mechanism has been applied to all operators' pairs, let us concentrate –for instance– on the relation that holds between *stack* (source node) and *pick-up* (destination node).

Considering that the effects of the *stack* operator are:

```
(not (holding ?x))
(not (clear ?y))
(clear ?x)
(handempty)
(on ?x ?y)
```

and that the preconditions of the *pick-up* operator are:

```
(clear ?x)
(ontable ?x)
(handempty)
```

we label the corresponding edge with the pair <2 1>. It is apparent that *stack* establishes two preconditions for *pick-up*, while negating another.

As for the pruning activity, Figure 3 shows the resulting graph for the blocks-world domain. [2] The resulting macro-operator schemata are (";" being used for concatenation): *pick-up;stack*, *unstack;put-down*, *pick-up;put-down*, and *stack;unstack*.

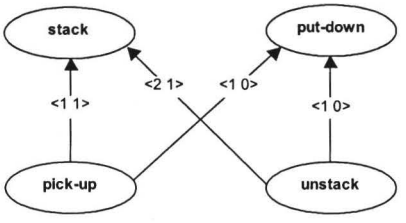


Figure 3. The directed graph (after pruning), representing static relations between operators of the blocks-world domain.

Step (ii) is performed by selecting a subset of the resulting macro-operator schemata, and by translating them into abstract operators. In principle, generating an abstract operator is not a deterministic task; for this reason in the current implementation of the system this mechanism has not yet been completely automated. Nevertheless, the simplest way of generating an abstract operator consists of deleting from the abstract level all predicates that do not occur among preconditions or effects of any selected macro-operator. This process influences (and is influenced by) the translation rules that apply to both types and predicates. For instance, the absence of a predicate as a precondition or effect of any induced abstract operator entails its deletion from the abstract level.

As for the blocks-world example, two macro-operator schemata have been disregarded (i.e., *stack;unstack* and *pick-up;put-down)*, as they do not alter the state of the domain (the resulting set of effects being empty). In fact, it is apparent that they are composed of complementary actions.

It is worth pointing out that the approach described above can be used also for generating abstractions tailored to a given problem, by simply adding a dummy operator representing the goal(s) of the problem itself. This "goal" operator has only preconditions (its set of effects being empty), representing a logic conjunct of predicates that characterize the goal of the input problem. In this way, all sequences deemed relevant to solve the problem are easily put into evidence (as they end with the "goal" operator).

4 Experimental Results

The current prototype of the system has been implemented in C++. Experiments have been performed with three planners: GRAPHPLAN [Blum and Furst, 1997], BLACKBOX [Kautz and Selman, 1998], and LPG [Gerevini and Serina, 2002]. In the following, GP, BB, and LPG shall be used to denote the GRAPHPLAN, BLACKBOX, and LPG algorithms, whereas *HW[GP]*,

[2] Since we are interested in finding macro-operators, we do not take into account self-references.

HW[BB], and *HW[LPG]* shall be used to denote their hierarchical counterparts.

To assess the capability of abstraction to improve the search, we performed some tests on five domains taken from the 1998, 2000, and 2002 AIPS planning competitions [Long, 1998; Bacchus, 2000; Long, 2002]: elevator, logistics, blocks-world, gripper, and zeno-travel. Experiments were conducted on a machine powered by an Intel Celeron CPU, working at 1200 Mhz and equipped with 256Mb of RAM. A time bound of 1000 CPU seconds has also been adopted.

#	*GP*	*HW [GP]*	*BB*	*HW [BB]*	*LPG*	*HW [LPG]*
			elevator			
1-4	0.01	0.06	0.1	0.33	0.01	0.11
3-1	0.23	0.36	1.34	1.20	0.02	0.15
4-1	1.96	0.83	1.03	1.74	0.02	0.16
4-4	10.11	0.84	311.5	1.79	0.02	0.16
5-1	364.7	2.03	180.8	2.54	0.02	0.18
7-2	--	12.04	--	3.89	0.03	0.29
			logistics			
4-2	0.68	1.22	0.27	0.46	17.93	--
5-2	0.08	0.16	0.15	0.46	0.02	--
7-0	--	10.93	4.49	2.17	2.12	--
8-1	--	16.26	2.90	3.02	1.55	--
10-0	--	43.43	8.27	3.76	2.17	--
15-0	--	203.4	10.91	6.33	0.15	--
			blocks-world			
4-0	0.34	0.32	0.16	0.67	0.02	0.08
6-0	3.04	1.82	0.26	1.68	0.05	0.23
8-0	31.61	11.13	0.92	2.46	0.36	0.31
10-0	--	--	6.82	5.00	0.62	0.67
11-0	--	--	16.23	4.25	4.23	0.83
14-0	--	--	--	9.84	5.00	1.91
15-0	--	--	--	--	7.49	2.07
17-0	--	--	--	--	33.93	3.49
20-0	--	--	--	--	66,78	7.88
22-0	--	--	--	--	183.16	12.21
25-0	--	--	--	--	668.98	24.94
			zeno-travel			
1	0.02	0.52	0.22	0.36	0.02	0.03
8	--	42.55	0.94	2.36	0.14	0.49
9	--	--	0.34	3.37	0.13	1.08
11	--	--	11.20	2.78	0.16	1.06
13	--	--	62.99	20.52	0.42	2.47
14	--	--	--	20.04	3.90	21.93
			gripper			
2	4.72	0.56	0.42	0.63	0.02	0.07
3	7.91	1.73	5.22	1.20	0.02	0.12
4	18.32	2.63	268.7	1.55	0.02	0.14
5	57.21	4.38	421.1	1.54	0.03	0.15
6	--	7.97	586.4	2.26	0.03	0.17
9	--	24.29	--	3.63	0.05	0.36

Table 2. Performance comparison of *BB*, *GP*, and *LPG* with their hierarchical counterparts.

All domains have been structured according to a ground and an abstract level, the latter having been generated following the approach described in the previous subsection. For each domain, several tests have been performed –characterized by increasing complexity. Table 2 compares the CPU time of each planner over the set of problems taken from the AIPS planning competitions. Dashes show problem instances that could not be solved by the corresponding system within the adopted time-bound.

Elevator. Experiments show that –for GP and BB– the CPU time increases very rapidly while trying to solve problems of increasing length, whereas *HW[GP]* and *HW[BB]* keep solving problems with greater regularity (although the relation between number of steps and CPU time remains exponential). LPG is able to solve long plans in a very short time, thus doing away with the need to resort to *HW[LPG]*.

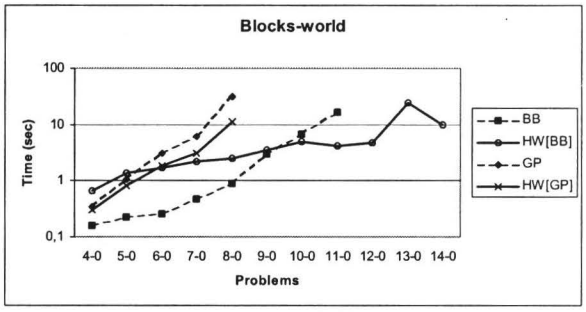


Figure 4. CPU time comparisons in the blocks-world domain.

Logistics. In this domain GP easily solves problems up to a certain length, but it is unable to solve problems within the imposed time limits if a given threshold is exceeded. On the other hand, *HW[GP]* succeeds in solving problems of increasing length without encountering the above difficulties. BB performs better than *HW[BB]* for small problems, whereas *HW[BB]* outperforms BB on more complex problems. LPG is able to solve long plans in a few seconds at the most. For unknown reasons LPG was not able to refine any abstract operator when invoked by the engine of *HW[LPG]*.

Blocks-world. Tests performed on this domain reveal a similar trend for GP and *HW[GP]*, although the latter performs slightly better than the former. BB performs better than *HW[BB]* for simple problems, whereas *HW[BB]* outperforms BB on problems of medium complexity. LPG is able to solve problems whose solution length is limited to 100 steps. In this domain, *HW[LPG]* clearly outperforms LPG on more complex problems.

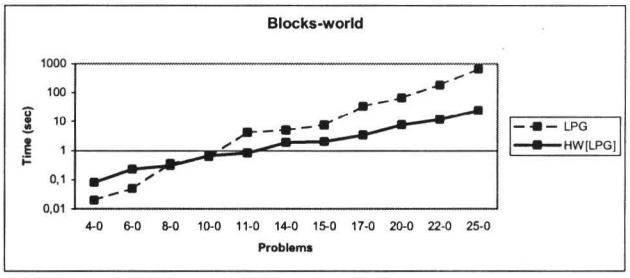


Figure 5. CPU time comparisons in the blocks-world domain.

Zeno-travel. Unfortunately, in this domain, neither GP nor *HW[GP]* are able to successfully tackle most of the problems of this domain. An improvement of *HW[BB]*

over BB can be observed, similar to the one shown for the blocks-world domain. LPG is able to solve long plans in a few seconds at the most, thus avoiding the need to resort to *HW[LPG]*.

Gripper. For the gripper domain, both *HW[GP]* and *HW[BB]* clearly outperform their non-hierarchical counterparts. LPG is able to solve long plans in a very short time.

For the sake of brevity, only two plots (i.e., Figures 4-5 concerning the blocks-world domain) of relative performances –i.e. non hierarchical vs. hierarchical– are reported.

5 Conclusions and Future Work

In this paper a parametric system has been presented, devised to perform hierarchical planning by delegating the actual search to an external planner (the parameter). Aimed at giving a better insight of whether or not the exploitation of abstract spaces can be useful for solving complex planning problems, comparisons have been made between any instances of the hierarchical planner and its non-hierarchical counterpart.

To better investigate the significance of the results, three different planners have been used to make experiments. To facilitate the setting of experiments, a novel semi-automatic technique for generating abstract spaces has been devised and adopted. Experimental results highlight that abstraction is useful for classical planners, such as GP and BB. On the contrary, the usefulness of resorting to hierarchical planning for the latest-generation planner used for experiments (i.e., LPG) clearly emerges only in the blocks-world domain.

As for future work, we are currently addressing the problem of automatically generating abstract operators.

References

[Armano and Vargiu, 2001] Giuliano Armano and Eloisa Vargiu. An Adaptive Approach for Planning in Dynamic Environments. *Proceedings of the International Conference on Artificial Intelligence (IC-AI 2001), Special Session on Learning and Adapting in AI Planning*, pages 987--993, Las Vegas, Nevada, June 2001.

[Armano *et al.*, 2003] Giuliano Armano, Giancarlo Cherchi, and Eloisa Vargiu. Experimenting the Performance of Abstraction Mechanisms through a Parametric Hierarchical Planner. *Proceedings of IASTED International Conference on Artificial Intelligence and Applications (AIA'2003)*, Innsbruck, Austria, Febbraio 2003.

[Bacchus, 2000] Fahiem Bacchus. Results of the AIPS 2000 Planning Competition, 2000.
URL: http://www.cs.toronto.edu/aips2000.

[Blum and Furst, 1997] Avrim L. Blum and Marrick L. Furst. Fast Planning through Planning Graph Analysis. *Artificial Intelligence*, 90(1-2):279--298, 1997.

[Gerevini and Serina, 2002] Alfonso Gerevini and Ivan Serina. LPG: A Planner Based on Local Search for Planning Graphs. *Proceedings of the 6th International Conference on AI Planning and Scheduling*, AAAI Press, Menlo Park, 2000.

[Giunchiglia and Walsh, 1990] Fausto Giunchiglia and Toby Walsh. A theory of Abstraction, Technical Report 9001-14, IRST, Trento, Italy, 1990.

[Kautz and Selman, 1998] Henry Kautz and Bart Selman. BLACKBOX: A New Approach to the Application of Theorem Proving to Problem Solving. In *Working notes of the Workshop on Planning as Combinatorial Search, AIPS-98*, pages 58--60, Pittsburg, PA, 1998.

[Korf, 1987] Rich E. Korf. Planning as Search: A Quantitative Approach. *Artificial Intelligence*, 33(1):65--88, 1987.

[Knoblock, 1991] Craig A. Knoblock. Search Reduction in Hierarchical Problem Solving. *Proceedings of the 9th National Conference on Artificial Intelligence*, pages 686--691, Anaheim, CA, 1991.

[Knoblock, 1994] Craig A. Knoblock. Automatically Generating Abstractions for Planning. *Artificial Intelligence*, 68(2):243--302, 1994.

[Long, 2002] Derek Long. Results of the AIPS 2002 Planning Competition, 2000.
URL: http://www.dur.ac.uk/d.p.long/competition.html.

[Long, 1998] Derek Long. The AIPS-98 Planning Competition. *AI Magazine*, 21(2):13--33, 1998.

[McDermott *et al.*, 1998] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Anshwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL – The Planning Domain Definition Language, Technical Report CVC TR-98-003 / DCS TR-1165, Yale Center for Communicational Vision and Control, October 1998.

[Sacerdoti, 1974] Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115--135, 1974.

[Stillings *et al.*, 1987] Neil A. Stillings, Christopher H. Chase, Mark H. Feinstein, Jay L. Garfield, Edwina L. Rissland, and Steven E.Weisler. *Cognitive Science: An Introduction*. MIT Press, Cambridge, Massachusetts, 1987.