Automatic Generation of Macro-Operators from Static Domain Analysis

Giuliano Armano and **Giancarlo Cherchi** and **Eloisa Vargiu** [1]

Abstract. The attempt of dealing with the complexity of planning tasks by resorting to abstraction techniques is a central issue in the field of automated planning. Although the generality of the approach has not been proved always useful on domains selected for benchmarking purposes, in our opinion it will play a central role as soon as the focus will move from artificial to real problems. This paper addresses the problem of how to identify macro-operators starting from a ground-level description of a domain, to be used for generating useful abstract-level descriptions. In particular, a preliminary release of a system devised to automatically generate abstraction hierachies has been implemented. The system is able to take into account also state invariants, to resolve ambiguities that may arise while performing abstraction. Experimental results highlight the ability of the system to identify suitable macro-operators, used as a starting point for populating the abstract level. Such macro-operators usually represent good alternatives to those extracted by a knowledge engineer after a thorough (and sometimes painful!) domain analysis.

1 INTRODUCTION

Complex environments are difficult to handle by traditional planning methods, since the search space can be very large, even for relatively simple problems. The issue of dealing with the increasing complexity of the problems is going to play a central role as soon as planners will be used to solve problems encountered in real-life applications. In the past, abstraction techniques have been used in a variety of planning systems, and have proven to be effective when applied to problems of medium/high complexity [4]. The most relevant abstraction techniques proposed in the literature are: (i) action-based ([11]), (ii) state-based ([13] and [10]), (iii) Hierarchical Task Networks ([7]), and (iv) case-based ([6]). The performance of planners can also be improved by exploiting the knowledge about the domain. In particular, the fact that state invariants can play an important role in "compiling" planning domains is widely acknowledged (a detailed discussion can be found in [8]).

This paper addresses the problem of how to identify macro-operators starting from a ground-level description of a domain, to be used for generating useful abstract-level descriptions. Compared to our previous work ([4], [1], and [3]), this paper reports a step further, in the direction of fully automatizing the process. The process of macro-operators extraction involves the exploitation of state invariants, useful for solving the problem of parameters' unification.

[1] Department of Electrical and Electronical Engineering - University of Cagliari, Italy email: {armano, cherchi, vargiu}@diee.unica.it

2 MACRO-OPERATORS GENERATION

Basically, a planning domain can be defined in terms of predicates and operators. [2] Although, in principle, abstraction can be performed along both such dimensions, this paper is mainly concerned with the latter –in particular, with the automatic extraction of macro-operators. Figure 1 depicts the architecture of the *DHG* system –standing for Domain-oriented Hierarchy Generator– devised to generate abstraction hierarchies. Starting from a planning domain description, *DHG* is able to find a set of relevant sequences of operators by resorting to the *domain analyzer* module. First, a directed graph G is built, containing information about the dependencies between ground operators (nodes represent operators and edges represent relations between effects of the source node and preconditions of the destination node). From each acyclic path a relevant sequence of operators could be extracted; nevertheless, considering all possible paths would end up to a large amount of sequences. Hence, a pruning activity on G –yielding the pruned graph G_p– is performed according to suitable, domain independent, heuristics (see [4] for further details). For each relevant sequence a corresponding macro-operator is generated, whose pre- and post-conditions are evaluated from pre- and post-conditions of the operators belonging to the sequence. Generating pre- and post-conditions starting from a sequence of operators with variable parameters involves an unification process, which may lead to semantic inconsistencies. For instance, while considering predicates that account for spatial relations –e.g., *(at ?o - object ?l - location)* used in the *Logistics* domain– there cannot be two predicates stating that the same object is in two different locations. The corresponding state invariant, not explicitly stated in the domain description, can be retrieved using TIM ([8]). The information about the domain, enriched with invariants, allows to correctly unify macro-operators' parameters. To automatically build the domain hierarchy, the *hierarchy generator* module requires a set of mapping functions that contain the translation rules (on types, predicates, operators, and invariants) between two adjacent levels of abstraction. These are expressed through the *:mapping* clause of the *define hierarchy* statement, described in [2]. In order to deal with state invariants, the following extension to the representation has been adopted:

```
<mapping-def>::=
  (:mapping (<src-domain> <dst-domain>)
    [:types <types-def>]
    [:predicates <predicates-def>]
    [:actions <actions-def>]
    [:invariants <invariants-def>])
```

[2] A particular kind of unary predicates can also be taken into account, giving rise to a third kind of entities -i.e., types- possibly organized according to a suitable "is-a" hierarchy

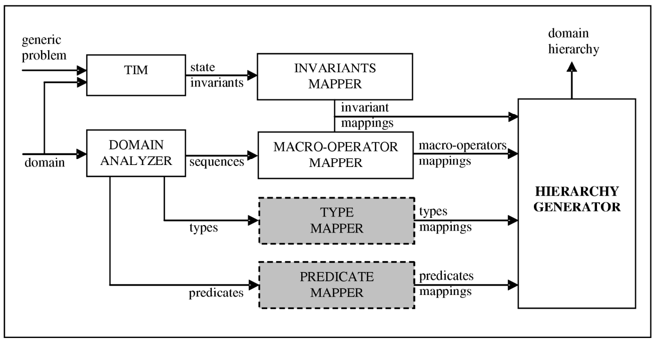

Figure 1. The DHG architecture.

In short, one *:invariants* statement for each mapping definition between two adjacent levels has been added, to support three kinds of invariants (identity, state membership, and uniqueness of state membership). The *:invariants* statement can be used to include the information about state invariants, either hand-coded or automatically generated (using TIM). Given the mapping functions, abstract operators and predicates can be generated according to a simple strategy: for each macro-operator a suitable abstract operator is generated, whose pre- and post-conditions are made coincident with those of the selected macro-operator; predicates at the abstract level are the same of the ground level, except for those not involved in any pre- or post-condition of the abstract operators.

3 EXPERIMENTAL RESULTS

To assess the *DHG* system, we compared the automatically-generated domain hierarchies with the ones hand-coded by a knowledge engineer, and characterized by mappings on types, predicates, and operators. Some domains, taken from the planning competitions ([12], [5]), including *Depots*, *Blocks-World* and *Elevator* (*simple miconic*), have been selected for benchmarking purposes. The corresponding domain hierarchies have been used as input for the *HW[]* system ([4]), able to exploit any external PDDL-compliant planner while searching for solutions at any required level of abstraction. Experiments have been performed using *FF* ([9]) as external planner, being *HW[FF]* the resulting system.

The abstract level found by *DHG* for the *Depots* domain is composed by four abstract operators. Two of them (*lift* and *drop*) are identical to those defined at the ground level, while the others are obtained from the sequences *drive;load* and *drive;unload*. The hand-coded abstraction hierarchy defines two abstract operators (obtained from the sequences *drive;unload;drop* and *drive;lift;load*), disregards the *lifting* predicate, and substitutes *depot* and *distributor* with the supertype *place*. Experiments show that the performances of *HW[FF]* using the automatically-generated hierarchy (which does not include abstraction on types or predicates) are in general slightly worse than those obtained by resorting to the hand-coded hierarchy (the difference is about 25%).

The abstract level found by *DHG* for the *Elevator* domain is composed by four abstract operators (obtained from the sequences *up;board*, *up;depart*, *down;board*, and *down;depart*). The corresponding hand-coded hierarchy defines two abstract operators (*load* and *unload*), and disregards two predicates (*lift-at* and *above*). The performance measured while feeding *HW[FF]* with the hierarchy found by *DHG* is about 20% worse than the one obtained by running *HW[FF]* with the hand-coded hierarchy.

The abstract level found by *DHG* for the *Blocks-World* domain is composed by two abstract operators (obtained from the sequences *pick-up;stack* and *unstack;put-down*). The corresponding hand-coded hierarchy shows an abstract domain composed by the same operators, although the predicates *handempty* and *holding* have been disregarded. In this domain, time intervals are approximately the same, since the hierarchy obtained from *DHG* is nearly identical to the one coded by hand. [3]

4 CONCLUSIONS

The automatic definition of macro-operators is one of the most important steps in the task of abstracting a planning domain. In this paper, a technique devised to perform this task is briefly described, as implemented by the *DHG* system. Experimental results show that the slightly negative impact on performances obtained by resorting to the automatic generation of abstraction hierarchies is more than counterbalanced by the fact that a negligible effort is required to the knowledge engineer in order to obtain suitable abstractions.

REFERENCES

[1] G. Armano, G. Cherchi, and E. Vargiu, *Experimenting the Performance of Abstractions Mechanisms through a Parametric Hierarchical Planner*, 399–404, Proceedings of IASTED International Conference on Artificial Intelligence and Applications(AIA 2003), Innsbruck (Austria), 2003.

[2] G. Armano, G. Cherchi, and E. Vargiu, *An Extension to PDDL for Hierarchical Planning*, 1–6, Workshop on PDDL (ICAPS'03), Trento (Italy), 2003.

[3] G. Armano, G. Cherchi, and E. Vargiu, *Generating Abstractions from Static Domain Analysis*, Workshop dagli Oggetti agli Agenti - Sistemi Intelligenti e Computazione Pervasiva (WOA'03), Cagliari (Italy), 2003.

[4] G. Armano, G. Cherchi, and E. Vargiu, *A Parametric Hierarchical Planner for Experimenting Abstraction Techniques*, 936–941, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03), Acapulco (Mexico), 2003.

[5] F. Bacchus. Results of the aips 2000 planning competition, 2000. Url: http://www.cs.toronto.edu2000.

[6] R. Bergmann and W. Wilke, 'Building and refining abstract planning cases by change of representation language', *Journal of Artificial Intelligence Research (JAIR)*, **3**, 53–118, (1995).

[7] K. Erol, J. Hendler, and D.S. Nau, *HTN Planning: Complexity and Expressivity*, 1123–1128, Proceedings of the Twelveth National Conference on Artificial Intelligence (AAAI-94), AAAI Press / MT Press, Seattle, WA, 1994.

[8] M. Fox and D. Long, 'The automatic inference of state invariants in tim', *Journal of Artificial Intelligence Research (JAIR)*, **9**, 367–421, (1998).

[9] J. Hoffmann and B. Nebel, 'The FF planning system: Fast plan generation through heuristic search', *Journal of Artificial Intelligence Research (JAIR)*, **14**, 253–302, (2001).

[10] C.A. Knoblock, 'Automatically generating abstractions for planning', *Artificial Intelligence*, **68(2)**, 243–302, (1994).

[11] R.E. Korf, 'Planning as search: A quantitative approach', *Artificial Intelligence*, **33(1)**, 65–88, (1987).

[12] D. Long. Results of the aips 2002 planning competition, 2002. Url: http://www.dur.ac.uk/d.p.long/competition.html.

[13] E.D. Sacerdoti, 'Planning in a hierarchy of abstraction spaces', *Artificial Intelligence*, **5**, 115–135, (1974).

[3] The fact that –in the hand-coded hierarchy– *holding* and *handempty* are disregarded at the abstract level clearly does not introduce a substantial improvement; in fact, *holding* does not appear in preconditions or effects of any macro-operator, and there is no macro-operator that negates the *handempty* predicate.