Beyond Classical Planning: Procedural Control Knowledge and Preferences in State-of-the-Art Planners Revisited

Jorge A. Baier[†] **Christian Fritz**[†] **Meghyn Bienvenu**[‡] **Sheila A. McIlraith**[†]

[†]Department of Computer Science,
University of Toronto,
Toronto, ON M5S 3G4, CANADA
{jabaier,fritz,sheila}@cs.toronto.edu

[‡]IRIT, Université Paul Sabatier
31062 Toulouse Cedex, FRANCE
bienvenu@irit.fr

Abstract

Real-world planning problems can require search over thousands of actions and may yield a multitude of plans of differing quality. To solve such real-world planning problems, we need to exploit domain control knowledge that will prune the search space to a manageable size. And to ensure that the plans we generate are of high quality, we need to guide search towards generating plans in accordance with user preferences. Unfortunately, most state-of-the-art planners cannot exploit control knowledge, and most of those that can exploit user preferences require those preferences to only talk about the final state. Here, we report on a body of work that extends classical planning to incorporate procedural control knowledge and rich, temporally extended user preferences into the specification of the planning problem. Then to address the ensuing nonclassical planning problem, we propose a broadly-applicable *compilation technique* that enables a diversity of state-of-the-art planners to generate such plans without additional machinery. While our work is firmly rooted in AI planning it has broad applicability to a variety of computer science problems relating to dynamical systems.

Preamble

This paper is reprinted from the Proceedings of AAAI-08 where it appeared as a Nectar report entitled *Beyond Classical Planning: Procedural Control Knowledge and Preferences in State-of-the-Art Planners* (Baier *et al.* 2008). The text that follows has been augmented with data that appeared in (Baier, Fritz, & McIlraith 2007) and some of the discussion updated to include mention of new related work.

We wanted to share this work with participants of the ICAPS 2009 workshop on Generalized Planning. In this work, we do *not* address the problem of how to synthesize a generalized plan with complex control structures such as conditionals and loops. Rather we address the problem of how to take a generalized plan, in this case, represented in a Golog-inspired procedural programming language akin to Algol, and synthesize a problem-specific instance of it – a sequential plan – that will achieve a particular goal. Our objective is to do so in such a way that enables us to exploit the latest advances in state-of-the-art classical planning, and in particular, heuristic search. To this end, we treat a generalized plan – a plan that has been hand-crafted in a Golog-inspired language to address a family of planning problems – as domain control knowledge. We then compile this domain control knowledge into a classical planning domain so that it directs the search of any PDDL-compliant classical planner. We also show how such a planner can be further augmented with preferences so that we can optimize the instantiation of our generalized plan with respect to certain properties of a plan that an individual deems desirable. As discussed at the end of this paper, we have recently done similar work customizing the instantiation of generalized plans represented in Hierarchical Task Networks (Sohrabi, Baier, & McIlraith 2009).

The contribution of this work is not that we can generate an instance of a given generalized plan, but that we propose an approach to do so very efficiently, and that we can further refine the generation of this plan instance to optimize for custom-specified plan properties.

Introduction

Planning has been a significant area of AI research for decades, dating at least as far back as Newell and Simon's General Problem Solver (GPS). For much of this time, the planning problem has been specified in terms of a domain theory that describes the preconditions and effects of actions, a description of the initial state, and a final-state goal formula – a set of properties that must hold upon successful execution of the plan. This specification of planning lends itself well to study, but does not capture many of the needs of real-world planning systems.

To substantiate this claim, consider the oft-cited example of travel planning on the web. The search space for this problem is enormous. There are tens of thousands of different "actions" that can be performed on the web, and hundreds of different ways to book flights and hotels online, and when you consider all the groundings for specific origins, destinations and days, the search space becomes far too large to manage with classical planning techniques. And yet we humans plan our travel on the web all the time, relatively seamlessly. We do so by utilizing a script of how to plan travel – *domain control knowledge* that helps guide our search for a plan. Even with such a script constraining the solutions, we still get a large number of solutions – some of higher quality than others. Indeed, our complex personal preferences over such things as departure times, airlines, and travel bonus points, all play a role in softly constraining and guiding our search towards high-quality plans.

So what's the problem with limiting ourselves to final-state goals? In specifying a planning problem a user may not care exclusively about what holds in the final state, but may equally care about *how* the goal is achieved – properties of the world that are to be achieved, maintained or avoided during plan execution, or adherence to a particular way of doing something. These are legitimate goals of a planning problem, that are "temporally extended" rather than "final state". Further, the user may have insight into how the plan should be realized from a search perspective, and may wish to provide guidance to the planner on actions to take, states to avoid, and so on. Together, such control knowledge has the potential to tremendously reduce the search space for a plan, an issue that is critical to planning in the real world.

We propose to incorporate such knowledge into the specification of a planning problem by replacing the final-state goal with a formula describing temporally extended goals and domain control knowledge, henceforth referred to as *control knowledge*. Our control knowledge is action-centric and procedural, in contrast to (state-centric) linear temporal logic (LTL) based control knowledge in such planners as TLPlan. We contend that it is more natural for a user that wants to specify how to construct a plan.

Even with the stipulation of control knowledge, we need look no further than travel planning to realize that many plans that are technically valid solutions, are not all equally desirable. As with control knowledge, a user may have temporally extended preferences over properties of a plan. For example, a user may prefer not to book her hotel until after her flight is booked. She may always wish to pay with a particular credit card, or use a particular airline. In addition to our procedural control knowledge, we propose a language for specifying rich, temporally extended user preferences that is unique in that it provides for both state-centric (e.g., always maintain $100 in my bank account) and action-centric (e.g., book my flight then book my hotel) preferences. In contrast to many other languages which are typically state-centric and ultimately quantitative, our language is qualitative, making it more amenable to human elicitation.

We now have a nonclassical planning problem with our final-state goal formula replaced by a specification of procedural control knowledge and user preferences. Unfortunately, most state-of-the-art planners are not designed to exploit control knowledge. A barrier to this is that much of it is temporally extended and most planners work towards achieving a goal, using some measure of progress towards goal/preference satisfaction. This is however difficult for temporally extended formulae.

A main contribution of our work is a compilation technique that can take action-centric and/or state-centric control knowledge and preference formulae, and compile them into a new planning problem that is specified in terms of final-state goals and preferences. This enables some of the fastest state-of-the-art classical planners to exploit control knowledge, and for those that use heuristic search, it provides a means of measuring progress towards satisfaction of temporally extended goals. Also problem specification including preferences can be reduced to a basic final-state goal and preference problems, which all preference-based planners address, and this again enables heuristic search.

The work presented here is part of a body of research originally presented in (Baier, Fritz, & McIlraith 2007; Bienvenu, Fritz, & McIlraith 2006; Baier & McIlraith 2007). In the sections that follow we briefly overview our specification language and compilation technique.

Specification

The specification of our planning problem comprises a domain theory, an initial state, and in place of a final-state goal we provide control knowledge in the form of a *procedure* δ, and user preferences in the form of a *preference formula* Φ. Final state goals can be expressed as a special case. In this section we intuitively describe the Golog language we use to specify δ, and the language we use to specify Φ.

Golog (Reiter 2001) has classically been used for agent programming and can be thought of in two ways: a programming language with non-deterministic constructs that are "filled-in" using planning, or a language for constraining the search space of a planner. Its syntax contains conventional programming language constructs such as if-then-else and while-loops, together with a set of nondeterministic constructions – $(\delta_1|\delta_2)$ for non-deterministic choice between sub-procedures, δ^* for non-deterministic iteration, and $\pi(x\text{-}type)\delta(x)$ for non-deterministic choice of parameter x. Returning to our travel planning example, we can specify our knowledge of how to plan a trip on the web concisely by the following Golog procedure, slightly abusing syntax and simplifying the task.

[π(*flight-Flight*) π(*pm-PaymentMethod*) *book*(*flight, pm*);
if (*IsBusinessTrip*) **then** *bookLuxuryHotel* **else**
π(*hotel-Hotel*) π(*pm-PaymentMethod*) *book*(*hotel, pm*))]

Using the sequencing construct $[a; b]$, the procedure tells us to first pick a flight non-deterministically from all available flights, choose a payment method, and book the flight. After that, if this is a business trip, we are to book a luxury hotel, or otherwise find and book a hotel. While providing some guidance as to how to plan a trip, the procedure still leaves some non-determinism. The choices are made through planning with respect to the user's preferences.

The semantics of Golog was originally defined in the situation calculus. To make such procedures usable by state-of-the-art planners, we have developed a method for compiling them to PDDL, the Planning Domain Definition Language, the input language in the International Planning Competition (IPC). The compilation is described in the next section.

In this paper we use a language for specifying rich temporal user preferences based on LTL (Bienvenu, Fritz, & McIlraith 2006) which we here refer to as $\mathcal{LPP}$. $\mathcal{LPP}$ is one possible language for expressing preferences, enabling state-centric preferences through LTL and action-centric preferences through the use of the $\mathcal{LPP}$ construct **occ**. Despite our use of $\mathcal{LPP}$ here, our technique is applicable to other preference languages, including IPC's quantitative preference language, PDDL3 (Gerevini & Long 2005). $\mathcal{LPP}$ allows the user to express temporal properties over states and actions, and to qualitatively rank such expressions to create preferences. Rankings can be complex and conditional. Finally, the language allows the user to logically combine several rankings into one general preference formula. In the first step of writing a formula, a user specifies properties over plans, for instance:

$$\textbf{always}((\forall h\text{-}Hotel, pm)(\textbf{occ}(book(h, pm)) \rightarrow h = hilton)) \quad \text{(P1)}$$
$$\textbf{always}((\forall h\text{-}Hotel, pm)(\textbf{occ}(book(h, pm)) \rightarrow h = delta)) \quad \text{(P2)}$$
$$IsBusinessTrip \rightarrow \textbf{eventually}(\textbf{occ}(fileExpenses)) \quad \text{(P3)}$$

where P1 and P2 say that a Hilton, resp. Delta, hotel is booked, and P3 states that if it is a business trip, at some point an expense report needs to be filed.

Such properties can be ranked to express preferences over them in case they turn out to be mutually exclusive in practice. The ranking does not need to be totally ordered. If the user, for instance, prefers P1 over P2, denoted P1 $\gg$ P2, but also has a second independent ranking, these can be combined using disjunction or conjunction. Rankings can also be conditioned on other properties, for instance

$$(\exists f\text{-}Flight, pm)\big(\textbf{occ}'(book(f, pm)) \wedge ArrivalTime(f) > 12am\big) :$$
$$(\forall h\text{-}Hotel, pm)\textbf{occ}'(book(h, pm)) \rightarrow NearAirport(h) \quad \text{(P4)}$$

says that if the booked flight arrives after midnight, we prefer to stay near the airport – $\textbf{occ}'(a)$ abbreviates $\textbf{eventually}(\textbf{occ}(a))$. Details of these more complex formulae and their semantics in the situation calculus can be found in (Bienvenu, Fritz, & McIlraith 2006).

Computation

Our planning problem consists of a control procedure δ and a preference formula Φ. Now we propose a compilation strategy to allow state-of-the-art planners to plan in this setting.

A standard approach to planning in the presence of control knowledge, is to build a search algorithm based on *progression*. Progression – one of the tools used in planning with temporally extended control (Bacchus & Kabanza 2000; Pistore, Bettin, & Traverso 2001) – enables the planner to prune states from the search space by determining whether or not a state visited by the search algorithm can be reached by an execution of the control procedure. The sole use of progression is not appealing for two reasons. First, it does not make procedural control available to people who wish to use procedural control but are bound to using a specific planner. Second, it does not allow the planner to exploit techniques that are central to the efficiency of state-of-the-art planners, such as domain-independent heuristic search.

Domain-independent heuristics can play a key role in efficient planning in the presence of control knowledge and preferences. By way of illustration, consider the example in the previous section. Assume the planner is about to instantiate the action of picking a flight, and furthermore suppose there is a budget limit of \$2,500. Here the planner should realize that it cannot choose an exceedingly expensive ticket, as that may lead to backtracking when later booking the hotel. In order to realize this, the planner must do some kind of lookahead computation to determine that there is an unavoidable use of money in the future. This type of computation, which can save a great deal of search effort, is standard in state-of-the-art heuristic planners (such as e.g. FF (Hoffmann & Nebel 2001)) which usually estimate the cost of achieving a goal by performing some sort of reachability

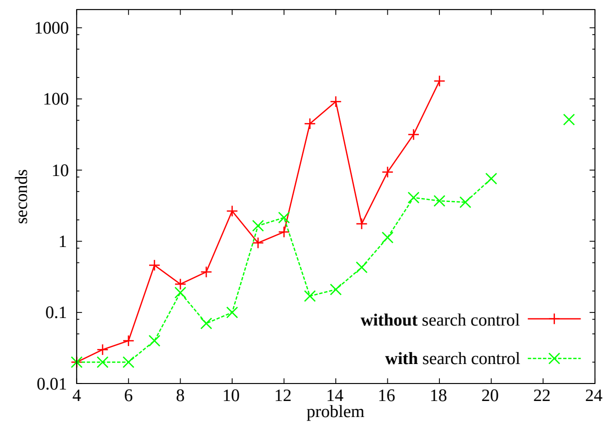

Figure 1: Run-time comparison of a heuristic search based planner solving instances of the `storage` domain of the International Planning Competition, with and without Golog search control compiled into the PDDL domain definition.

analysis. On the other hand, by using only progression, one cannot extract that type of lookahead information.

To efficiently plan in the presence of control, we have proposed a method to compile a planning instance I and a control procedure δ into a new, classical planning instance I_δ represented in PDDL, such that a plan for I_δ corresponds exactly to an execution of δ (Baier, Fritz, & McIlraith 2007). The key idea of the compilation is that a procedure δ can be represented as a finite-state automaton (whose size is polynomial in the size of δ) that in turn is represented *within* the planning domain. The state of the automaton for δ is represented by an additional predicate, and the effects and preconditions of actions in I are modified to respect the execution of the procedure by referring to those new predicates (see Fig. 2). The resulting instance is amenable to use by any state-of-the-art planner, including those exploiting heuristics, and because action preconditions are modified, search algorithms implicitly behave as if they were implementing progression. We have shown that Golog control knowledge can be effectively used to improve the efficiency of state-of-the-art planners in standard benchmark domains. Figure 1 shows a sample of these results. Full details are found in (Baier, Fritz, & McIlraith 2007).

Now that we can convert any problem with domain control into a classical planning problem, we consider the case of adding preferences. To plan efficiently for preferences we also need mechanisms to guide the search towards the satisfaction of the preferences. For example, if a preference establishes $\textbf{eventually}(\varphi)$, we want the planner to choose actions that will lead to the satisfaction of φ. By utilizing the relationship between linear temporal logic and automata, we have proposed a parametric compilation from temporal $\mathcal{LPP}$ preferences into a problem with non-temporal $\mathcal{LPP}$ preferences (Baier & McIlraith 2006; 2007). Those non-temporal preferences refer only to the final state of the plan, i.e. could be interpreted as *soft* goals. Interestingly, this enables existing state-of-the-art planning technology to be exploited to guide the search towards the

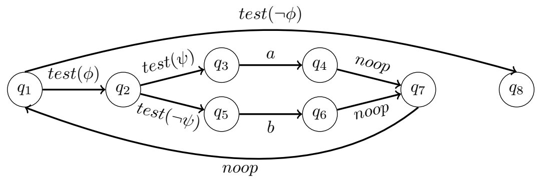

Figure 2: Automaton for $\delta =$ **while** ϕ **do if** ψ **then** a **else** b. The fluent $state$ in I_δ represents the automaton's state. In I_δ's initial state, $state = q_1$. Additional goal: $state = q_8$.

satisfaction of the preferences. Most importantly, our translation generates a problem that can be the input to almost any preference-based planner existing at the moment, since preferences only refer to the final state of the plan. We have also developed our own planner, HPLAN-QP, which is able to plan heuristically with $\mathcal{LPP}$ preferences.

Discussion

This paper summarizes and connects research published by the authors in the last two years. We have shown experimentally that our compilation techniques enable state-of-the-art planners to plan for various types of goals and preferences, typically obtaining improved performance over existing planners. Details can be found in the original papers.

The results presented here have the potential for broad applicability beyond planning. Planning can be conceived as a reachability analysis problem, as can a number of other problems in diverse areas of computer science that relate to dynamical systems. As such, the research described here is applicable to a variety of problems. Among these are controller synthesis; requirements engineering; software synthesis, particularly synthesis of component-based software such as web services; business process and workflow management; and software or hardware verification, all of which have demonstrated some use of temporally extended hard or soft constraints, to encode their problem, to control search, and/or to enforce solution quality.

We substantiate this claim with a few specific examples. Erdem & Tillier (2005) already use planning technology together with domain control knowledge to address the genome rearrangement problem. They would benefit both from the speed up provided by our compilation technique and the ability to express preferences over rearrangement alternatives. Further, Bryl, Giorgini, & Mylopoulos (2006) have considered the problem of assigning delegations of tasks to actors in the development of information systems. They have characterized this task as a planning problem with preferences. The tasks that can be assigned to the actors in the system are described procedurally, in terms of decompositions into other sub-tasks, something easily expressible in our procedural control language. Finally, de Leoni, Mecella, & de Giacomo (2007) have deployed ConGolog, a concurrent variant of Golog, to model business processes and monitor their execution. We believe our approach could help to speed up computation in these applications as well.

Despite their common heritage in the very early stages of AI, agent programming and planning have been largely studied in isolation in recent history. While the focus of agent programming has been on increasing expressiveness to address the needs of real-world applications, in classical planning the speed of plan generation has remained a central concern. Our work makes a significant step towards reuniting these two branches of research to the betterment of both. The provision of a compilation technique that enables any state-of-the-art planner to exploit control knowledge has the potential for broad impact within the planning community. Likewise agent programming applications can benefit from the opportunity to exploit state-of-the-art planners, while the integration of research on preferences to specify and generate high-quality solutions further benefits both communities.

We now turn our attention to related work, situating our contributions in the context of previous work. There is a body of related work in using domain control knowledge to speed up planning. TLPLan (Bacchus & Kabanza 2000), the winner of the 2002 International Planning Competition (IPC), is able to use state-centric temporal logic formulae to significantly prune the search space. HTN planning (Nau *et al.* 1999), is also a successful framework to incorporate procedural control. A significant difference between those approaches and ours is (1) these approaches are usually tied to specific planners and (2) planners such as TLPLan or the HTN planner SHOP2 (Nau *et al.* 2003) cannot provide guidance to achieving goals, because the algorithmic semantics given to the respectively deployed control languages is not immediately compatible with known successful heuristics.

Also related is work that compiles LTL preferences into final-state preferences (e.g. Edelkamp (2006)) and LTL goals into final-state goals (e.g. Cresswell & Coddington (2004)). These approaches, however, are not parametrized like ours and are therefore more prone to exponential blowups. Finally, Sohrabi, Prokoshyna, & McIlraith (2006) have proposed a Golog interpreter which integrates $\mathcal{LPP}$ preferences. Unlike our work, they exploit progression to compute plans.

The future prospects for this work are plentiful. A number of compelling extensions can be easily integrated into our approach. Among the most compelling is the incorporation of Golog *operators*, snippets of procedures that can be used like macro actions in the place of primitive actions during plan construction. This is very natural to do in a number of domains. Returning to our travel example, one could represent the "book hotel" action as a procedure that evaluates different prices over a number of web services eventually booking a hotel. $\mathcal{LPP}$ preferences could then refer to the execution of procedures (e.g. **occ**(*BookHotelProcedure*)) and/or impose specific user preferences on particular procedures (e.g. "I'd like to pay for the air ticket using my airmiles Visa, but the hotel with my low-interest Mastercard"). Alternatively, we could think of Golog control procedures that at some point during the execution activate a particular preference. Here, a user would declare a preference in the Golog procedure that would affect only part of it.

Although we have presented our approach as a combination of Golog action-centric control and action-centric and state-centric $\mathcal{LPP}$ user preferences, our compilation technique is sufficiently general to handle a variety of domain control specification languages and preference specification languages. For example, after compiling our control knowledge into PDDL, we could use PDDL3 preferences. In PDDL3, state-centric preference formulae are expressed in a subset of LTL. The LTL preferences can also be converted into final-state preferences in a similar way as we have done with $\mathcal{LPP}$ preferences (Baier, Bacchus, & McIlraith 2007; Edelkamp 2006). The quality of the plan, on the other hand, is expressed in a quantitative manner. The resulting planning problem could then be solved by any PDDL3-compliant planner for final-state preferences (see IPC-5 booklet for details (Gerevini *et al.* 2006)).

It is also possible to translate a subset of HTN action-centric domain control knowledge into PDDL, which can speed up planning as well (Alford, Kuter, & Nau 2009; Fritz, Baier, & McIlraith 2008). Conceptually, the result could be integrated with $\mathcal{LPP}$ or PDDL3 preferences using a similar translation. As an alternative, preferences of the kind presented here could be integrated into HTN planning directly, where the hierarchical structure can be used to express preferences not only over the occurrence of primitive actions, but also over the use of specific methods to decompose tasks into primitive actions (Sohrabi, Baier, & McIlraith 2009). This often speaks to the intuition of the user in the same way that the hierarchy helps structuring the problem conceptually.

Acknowledgements We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Ontario Research Fund Early Researcher Award.

References

Alford, R.; Kuter, U.; and Nau, D. S. 2009. Translating HTNs to PDDL: A small amount of domain knowledge can go a long way. In *Proc. of the 21th Int'l Joint Conference on Artificial Intelligence (IJCAI)*.

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116(1-2):123–191.

Baier, J. A., and McIlraith, S. A. 2006. Planning with first-order temporally extended goals using heuristic search. In *Proc. of the 21st National Conference on Artificial Intelligence*, 788–795.

Baier, J. A., and McIlraith, S. A. 2007. On domain-independent heuristics for planning with qualitative preferences. In *7th Workshop on Nonmonotonic Reasoning, Action and Change (NRAC)*.

Baier, J. A.; Bacchus, F.; and McIlraith, S. 2007. A heuristic search approach to planning with temporally extended preferences. In *Proc. of the 20th Int'l Joint Conference on Artificial Intelligence (IJCAI)*, 1805–1818.

Baier, J. A.; Fritz, C.; Bienvenu, M.; and McIlraith, S. A. 2008. Beyond classical planning: Procedural control knowledge and preferences in state-of-the-art planners. In *AAAI08*, 1509–1512.

Baier, J. A.; Fritz, C.; and McIlraith, S. A. 2007. Exploiting procedural domain control knowledge in state-of-the-art planners. In *Proc. of the 17th Int'l Conference on Automated Planning and Scheduling (ICAPS)*, 26–33.

Bienvenu, M.; Fritz, C.; and McIlraith, S. 2006. Planning with qualitative temporal preferences. In *Proc. of the 10th Int'l Conference on Knowledge Representation and Reasoning*, 134–144.

Bryl, V.; Giorgini, P.; and Mylopoulos, J. 2006. Designing cooperative IS: Exploring and evaluating alternatives. In *Proc. of the Int'l Conference on Cooperative Information Systems*, 533–550.

Cresswell, S., and Coddington, A. M. 2004. Compilation of LTL goal formulas into PDDL. In *Proc. of the 16th European Conference on Artificial Intelligence (ECAI)*, 985–986.

de Leoni, M.; Mecella, M.; and de Giacomo, G. 2007. Highly dynamic adaptation in process management systems through execution monitoring. In *Proc. of Business Process Management (BPM)*, volume 4714 of *LNCS*, 182–197.

Edelkamp, S. 2006. On the compilation of plan constraints and preferences. In *Proc. of the 16th Int'l Conference on Automated Planning and Scheduling (ICAPS)*, 374–377.

Erdem, E., and Tillier, E. R. M. 2005. Genome rearrangement and planning. In *Proc. of the 20th National Conference on Artificial Intelligence (AAAI)*, 1139–1144.

Fritz, C.; Baier, J. A.; and McIlraith, S. A. 2008. ConGolog, sin trans: Compiling ConGolog into basic action theories for planning and beyond. In *Proc. on the 11th Int'l Conference on Principles of Knowledge Representation and Reasoning*, 600–610.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences for PDDL3. Technical Report 2005-08-07, Department of Electronics for Automation, Univ. of Brescia, Brescia, Italy.

Gerevini, A.; Dimopoulos, Y.; Haslum, P.; and Saetti, A. 2006. 5th International Planning Competition booklet. `http://zeus.ing.unibs.it/ipc-5/`.

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

Nau, D. S.; Cao, Y.; Lotem, A.; and Muñoz-Avila, H. 1999. SHOP: Simple hierarchical ordered planner. In *Proc. of the 16th Int'l Joint Conference on Artificial Intelligence (IJCAI)*, 968–975.

Nau, D. S.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research* 20:379–404.

Pistore, M.; Bettin, R.; and Traverso, P. 2001. Symbolic techniques for planning with extended goals in non-deterministic domains. In *Proc. of the 6th European Conference on Planning*.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. Cambridge, MA: MIT Press.

Sohrabi, S.; Baier, J. A.; and McIlraith, S. A. 2009. HTN planning with preferences. In *Proc. of the 21th Int'l Joint Conference on Artificial Intelligence (IJCAI)*, 1790 –1797. This paper also appears in Commonsense 2009.

Sohrabi, S.; Prokoshyna, N.; and McIlraith, S. A. 2006. Web service composition via generic procedures and customizing user preferences. In *Proc. of the 5th International Semantic Web Conference (ISWC)*, 597–611.