# On Planning with Preferences in HTN*

**Shirin Sohrabi** and **Sheila A. McIlraith**
Department of Computer Science
University of Toronto
Toronto, Canada.
{shirin,sheila}@cs.toronto.edu

**Abstract**

In this paper, we address the problem of generating preferred plans by combining the procedural control knowledge specified by Hierarchical Task Networks (HTNs) with rich qualitative user preferences. The outcome of our work is a language for specifying user preferences, tailored to HTN planning, together with a provably optimal preference-based planner, **HTNPLAN**, that is implemented as an extension of **SHOP2**. To compute preferred plans, we propose an approach based on forward-chaining heuristic search. Our heuristic uses an admissible evaluation function measuring the satisfaction of preferences over partial plans. Our empirical evaluation demonstrates the effectiveness of our **HTNPLAN** heuristics. We prove our approach sound and optimal with respect to the plans it generates by appealing to a situation calculus semantics of our preference language and of HTN planning. While our implementation builds on **SHOP2**, the language and techniques proposed here are relevant to a broad range of HTN planners.

## 1 Introduction

Hierarchical Task Network (HTN) planning is a popular and widely used planning paradigm, and many domain-independent HTN planners exist (e.g., **SHOP2**, SIPE-2, I-X/I-PLAN, O-PLAN) (Ghallab, Nau, and Traverso 2004). In HTN planning, the planner is provided with a set of tasks to be performed, possibly together with constraints on those tasks. A plan is then formulated by repeatedly decomposing tasks into smaller and smaller subtasks until primitive, executable tasks are reached. A primary reason behind HTN's success is that its task networks capture useful procedural control knowledge—advice on how to perform a task—described in terms of a decomposition of subtasks. Such control knowledge can significantly reduce the search space for a plan while also ensuring that plans follow one of the stipulated courses of action. However, while HTNs specify a family of satisfactory plans, they are, for the most part, unable to distinguish high-quality plans.

In this paper, we address the problem of generating preferred plans by augmenting HTN planning problems with rich qualitative user preferences. User preferences can be arbitrarily complex, often involving combinations of conditional, interacting, and mutually exclusive preferences that can range over multiple states of a plan. This makes finding an optimal plan hard. There are two aspects to addressing the problem of preference-based planning with HTNs. The first is to propose a preference specification language that is tailored to HTN planning. The second, is to generate preferred, and ideally optimal, plans efficiently.

To specify user preferences, we augment a rich qualitative preference language, $\mathcal{LPP}$, proposed in (Bienvenu, Fritz, and McIlraith 2006) with HTN-specific constructs. $\mathcal{LPP}$ specifies preferences in a variant of linear temporal logic (LTL). Among the HTN-specific properties that we add to our language, $\mathcal{LPH}$, is the ability to express preferences over how tasks in our HTN are decomposed into subtasks, preferences over the parameterizations of decomposed tasks, and a variety of temporal and nontemporal preferences over the task networks themselves.

To compute preferred plans, we propose an approach based on forward-chaining heuristic search. Key to our approach is a means of evaluating the (partial) satisfaction of preferences during HTN plan generation based on progression. The optimistic evaluation of preferences yields an admissible evaluation function which we use to guide search. We implemented our planner, **HTNPLAN**, as an extension to the **SHOP2** HTN planner. Our empirical evaluation demonstrates the effectiveness of **HTNPLAN** heuristics in finding high-quality plans. We provide a semantics for our preference language in the situation calculus (Reiter 2001) and appeal to this semantics to prove the soundness and optimality of our planner with respect to the plans it generates.

In Section 2, we review HTN planning, situation calculus, Golog, ConGolog, and provide an encoding of a preference-based HTN planning problem. In Section 3, we provide the syntax and the semantics for our preference language. In Section 4, we turn our attention to computing preferred plans describing how we evaluate the satisfaction of preferences over partial plans using progression. In Section 5, we describe the implementation of our HTN preference-based planner, **HTNPLAN** that is built on top of **SHOP2**, and provide empirical results that establish the effectiveness of our evaluation function in guiding search. We conclude with a summary and discussion of related work.

*A shorter version of this paper appears in the Proceedings of the Fourth Multidisciplinary Workshop on Advances in Preference Handling (MPref 2008).

## 2 HTN Planning

In this section, we provide a brief overview of both HTN planning, following (Ghallab, Nau, and Traverso 2004), and our situation calculus encoding of preference-based HTN planning.

**Travel Example:** Consider a simple HTN planning problem to address the task of arranging travel. This task can be decomposed into arranging transportation, accommodations, and local transportation. Each of these tasks can again be decomposed based on alternative modes of transportation and accommodations, reducing eventually to primitive actions that can be executed in the world. Further constraints can be imposed to restrict decompositions.

**Definition 1 (HTN Planning Problem)** *An HTN planning problem is a 3-tuple $\mathcal{P} = (s_0, w, D)$ where $s_0$ is the initial state, $w$ is a task network called the initial task network, and $D$ is the HTN planning domain. $\mathcal{P}$ is a total-order planning problem if w and $D$ are totally ordered; otherwise it is said to be partially ordered.*

A *task* consists of a task symbol and a list of arguments. A task is primitive if its task symbol is an operator name and its parameters match, otherwise it is *nonprimitive*. In our example, *arrange-trans* and *arrange-acc* are nonprimitive tasks, while *book-flight* and *book-car* are primitive tasks.

**Definition 2 (Task Network)** *A task network is a pair w=(U, C) where U is a set of task nodes and C is a set of constraints. Each task node u ∈ U contains a task $t_u$. If all of the tasks are ground then w is ground; If all of the tasks are primitive, then w is called primitive; otherwise is called nonprimitive. Task network $w$ is totally ordered if $C$ defines a total ordering of the nodes in U.*

In our example, we could have a task network $(U, C)$ where $U = \{u_1, u_2\}$, $u_1$ =*book-car*, and $u_2$= *pay*, and $C$ is a precedence constraint such that $u_1$ must occur before $u_2$ and a before-constraint such that at least one car is available for rent before $u_1$.

A domain is a pair $D = (O, M)$ where $O$ is a set of operators and $M$ is a set of methods. Operators are essentially primitive actions that can be executed in the world. They are described by a triple *o =(name(o), pre(o), eff(o))*, corresponding to the operator's name, preconditions and effects. Preconditions are restricted to a set of literals, and effects are described as STRIPS-like Add and Delete lists. An operator $o$ can accomplish a ground primitive task in a state $s$ if their names match and $o$ is applicable in $s$. In our example, ignoring the parameters, operators might include: *pay, book-train, book-car, book-hotel,* and *book-flight*.

A method, $m$, is a 4-tuple (*name(m), task(m),subtasks(m), constr(m))* corresponding to the method's name, a nonprimitive task and the method's task network, comprising subtasks and constraints. A method is *totally ordered* if its task network is *totally ordered*. A domain is a total-order domain if every $m \in M$ is *totally ordered*. Method $m$ is relevant for a task $t$ if there is a substitution $\sigma$ such that $\sigma(t)$ =*task*$(m)$. Several different methods can be relevant to a particular nonprimitive task $t$, leading to different decompositions of $t$. In our example, the method with *name by-flight-trans* can be used to decompose the *task arrange-trans* into the *subtasks* of booking a flight and paying, with the constraint (*constr*) that the booking precede payment.

**Definition 3 (Solution to HTN Planning Problem)** *Given HTN planning problem $\mathcal{P} = (s_0, w, D)$, a plan $\pi = (o_1, ..., o_k)$ is a solution for $\mathcal{P}$, depending on these two cases: 1) if w is primitive, then there must exist a ground instance (U′, C′) of (U, C) and a total ordering $(u_1, ..., u_k)$ of the nodes in $U'$ such that for all $1 \leq i \leq k$, name($o_i$) = $t_{u_i}$, the plan $\pi$ is executable in the state $s_0$, and all the constraints hold, 2) if w is nonprimitive, then there must exist a sequence of task decompositions that can be applied to w to produce a primitive task network $w'$, where $\pi$ is a solution for $w'$.*

Finally, we define the HTN preference-based planning problem. This definition appeals to two concepts that are not yet well-defined and which we defer to later sections: definitions of the form and content of the the formula $\Phi_{htn}$ that captures user preferences for HTN planning as well as and the precise definition of *more preferred* appears in Section 3.

**Definition 4 (Preference-based HTN Planning)** *An HTN planning problem with user preferences is described as a 4-tuple $\mathcal{P} = (s_0, w, D, \Phi_{htn})$ where $\Phi_{htn}$ is a formula describing user preferences. A plan $\pi$ is a solution to $\mathcal{P}$ if and only if: $\pi$ is a plan for $\mathcal{P}' = (s_0, w, D)$ and there does not exists a plan $\pi'$ such that $\pi'$ is more preferred than $\pi$ with respect to the preference formula $\Phi_{htn}$.*

### 2.1 Situation Calculus Specification of HTN

We now have a definition of preference-based HTN planning. Later in the paper, we propose an approach to computing preferred plans, together with a description of our implementation. To prove the correctness and optimality of our algorithm, we appeal to an existing situation calculus encoding of HTN planning, which we augment and extend to provide an encoding of preference-based HTN planning. Since the situation calculus has a well-defined semantics, we have a semantics for our encoding which we use in our proofs. In this section, we review the salient features of this encoding.

**The Situation Calculus** is a logical language for specifying and reasoning about dynamical systems (Reiter 2001). In the situation calculus, the *state* of the world is expressed in terms of functions and relations (fluents) relativized to a particular *situation* $s$, e.g., $F(\vec{x}, s)$. A situation $s$ is a *history* of the primitive actions, $a \in \mathcal{A}$, performed from a distinguished initial situation $S_0$. The function $do(a, s)$ maps a situation and an action into a new situation thus inducing a tree of situations rooted in $S_0$. A *basic action theory* in the situation calculus $\mathcal{D}$ includes *domain independent foundational axioms*, and *domain dependent axioms*. A situation $s'$ precedes a situation $s$, i.e., $s' \sqsubset s$, means that the sequence $s'$ is a proper prefix of sequence $s$.

**Golog** (Reiter 2001) is a high-level logic programming language for the specification and execution of complex actions in dynamical domains. It builds on top of the situation

calculus by providing Algol-inspired extralogical constructs for assembling primitive situation calculus actions into complex actions (*programs*) $\delta$. Example complex actions include action sequences, if-then-else, while loops, nondeterministic choice of actions and action arguments, and procedures. These complex actions serve as constraints upon the situation tree. ConGolog (De Giacomo, Lespérance, and Levesque 2000) is the concurrent version of Golog in which the language can additionally deal with execution of concurrent processes, interrupts, prioritized concurrency, and exogenous actions.

A number of researchers have pointed out the connection between HTN and ConGolog. Following Gabaldon (Gabaldon 2002), we map an HTN state to a situation calculus *situation*. Consequently, the initial HTN state $s_0$ is encoded as the initial situation, $S_0$. The HTN domain description maps to a corresponding situation calculus domain description, $\mathcal{D}$, where for every operator $o$ there is a corresponding primitive action $a$, such that the preconditions and the effects of $o$ are axiomatized in $\mathcal{D}$. Every method and nonprimitive task together with constraints is encoded as a ConGolog procedure. For the purposes of this paper, the set of procedures in a ConGolog domain theory is referred to as $\mathcal{R}$.

We use a predicate *badSituation*$(s)$ proposed by Reiter (Reiter 2001) to encode the constraints in a task network. The purpose of these constraints is to prune part of a search space similar to using temporal constraints.

To deal with partially ordered task networks, we add two new primitive actions *start*$(P(\vec{v}))$, *end*$(P(\vec{v}))$, and two new fluents *executing*$(P(\vec{v}), s)$ and *terminated*$(X, s)$, where $P(\vec{v})$ is a ConGolog procedure and $X$ is either $P(\vec{v})$ or an action $a \in \mathcal{A}$. *executing*$(P(\vec{v}), s)$ states that $P(\vec{v})$ is executing in situation $s$, *terminated*$(X, s)$ states that $X$ has terminated in $s$. *executing*$(a, s)$ where $a \in \mathcal{A}$ is defined to be false. The successor state axioms for these fluents follow. They show how the actions *start*$(P(\vec{v}))$, *end*$(P(\vec{v}))$ change the truth value of these fluents:

$$\begin{aligned} &\textit{executing}(P(\vec{v}), do(a, s)) \equiv a = \textit{start}(P(\vec{v})) \vee \\ &\quad \textit{executing}(P(\vec{v}), s) \wedge a \neq \textit{end}(P(\vec{v})) \\ &terminated(X, do(a, s)) \equiv X = a \vee \\ &\quad (X \in \mathcal{R} \wedge a = \textit{end}(X)) \vee \textit{terminated}(X, s) \end{aligned}$$

where $\mathcal{R}$ is the set of ConGolog procedures in our domain.

**Definition 5 (Preference-based HTN in Situation Calculus)** *An HTN planning problem with user preferences described as a 4-tuple $\mathcal{P} = (s_0, w, D, \Phi_{htn})$ is encoded in situation calculus as a 5-tuple $(\mathcal{D}, \mathcal{C}, \Delta, \delta_0, \Phi_{sc})$ where $\mathcal{D}$ is the basic action theory, $\mathcal{C}$ is the set of ConGolog axioms,$\Delta$ is the sequence of procedure declarations for all ConGolog procedures in $\mathcal{R}$, $\delta_0$ is an encoding of the initial task network in ConGolog, and $\Phi_{sc}$ is a mapping of the preference formula $\Phi_{htn}$ in situation calculus. A plan $\vec{a}$ is a solution to the encoded preference-based HTN problem if and only if:*

$$\begin{aligned} \mathcal{D} \cup \mathcal{C} \models\ & (\exists s) Do(\Delta; \delta_0, S_0, s) \wedge s = do(\vec{a}, S_0) \\ & \wedge \neg \textit{badSituation}(s) \wedge \nexists s'.[Do(\Delta; \delta_0, S_0, s') \\ & \wedge \neg \textit{badSituation}(s') \wedge pref(s', s, \Phi_{sc})] \end{aligned}$$

where $pref(s', s, \Phi_{sc})$ denotes that the situation $s'$ is preferred to situation s with respect to the preference formula $\Phi_{sc}$, and $Do(\delta, S_0, do(\vec{a}, S_0))$ denotes that the ConGolog program $\delta$, starting execution in $S_0$ will legally terminate in situation $do(\vec{a}, S_0)$. Removing all the *start*$(P(\vec{v}))$ and $end(P(\vec{v}))$ actions from $\vec{a}$ to obtain $\vec{b} = (b_1, ..., b_n)$, a preferred plan for the original HTN planning problem $\mathcal{P}$ is a plan $\pi = (o_1, ..., o_n)$ where for all $1 \leq i \leq n$, name($o_i$)= $b_i$.

## 3 HTN Preference Specification

In this section, we describe how to specify the preference formula $\Phi_{htn}$. Our preference language, $\mathcal{LPH}$, modifies and extends the $\mathcal{LPP}$ qualitative preference language proposed in (Bienvenu, Fritz, and McIlraith 2006) to capture HTN-specific preferences.

Our $\mathcal{LPH}$ language has the ability to express preferences over certain parameterization of a task (e.g., preferring one task grounding to another), over a certain decomposition of nonprimitive tasks (i.e., prefer to apply a certain method over another), and a soft version of the before, after, and in between constraints. A soft constraint is defined via a preference formula whose evaluation determines when a plan is *more preferred* than another. However, unlike the task network constraints which will prune or eliminate those plans that have not satisfied them, not meeting a soft constraint simplify deems a plan to be of poorer quality.

**Definition 6 (Basic Desire Formula (BDF))** *A basic desire formula is a sentence drawn from the smallest set $\mathcal{B}$ where:*

1. *If $l$ is a literal, then $l \in \mathcal{B}$ and **final**$(l) \in \mathcal{B}$*
2. *If $t$ is a task, then **occ**$(t) \in \mathcal{B}$*
3. *If $m$ is a method, and $n = name(m)$, then **apply**$(n) \in \mathcal{B}$*
4. *If $t_1$, and $t_2$ are tasks, and $l$ is a literal, then **before**$(t_1, t_2)$, **holdBefore**$(t_1, l)$, **holdAfter**$(t_1, l)$, **holdBetween**$(t_1, l, t_2)$ are in $\mathcal{B}$.*
5. *If $\varphi_1$ and $\varphi_2$ are in $\mathcal{B}$, then so are $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$,$(\exists x)\varphi_1$, $(\forall x)\varphi_1$, **next**$(\varphi_1)$, **always**$(\varphi_1)$, **eventually**$(\varphi_1)$, and **until**$(\varphi_1, \varphi_2)$.*

**final**$(l)$ states that the literal $l$ holds in the final state, **occ**$(t)$ states that the task $t$ occurs in the present state, and **next**$(\varphi_1)$, **always**$(\varphi_1)$, **eventually**$(\varphi_1)$, and **until**$(\varphi_1, \varphi_2)$ are basic LTL constructs. **apply**$(n)$ states that a method whose name is $n$ is applied to decompose a nonprimitive task. **before**$(t_1, t_2)$ states a precedence ordering between two tasks. **holdBefore**$(t_1, l)$, **holdAfter**$(t_1, l)$, **holdBetween**$(t_1, l, t_2)$ state a soft constraint over when the fluent $l$ is preferred to hold. (i.e., **holdBefore**$(t_1, l)$ state that $l$ must be true right before the last operator descender of $t_1$ occurs). Combining **occ**$(t)$ with the rest of $\mathcal{LPH}$ language enables the construction of preference statements over parameterizations of tasks.

BDFs establish properties of different states within a plan. By combining BDFs using boolean and temporal connectives, we are able to express other properties of state. The

following are a few examples from our travel domain[1].

$$(\exists c).\textbf{occ}'(\textit{book-car}(c, Enterprise)) \quad \text{(P1)}$$
$$\textbf{apply}'(\textit{by-car-local(SUV, Avis)}) \quad \text{(P2)}$$
$$\textbf{before}(\textit{arrange-trans}, \textit{arrange-acc}) \quad \text{(P3)}$$
$$\textbf{holdBefore}(hotelReservation, \textit{arrange-trans}) \quad \text{(P4)}$$
$$\textbf{always}(\neg(\textbf{occ}'(\textit{pay}(Mastercard)))) \quad \text{(P5)}$$
$$(\exists h, r).\textbf{occ}'(\textit{book-hotel}(h, r)) \wedge \textit{starsGE}(r, 3) \quad \text{(P6)}$$
$$(\exists c).\textbf{occ}'(\textit{book-flight}(c, \textit{Economy}, \textit{Direct}, \textit{WindowSeat})) \wedge \textit{member}(c, \textit{StarAlliance}) \quad \text{(P7)}$$

P1 states that at some point the user books a car with Enterprise. P2 states that at some point, the *by-car-local* method is applied to book an SUV from Avis. P3 states that the *arrange-trans* task occurs before the *arrange-acc* task. P4 states that the hotel is reserved before transportation is arranged. P5 states that the user never pays by Mastercard. P6 states that at some point the user books a hotel that has a rating of 3 or more. P7 states that at some point the user books a direct economy window-seated flight with a Star Alliance carrier.

To define a preference ordering over alternative properties of states, *Atomic Preference Formulae* (APFs) are defined. Each alternative comprises two components: the property of the state, specified by a BDF, and a *value* term which stipulates the relative strength of the preference.

**Definition 7 (Atomic Preference Formula (APF))**
*Let $\mathcal{V}$ be a totally ordered set with minimal element $v_{min}$ and maximal element $v_{max}$. An atomic preference formula is a formula $\varphi_0[v_0] \gg \varphi_1[v_1] \gg ... \gg \varphi_n[v_n]$, where each $\varphi_i$ is a BDF, each $v_i \in \mathcal{V}$, $v_i < v_j$ for $i < j$, and $v_0 = v_{min}$. When $n = 0$, atomic preference formulae correspond to BDFs.*

While one could let $\mathcal{V} = [0, 1]$, you could choose a strictly qualitative set like $\{$*best* $<$ *good* $<$ *indifferent* $<$ *bad* $<$ *worst*$\}$ to express preferences over alternatives.

Now here are a few APF examples from the travel domain.

$$P2[0] \gg \textbf{apply}'(\textit{by-car-local(SUV, National)})[0.3] \quad \text{(P8)}$$
$$\textbf{apply}'(\textit{by-car-trans})[0] \gg \textbf{apply}'(\textit{by-flight})[0.4] \quad \text{(P9)}$$
$$\textbf{occ}'(\textit{book-train})[0] \gg \textbf{occ}'(\textit{book-car})[0.4] \quad \text{(P10)}$$

P8 states that the user prefers that the *by-car-local* method rents an SUV and that the rental car company Avis is preferred to National. P9 states that the user prefers to decompose the *arrange-trans* task by the method *by-car-trans* rather than the *by-flight* method. Note that the task is implicit in the definition of the method. P10 states that the user prefers travelling by train over renting a car.

To allow the user to specify more complex preferences and to aggregate preferences, General Preference Formulae (GPFs) extend the language to conditional, conjunctive, and disjunctive preferences.

[1]To simplify the examples many parameters have been suppressed, and we abbreviate **eventually**(**occ**($\varphi$)) by **occ**$'$, **eventually**(**apply**($\varphi$)) by **apply**$'$ and refer to preferences by their labels.

**Definition 8 (General Preference Formula (GPF))**
*A formula $\Phi$ is a GPF if one of the following holds:*
- *$\Phi$ is an APF*
- *$\Phi$ is $\gamma : \Psi$, where $\gamma$ is a BDF and $\Psi$ is a GPF [Conditional]*
- *$\Phi$ is one of $\Psi_0 \,\&\, \Psi_1 \,\&\, ... \,\&\, \Psi_n$ [General Conjunction] or $\Psi_0 \mid \Psi_1 \mid ... \mid \Psi_n$ [General Disjunction]*

*where $n \geq 1$ and each $\Psi_i$ is a GPF.*

General conjunction (resp.general disjunction) refines the ordering defined by $\Psi_0 \,\&\, \Psi_1 \,\&\, ... \,\&\, \Psi_n$ (resp. $\Psi_0|\Psi_1|...|\Psi_n$) by sorting indistinguishable states using the lexicograping ordering. Continuing our example:

$$\textbf{occ}(\textit{arrange-trans}) : (\exists c).\textbf{occ}'(\textit{book-car}(c, Avis)) \quad \text{(P11)}$$
$$\textbf{occ}(\textit{arrange-local-trans}) : P1 \quad \text{(P12)}$$
$$drivable : P10[0] \gg \textbf{occ}'(\textit{book-flight})[0.3] \quad \text{(P13)}$$
$$P4 \,\&\, P6 \,\&\, P7 \,\&\, P8 \,\&\, P9 \,\&\, P10 \,\&\, P12 \,\&\, P13 \quad \text{(P14)}$$

P11 states that if inter-city transportation is being arranged then the user prefers to rent a car from Avis. P12 states that if local transportation is being arranged the user prefers Enterprise. P13 states that if the distance between the origin and the destination is drivable then the user prefers to book a train over booking a car over booking a flight. P14 aggregates preferences into one formula.

Again, and only for the purpose of proving properties, we provide an encoding of the HTN-specific terms of $\mathcal{LPH}$ in the situation calculus. As such, for any preference formula $\Phi_{htn}$ there is a corresponding formula $\Phi_{sc}$ where every HTN-specific term is replaced as follows: each literal $l$ is mapped to a fluent or non-fluent relation in the situation calculus, as appropriate; each primitive task $t$ is mapped to an action $a \in \mathcal{A}$; and each nonprimitive task $t$ and each method $m$ is mapped to a procedure $P(\vec{v}) \in \mathcal{R}$ in ConGolog.

## 3.1 The Semantics

The semantics of $\mathcal{LPH}$ is achieved through assigning a weight to a situation $s$ with respect to a GPF, $\Phi$, written $w_s(\Phi)$. This weight is a composition of its constituents. For BDFs, a situation $s$ is assigned the value $v_{min}$ if the BDF is satisfied in $s$, $v_{max}$ otherwise. Similarly, given an APF, and a situation $s$, $s$ is assigned the weight of the best BDF that it satisfies within the defined APF. Finally GPF semantics follow the natural semantics of boolean connectives. As such General Conjunction yields the minimum of its constituent GPF weights and General Disjunction yields the maximum.

Similar to (Gabaldon 2004) and following $\mathcal{LPP}$, we use the notation $\varphi[s', s]$ to denote that $\varphi$ holds in the sequence of situations starting from $s'$ and terminating in $s$. Next, we will show how to interpret BDFs in the situation calculus.

If $f$ is a fluent, we will write $f[s', s] = f[s']$ since fluents are represented in situation-suppressed form. If $r$ is a non-fluent, we will have $r[s', s] = r$ since $r$ is already a situation calculus formula. Furthermore, we will write $\textbf{final}(f)[s', s] = f[s]$ since $\textbf{final}(f)$ means that the fluent $f$ must hold in the final situation.

The BDF $\textbf{occ}(X)$ states the occurrence of $X$ which can be either an action or a procedure. written as:

$$\textbf{occ}(X)[s', s] = \begin{cases} do(X, s') \sqsubseteq s & \text{if X} \in \mathcal{A} \\ do(start(X), s') \sqsubseteq s & \text{if X} \in \mathcal{R} \end{cases}$$

The BDF **apply**$(P(\vec{v}))$ will be interpreted as follows:

$$\textbf{apply}(P(\vec{v}))[s', s] = do(start(P(\vec{v})), s') \sqsubseteq s$$

Boolean connectives and quantifiers are already part of the situation calculus and require no further explanation here. The LTL constructs are interpreted in the same way as in (Gabaldon 2004). We interpret the rest of the connectives as follows [2].

$$\textbf{before}(X_1, X_2)[s', s] = (\exists s_1, s_2 : s' \sqsubseteq s_1 \sqsubseteq s_2 \sqsubseteq s) \{\textit{terminated}(X_1)[s_1] \wedge \neg \textit{executing}(X_2)[s_1] \wedge \neg \textit{terminated}(X_2)[s_1] \wedge \textbf{occ}(X_2)[s_2, s]\}$$

$$\textbf{holdBefore}(X, f)[s', s] = (\exists s_1 : s' \sqsubseteq s_1 \sqsubseteq s) \{f[s_1] \wedge \textbf{occ}(X)[s_1, s]\}$$

$$\textbf{holdAfter}(X, f)[s', s] = (\exists s_1 : s' \sqsubseteq s_1 \sqsubseteq s) \{terminated(X)[s_1] \wedge f[s_1]\}$$

$$\textbf{holdBetween}(X_1, f, X_2)[s', s] = (\exists s_1, s_2 : s' \sqsubseteq s_1 \sqsubseteq s_2 \sqsubseteq s) \{\textit{terminated}(X_1)[s_1] \wedge \neg \textit{executing}(X_2)[s_1] \wedge \neg \textit{terminated}(X_2)[s_1] \wedge \textbf{occ}(X_2)[s_2, s]\} \wedge (\forall s_i : s_1 \sqsubseteq s_i \sqsubseteq s_2) f[s_i]$$

From here, the semantics follows that of $\mathcal{LPP}$.

**Definition 9 (Basic Desire Satisfaction)** *Let $\mathcal{D}$ be an action theory, and let $s'$ and $s$ be situations such that $s' \sqsubseteq s$. The situations beginning in $s'$ and terminating in $s$ satisfy $\varphi$ just in the case that $\mathcal{D} \models \varphi[s', s]$. We define $w_{s',s}(\varphi)$ to be the weight of the situations originating in $s'$ and ending in $s$ wrt BDF $\varphi$. $w_{s',s}(\varphi) = v_{min}$ if $\varphi$ is satisfied, otherwise $w_{s',s}(\varphi) = v_{max}$.*

Note that for readability we are going to drop $s'$ from the index, i.e., $w_s(\varphi) = w_{s',s}(\varphi)$ in the special case of $s' = S_0$.

**Definition 10 (Atomic Preference Satisfaction)** *Let $s$ be a situation and $\Phi = \varphi_0[v_0] \gg \varphi_1[v_1] \gg ... \gg \varphi_n[v_n]$ be an atomic preference formula. Then $w_s(\Phi) = v_i$ if $i = min_j\{\mathcal{D} \models \varphi_j[S_0, s]\}$, and $w_s(\Phi) = v_{max}$ if no such $i$ exists.*

**Definition 11 (General Preference Satisfaction)** *Let $s$ be a situation and $\Phi$ be a general preference formula. Then $w_s(\Phi)$ is defined as follows:*

- $w_s(\varphi_0 \gg \varphi_1 \gg ... \gg \varphi_n)$ *is defined above*
- $w_s(\gamma : \Psi) = \begin{cases} v_{min} & \text{if } w_s(\gamma) = v_{max} \\ w_s(\Psi) & \text{otherwise} \end{cases}$
- $w_s(\Psi_0 \,\&\, \Psi_1 \,\&\, ... \,\&\, \Psi_n) = \max\{w_s(\Psi_i) : 1 \leq i \leq n\}$
- $w_s(\Psi_0 \mid \Psi_1 \mid ... \mid \Psi_n) = min\,\{w_s(\Psi_i) : 1 \leq i \leq n\}$

The following definition dictates how to compare two situations (and thus two plans) with respect to a GPF. This preference relation $pref$ is used to compare HTN plans in Definition 5 and provides the semantics for *more preferred* in Definition 4.

**Definition 12 (Preferred Situations)** *A situation $s_1$ is at least as preferred as a situation $s_2$ with respect to a GPF $\Phi$, written $pref(s_1, s_2, \Phi)$ if $w_{s_1}(\Phi) \leq w_{s_2}(\Phi)$.*

[2] We use the following abbreviations:
$(\exists s_1 : s' \sqsubseteq s_1 \sqsubseteq s)\Phi = (\exists s_1)\{s' \sqsubseteq s_1 \wedge s_1 \sqsubseteq s \wedge \Phi\}$
$(\forall s_1 : s' \sqsubseteq s_1 \sqsubseteq s)\Phi = (\forall s_1)\{[s' \sqsubseteq s_1 \wedge s_1 \sqsubseteq s] \subset \Phi\}$

# 4 Computing Preferred Plan

To compute a preferred plan, we proposed a heuristic-search, forwarding-chaining planner that searchs for the *most preferred* terminating state that satisfies the HTN planning problem. The search is guided by an admissible evaluation function that evaluates partial plans with respect to preference satisfaction. We use *progression* to evaluate the preference formula satisfaction over partial plans.

## 4.1 Progression

Given a situation and a temporal formula, progression evaluates it with respect to the state of a situation to generate a new formula representing those aspects of the formula that remain to be satisfied. In this section, we define the progression of the constructs we added/modified from $\mathcal{LPP}$ and show that progression preserves the semantics of preference formulae. To define the progression, similar to (Bienvenu, Fritz, and McIlraith 2006) we add the propositional constants TRUE and FALSE to both the situation calculus and to our set of BDFs, where $\mathcal{D} \models$ TRUE and $\mathcal{D} \not\models$ FALSE for every action theory $\mathcal{D}$. We also add the BDF **occNext**$(X)$, and **applyNext**$(P(\vec{v}))$ to capture the progression of **occ**$(X)$ and **apply**$(P(\vec{v}))$. Below we show the progression of the added constructs.

**Definition 13 (Progression)** *Let $s$ be a situation, and let $\varphi$ be a BDF. The progression of $\varphi$ through $s$, written $\rho_s(\varphi)$, is given by:*

- *If $\varphi$=**occ**$(X)$ then*
  $\rho_s(\varphi)$ = ***occNext***$(X) \wedge$***eventually***$(terminated(X))$
- *If $\varphi =$ **occNext**$(X)$ , then*
  $\begin{cases} \textit{TRUE} & \textit{if } X \in \mathcal{A} \wedge \mathcal{D} \models \exists s'.s = do(X, s') \\ \textit{TRUE} & \textit{if } X \in \mathcal{R} \wedge \mathcal{D} \models \exists s'.s = do(start(X), s') \\ \textit{FALSE} & \textit{otherwise} \end{cases}$
- *If $\varphi =$ **apply**$(P(\vec{v}))$, then*
  $\rho_s(\varphi) =$ ***applyNext***$(P(\vec{v})) \wedge$***eventually***$(terminated(P(\vec{v})))$
- *If $\varphi =$ **applyNext**$(P(\vec{v}))$ , then*
  $\rho_s(\varphi) = \begin{cases} \textit{TRUE} & \textit{if } \mathcal{D} \models \exists s'.s = do(start(P(\vec{v})), s') \\ \textit{FALSE} & \textit{otherwise} \end{cases}$
- *If $\varphi =$ **before**$(X_1, X_2)$, **holdBefore**$(X, f)$, **holdAfter**$(X, f)$, or **holdBetween**$(X_1, f, X2)$, then*
  $\rho_s(\varphi) = \begin{cases} \textit{TRUE} & \textit{if } w_s(\varphi) = v_{min} \\ \textit{FALSE} & \textit{otherwise} \end{cases}$

To see how the other constructs are progressed please refer to (Bienvenu, Fritz, and McIlraith 2006).

## 4.2 Admissible Evaluation Function

In this section, we describe an admissible evaluation function using the notion of *optimistic* and *pessimistic* weights that provide a bound on the best and worst weights of any successor situation with respect to a GPF $\Phi$. Optimistic (resp. pessimistic) weights, $w_s^{opt}(\Phi)$ (resp. $w_s^{pess}(\Phi)$) are defined based on optimistic (resp. pessimistic) satisfaction of BDFs. Optimistic satisfaction ($\varphi[s', s]^{opt}$) assumes that any parts of the BDF not yet falsified will eventually be satisfied. Pessimistic satisfaction ($\varphi[s', s]^{pess}$) assumes the opposite. The following definitions highlight the key differences between this work and the definitions in (Bienvenu, Fritz, and McIlraith 2006).

$$\textbf{occ}(X)[s', s]^{opt} \stackrel{\text{def}}{=} \begin{cases} do(X, s') \sqsubseteq s \vee s' = s & \text{if X} \in \mathcal{A} \\ do(start(X), s') \sqsubseteq s \vee s' = s & \text{if X} \in \mathcal{R} \end{cases}$$

$$\textbf{occ}(X)[s', s]^{pess} \stackrel{\text{def}}{=} \begin{cases} do(X, s') \sqsubseteq s & \text{if X} \in \mathcal{A} \\ do(start(X), s') \sqsubseteq s & \text{if X} \in \mathcal{R} \end{cases}$$

$$\textbf{apply}(P(\vec{v}))[s', s]^{opt} \stackrel{\text{def}}{=} do(start(P(\vec{v})), s') \sqsubseteq s \vee s' = s$$

$$\textbf{apply}(P(\vec{v}))[s', s]^{pess} \stackrel{\text{def}}{=} do(start(P(\vec{v})), s') \sqsubseteq s$$

If $\varphi =$ **before**$(X_1, X_2)$, **holdBefore**$(X, f)$, **holdAfter**$(X, f)$ **holdBetween**$(X_1, f, X_2)$, then

$$\varphi[s', s]^{opt} \stackrel{\text{def}}{=} \varphi[s', s]^{pess} \stackrel{\text{def}}{=} w_{s',s}(\varphi)$$

**Theorem 1** *Let $s_n = do([a_1, ..., a_n], S_0), n \geq 0$ be a collection of situations, $\varphi$ be a BDF, $\Phi$ a general preference formula, and $w_s^{opt}(\Phi)$, $w_s^{pess}(\Phi)$ be the optimistic and pessimistic weights of $\Phi$ with respect to $s$. Then for any $0 \leq i \leq j \leq k \leq n$,*

*1. $\mathcal{D} \models \varphi[s_i]^{pess} \Rightarrow \mathcal{D} \models \varphi[s_j]$, $\mathcal{D} \not\models \varphi[s_i]^{opt} \Rightarrow \mathcal{D} \not\models \varphi[s_j]$,*

*2. $\left(w_{s_i}^{opt}(\Phi) = w_{s_i}^{pess}(\Phi)\right) \Rightarrow w_{s_j}(\Phi) = w_{s_i}^{opt}(\Phi) = w_{s_i}^{pess}(\Phi)$,*

*3. $w_{s_i}^{opt}(\Phi) \leq w_{s_j}^{opt}(\Phi) \leq w_{s_k}(\Phi)$, $w_{s_i}^{pess}(\Phi) \geq w_{s_j}^{pess}(\Phi) \geq w_{s_k}(\Phi)$*

Theorem 1 states that the optimistic weight is non-decreasing and never over-estimates the real weight. Thus, $f_\Phi$ is admissible and when used in best-first search, the search is optimal.

**Definition 14 (Evaluation function)** *Let $s = do(\vec{a}, S_0)$ be a situation and let $\Phi$ be a general preference formula. Then $f_\Phi(s) \stackrel{\text{def}}{=} w_s(\Phi)$ if $\vec{a}$ is a plan, otherwise $f_\Phi(s) \stackrel{\text{def}}{=} w_s^{opt}(\Phi)$.*

## 5 Implementation and Results

In this section, we describe our best-first search, ordered-task-decomposition planner. Figure 1 outlines the algorithm. **HTNPLAN** takes as input $\mathcal{P} = (s_0, w, D, pref)$ where $s_0$ is the initial state, $w$ the initial task network, $D$ is the HTN planning domain, and *pref* the general preference formula, and returns a sequence of ground primitive operators, i.e. a plan, and the weight of that plan.

The *frontier* is a list of nodes of the form [*optW, pessW, w, partialP, s, pref*], sorted by optimistic weight, pessimistic weight, and then by plan length. The frontier is initialized to the initial task network $w$, the empty partial plan, its *optW, pessW,* and *pref* corresponding to the progression and evaluation of the input preference formula in the initial state.

On each iteration of the **while** loop, **HTNPLAN** removes the first node from the frontier and places it in *current*. If $w$ is empty (i.e., $U$ is an empty set), the situation associated with this node is a terminating situation. Then **HTNPLAN** returns *current*'s partial plan and weight. Otherwise, it calls the function **EXPAND** with *current's* node as input.

**EXPAND** returns a new list of nodes that need to be added to the frontier. The new nodes are sorted by *optW, pessW,* and merged with the remainder of the frontier. If *w* is *nil* then the frontier is left as is. Otherwise, it generates a new set of nodes of the form [*optW, pessW, newW, newPartialP, newS, newProgPref*], one for each legal ground operator that can be reached by performing $w$ using a partial-order forward decomposition procedure (PFD) (Ghallab, Nau, and Traverso 2004). Currently **HTNPLAN** uses **SHOP2** (Nau et al. 2003) as its PFD. Hence, the current implementation of

**HTNPLAN**(*$s_0$, w, D, pref*)
*frontier* ← **INITFRONTIER**(*$s_0$, w, pref*)
**while** *frontier* ≠ ∅
  *current* ← **REMOVEFIRST**(*frontier*)
  % establishes values of *w, partialP, s, progPref*
  **if** *w*= ∅ and *optW=pessW* then **return** *partialP, optW*
  *neighbours* ← **EXPAND**(*w, D, partialP, s, progPref*)
  *frontier* ← **SORTNMERGE**(*neighbours, frontier*)
**return** [], ∞

Figure 1: A sketch of the **HTNPLAN** algorithm.

**HTNPLAN** is an implementation of **SHOP2** with user preferences. For each primitive task leading to terminating states, **EXPAND** generates a node of the same form but with *optW* and *pessW* replaced by the actual weight. If we reach the empty frontier, we return the empty plan.

**Theorem 2 (Soundness and Optimality)** *Let $\mathcal{P}$=$(s_0, w, D, \Phi)$ be a HTN planning problem with user preferences. Let $\pi$ be the plan returned by* **HTNPLAN** *from input $\mathcal{P}$. Then $\pi$ is a solution to the preference based HTN problem $\mathcal{P}$*

*Proof sketch:* We prove that the algorithm terminates appealing to the fact that the PFD procedure is sound and complete. We prove that the returned plan is optimal, by exploiting the correctness of progression of preference formula, and admissibility of our evaluation function.

### 5.1 Experiments

We implemented our preference-based HTN planner, **HTNPLAN**, on top of the LISP implementation of **SHOP2** (Nau et al. 2003). All experiments were run on a Pentium 4 HT, 3GHZ CPU, and 1 GB RAM, with a time limit of 1800 seconds (30 min). Since the optimality of **HTNPLAN**-generated plans was established in Theorem 2, our objective was to evaluate the effectiveness of our heuristics in guiding search towards the optimal plan, and to establish benchmarks for future study, since none currently exist.

We tested **HTNPLAN** with ZenoTravel and Logistics domains, which were adapted from the International Planning Competition (IPC). The ZenoTravel domain involves transporting people on aircrafts that can fly at two alternative speeds between locations. In the numeric variant the planes consume fuel at different rates according to the speed of travel, and distances between locations vary. The simple-time variant combines the speed of travel with the associated costs. We used both. The Logistics domain involves transporting packages to different destinations using trucks for delivery within cities and planes for between cities. Some of the preferences we used in the evaluations are as follows: we prefer that the high priority packages be delivered first, we prefer to use trucks with lower gas consumptions, and we prefer certain truck routes to another. The problems become harder as the number of objects and/or number of tasks in the domain increases.

In order to evaluate the effectiveness of **HTNPLAN** it would have been appealing to evaluate our planner with a preference-based planner that also makes use of procedural

| P | SHOP2 | | | HTNPLAN | | | |
|---|---|---|---|---|---|---|---|
| # | # Plan | NE | Time | NE | NC | Time | PL |
| 1 | 12 | 172 | 0.61 | 78 | 88 | 1.19 | 22 |
| 2 | 155 | 1628 | 8.60 | 448 | 547 | 9.45 | 26 |
| 3 | 230 | 2234 | 11.15 | 76 | 97 | 1.05 | 23 |
| 4 | 230 | 2234 | 11.10 | 361 | 413 | 4.67 | 23 |
| 5 | 485 | 6331 | 74.10 | 240 | 276 | 8.14 | 38 |
| 6 | 487 | 6226 | 113.20 | 1084 | 1218 | 63.60 | 46 |
| 7 | 720 | 6724 | 50.46 | 211 | 250 | 4.63 | 31 |
| 8 | 720 | 6724 | 50.90 | 699 | 808 | 13.63 | 28 |
| 9 | 851 | 9152 | 165.22 | 2689 | 3066 | 142.7 | 40 |
| 10 | 2069 | 23200 | 205.10 | 2290 | 2733 | 91.25 | 34 |
| 11 | 2875 | 27022 | 369.20 | 609 | 704 | 17.20 | 30 |
| 12 | 3956 | 35789 | 275.30 | 304 | 361 | 5.10 | 22 |
| 13 | >8K | >104K | >1800 | 150 | 167 | 5.64 | 63 |
| 14 | >13K | >143K | >1800 | 2153 | 2922 | 80.01 | 35 |
| 15 | >13K | >136K | >1800 | 1624 | 1910 | 36.02 | 29 |
| 16 | >31K | >293K | >1800 | 1510 | 1848 | 24.80 | 21 |

(a) ZenoTravel domain

| P | SHOP2 | | | HTNPLAN | | | |
|---|---|---|---|---|---|---|---|
| # | # Plan | NE | Time | NE | NC | Time | PL |
| 1 | 80 | 1297 | 1.27 | 73 | 93 | 0.64 | 14 |
| 2 | 90 | 540 | 0.28 | 19 | 24 | 0.20 | 12 |
| 3 | 808 | 4597 | 4.00 | 301 | 404 | 2.22 | 18 |
| 4 | 1024 | 10665 | 79.95 | 1626 | 1820 | 49.56 | 42 |
| 5 | 1024 | 10665 | 79.95 | 98 | 115 | 2.30 | 42 |
| 6 | 1260 | 6320 | 4.66 | 130 | 172 | 1.04 | 14 |
| 7 | 2178 | 15104 | 17.20 | 27 | 32 | 0.22 | 20 |
| 8 | 2520 | 14728 | 12.47 | 29 | 40 | 0.33 | 16 |
| 9 | 21776 | 114548 | 119.1 | 866 | 1163 | 9.44 | 15 |
| 10 | >28K | >264K | >1800 | 1062 | 1437 | 13.21 | 19 |
| 11 | >28K | >239K | >1800 | 1767 | 2417 | 32.76 | 14 |
| 12 | >30K | >118K | >1800 | 1417 | 1925 | 21.07 | 20 |
| 13 | >42K | >368K | >1800 | 2398 | 2968 | 82.62 | 42 |
| 14 | >54K | >407K | >1800 | 858 | 1088 | 19.26 | 33 |
| 15 | >65K | >428K | >1800 | 37 | 48 | 0.46 | 24 |
| 16 | >67K | >376K | >1800 | 451 | 618 | 5.14 | 22 |

(b) Logistic domain

Figure 2: Our criteria for comparisons are number of Nodes Expanded (NE), number of applied operators; number of Nodes Considered (NC), the number of nodes that were added to the frontier, and time measured in seconds. Note NC is equal to NE for **SHOP2**. PL is the Plan Length and # Plan is the total number of plans.

control knowledge. But since no comparable planner exists, and it would not have been fair to compare **HTNPLAN** with a preference-based planner that does not use control knowledge, we compared **HTNPLAN** with **SHOP2**, using a brute-force technique for **SHOP2** to determine the optimal plan. In particular, as is often done with Markov Decision Processes, **SHOP2** generated all plans that satisfied the HTN specification and then evaluated each to find the optimal plan. Note that the times reported for **SHOP2** do not actually include the time for posthoc preference evaluation, so they are lower bounds on the time to compute the optimal plan.

Figure 2 reports our experimental results for ZenoTravel and the Logistics domain. The problems varied in preference difficulty and are shown in the order of difficulty with respect to number of possible plans (# Plan) that satisfy the HTN control.

The results show that, in all but the first two cases of the ZenoTravel domain, **SHOP2** required more time to find the optimal plan, and expanded more nodes. In particular note that in a number of problems, for example problems 13 and 14 **SHOP2** ran out of time (1800 seconds) while **HTNPLAN** found the optimal plan well within the time limit. Also note that **HTNPLAN** expands far fewer nodes in comparison to **SHOP2**, illustrating the effectiveness of our evaluation function in guiding search.

## 6 Summary and Related Work

In this paper, we addressed the problem of generating preferred plans by combining the procedural control knowledge of HTNs with rich qualitative user preferences. The most significant contributions of this paper include: $\mathcal{LPH}$, a rich HTN-tailored preference specification language, developed as an extension of a previously existing language; an approach to (preference-based) HTN planning based on forward-chaining heuristic search, that exploits progression to evaluate the satisfaction of preferences during planning; a sound and optimal implementation of an ordered-task-decomposition preference-based HTN planner; and leveraging previous research, an encoding of HTN planning with preferences in the situation calculus, that enabled us to prove our theoretical results. While the implementation we present here exploits **SHOP2**, the language and techniques proposed are relevant to a broad range of HTN planners.

In previous work, we addressed the problem of integrating user preferences into Web service composition (Sohrabi, Prokoshyna, and McIlraith 2006). To that end, we developed a Golog-based composition engine that also exploits heuristic search. It similarly uses an optimistic heuristic. The language used in that work was $\mathcal{LPP}$ and had no Web-service or Golog-specific extensions for complex actions. This paper's HTN-tailored language and HTN-based planner are significantly different.

Preference-based planning has been the subject of much interest in the last few years, spurred on by an International Planning Competition (IPC) track on this subject. A number of planners were developed, all based on the the competition's PDDL3 language (Gerevini and Long 2005). Our work is distinguished in that it exploits *procedural* (action-centric) domain control knowledge in the form of an HTN, and action-centric and state-centric preferences in the form of $\mathcal{LPH}$. In contrast, the preferences and domain control in PDDL3 and its variants are strictly state-centric. Further, $\mathcal{LPH}$ is *qualitative* whereas PDDL3 is quantitative, appealing to a numeric objective function. We contend that qualitative, action- or task-centric preferences are often more compelling and easier to elicit that their PDDL3 counterparts.

While no other HTN planner can perform true preference-based planning, **SHOP2** (Nau et al. 2003) and **ENQUIRER** (Kuter et al. 2004) handle some simple user constraints. In particular the order of methods and sorted preconditions in a domain description specifies a user preference over which method is more preferred to decompose a task. Hence users may write different versions of a domain description to specify simple preferences. However, unlike

**HTNPLAN** the user constraints are treated as hard constraints and (partial) plans that do not meet these constraints will be pruned from the search space. Further, there is no way to handle temporally extended hard or soft constraints in **SHOP2**. We used progression in our approach to planning precisely to deal with these interesting preferences. Were we limiting the expressive power of preferences to **SHOP2**-like method ordering, we would have created a different planner. Interestingly, **SHOP2** method ordering can still be exploited in our approach, but requires a mechanism that is beyond the scope of this paper.

Finally, the **ASPEN** planner (Rabideau, Engelhardt, and Chien 2000) performs a simple form of preference-based planning, focused mainly on preferences over resources and with far less expressivity than our preference language. Moreover, unlike our planner ASPEN will not perform well on problems where preferences are interacting, nested, and not local to any specific activity. Nevertheless, **ASPEN** has the ability to plan with HTN-like task decomposition, and as such, this work is related in spirit, though not in approach to our work.

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

## References

Bienvenu, M.; Fritz, C.; and McIlraith, S. A. 2006. Planning with qualitative temporal preferences. In *Proceedings of the 10th International Conference on Knowledge Representation and Reasoning (KR)*, 134–144.

De Giacomo, G.; Lespérance, Y.; and Levesque, H. 2000. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence* 121(1–2):109–169.

Gabaldon, A. 2002. Programming hierarchical task networks in the situation calculus. In *AIPS'02 Workshop on On-line Planning and Scheduling*.

Gabaldon, A. 2004. Precondition control and the progression algorithm. In *Proceedings of the 9th International Conference on Knowledge Representation and Reasoning (KR)*, 634–643. AAAI Press.

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

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Hierarchical Task Network Planning. Automated Planning: Theory and Practice*. Morgan Kaufmann.

Kuter, U.; Sirin, E.; Nau, D. S.; Parsia, B.; and Hendler, J. A. 2004. Information gathering during planning for web service composition. In *Proceedings of the 3rd International Semantic Web Conference (ISWC)*, 335–349.

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.

Rabideau, G.; Engelhardt, B.; and Chien, S. A. 2000. Using generic preferences to incrementally improve plan quality. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*, 236–245.

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

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