Specifying and Generating Preferred Plans

Meghyn Bienvenu and **Sheila McIlraith**
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada.
{meghyn,sheila}@cs.toronto.edu

Abstract

In this paper, we address the problem of specifying and generating preferred plans using rich, qualitative user preferences. We propose a logical language for specifying preferences over the evolution of states and actions associated with a plan. We provide a semantics for our first-order preference language in the situation calculus and prove that progression of our preference formulae preserves this semantics. This leads to the development of **PPLAN**, a bounded best-first search planner that computes preferred plans. Our preference language is amenable to integration with many existing planners, and beyond planning, can be used to support arbitrary dynamical reasoning tasks.

1 Introduction

Research in automated planning has historically focused on classical planning – generating a sequence of actions to achieve a user-defined goal given a specification of a domain and an initial state. Nevertheless, one need look no further than the pervasive problem of travel planning to observe that generating a plan is not the only challenge. In many real-world settings, plans are plentiful, and it is the generation of *high-quality plans* meeting users' preferences and constraints that presents the biggest challenge [9].

In this paper we examine the problem of preference-based planning – generating a plan that not only achieves a user-defined goal, but that also conforms, where possible, to a user's preferences over properties of the plan. To that end, we propose a first-order language for specifying domain-specific, qualitative user preferences. Our language is rich, supporting preferences over the evolution of actions and states leading to achievement of a goal. Our language harnesses much of the expressive power of first-order and linear temporal logic. We define the semantics of our preference language in the situation calculus [11]. Nevertheless, nothing requires that the planner be implemented using deductive plan synthesis in the situation calculus. Indeed our planner **PPLAN**, a bounded best-first search planner, is a forward-chaining planner, in the spirit of TLPlan [1] and TALPlan [8], that exploits progression of preference formulae to more efficiently compute preferred plans. Experimental results with **PPLAN** illustrate the efficacy of our best-first heuristic.

There is a tremendous body of research on preferences and utility theory within the field of economics and related disciplines. Research on the specification of qualitative preferences has predominantly focused on static preferences (e.g., [4]) with a view towards preference elicitation. In a number of cases, such limited preference languages render states incomparable. Recent work extends the expressiveness of such languages somewhat (e.g., [5]). In the area of dynamic preferences, there are several recent and notable pieces of work. Son and Pontelli [13] have developed a propositional language for planning with preferences together with an implementation using answer-set programming. Indeed we leverage their preference language $\mathcal{PP}$ in our work, contrasting the two in subsequent sections. Also notable is the work of Delgrande et al. [6], who have developed a framework for characterizing preferences and properties of preference-based planning. Rossi and colleagues (e.g., [12]) exploit their work on soft constraints to develop temporal constraints for reasoning in temporal domains, sometimes with uncertainty. Their qualitative preferences are less expressive than ours, but their computational framework is more general. Finally research on decision-theoretic planning and MDPs also addresses the general problem of generating preferred plans [10]. Nevertheless, the elicitation of preferences in terms of Markovian numeric utilities makes these approaches less applicable to the types of preferences we are interested in capturing.

Our work is distinguished for a number of reasons. Our preference specification language is first-order, and thus more expressive. We provide compelling means of expressing relative importance of different preferences, thus reducing the incomparability of potential plans. By appealing to the situation calculus, where different plan trajectories are simply different branches (situations) within a single model of the domain, we are able to talk about the relative merits of evolving plans *within* the language, rather than characterizing preferences in terms of preferred models. Finally, our implementation of a preference-based planning system uses a forward chaining planner, making it conducive to online interleaved planning and execution, and to mixed-initiative planning with preferences.

In Section 2 we provide a brief review of the situation calculus. In Section 3, we describe the syntax and semantics of our preference language for planning, illustrating its use through a motivating example which is carried through the paper. With the semantics of our preferences in hand, we return to the general problem of planning with preferences in Section 4, proving that progression preserves the semantics of our preferences. In Section 5, we describe our implementation of **PPLAN**, a bounded best-first forward planner that plans with preferences, and prove the correctness of our algorithm. We conclude with a summary.

2 Preliminaries

The situation calculus is a logical language for specifying and reasoning about dynamical systems [11]. 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., $F(\vec{x}, s)$. In this paper, we distinguish between the set of fluent predicates $\mathcal{F}$ and the set of non-fluent predicates $\mathcal{R}$ representing properties that do not change over time. A situation s is a *history* of the primitive actions $a \in \mathcal{A}$, performed from an initial, distinguished situation S_0. The function $do(a, s)$ maps a situation and an action into a new situation. The theory induces a tree of situations rooted at S_0.

A basic action theory in the situation calculus $\mathcal{D}$ comprises four *domain-independent foundational axioms* and a set of *domain-dependent axioms*. The foundational axioms Σ define the situations, their branching structure, and the situation predecessor relation $\sqsubset$. $s \sqsubset s'$ states that situation s precedes situation s' in the situation tree. Σ includes a second-order induction axiom. The domain-dependent axioms are strictly first-order and are of the following form:

• successor state axioms $\mathcal{D}_{SS}$, one for every fluent $F \in \mathcal{F}$, which capture the effects of actions on the truth value of F.
• action precondition axioms $\mathcal{D}_{ap}$, one for every action a in the domain. These define the fluent $Poss(a, s)$, the conditions underwhich it's possible to execute an action a in situation s.
• axioms $\mathcal{D}_{S_0}$ describing the initial situation.
• unique names axioms for actions $\mathcal{D}_{una}$.

Details of the form of these axioms can be found in [11]. Following convention and to enhance readability, we will generally refer to fluents in situation-suppressed form, e.g., $at(home)$ rather than $at(home, s)$.

A *planning problem* Δ is a tuple $\langle \mathcal{D}, G \rangle$ where $\mathcal{D}$ is a basic action theory and G is a goal formula, representing properties that must hold in the final situation. In the situation calculus, planning is characterized as deductive plan synthesis. Given a planning problem $\langle \mathcal{D}, G \rangle$, the task is to determine a situation $s = do(a_n, (do(a_{n-1}, \ldots, do(a_1, S_0))))$[1], i.e., a sequence of actions from S_0, such that:

$$\mathcal{D} \models (\exists s).executable(s) \wedge G(s)$$

where $executable(s) \stackrel{\text{def}}{=} (\forall a, s').do(a, s') \sqsubseteq s \supset Poss(a, s')$.

We refer to this situation $s = do(\vec{a}, S_0)$ as the *plan trajectory*, and the sequence of actions $\vec{a} = a_1 a_2 \ldots a_n$ as the *associated plan*. The *length* of this plan is n. A planning problem $\langle \mathcal{D}, G \rangle$ is *solvable* if it has at least one plan. It is *k-solvable* if it has a plan of length k or less.

[1]Which we abbreviate to $do([a_1, \ldots, a_n], S_0)$, or $do(\vec{a}, S_0)$.

3 Preference Specification

In this section we describe the syntax and semantics of our first-order preference language. We illustrate the concepts in this paper in terms of the following motivating example.

The Dinner Example: It's dinner time, and Claire is tired and hungry. Her goal is to be at home with her hunger sated. There are three possible ways for Claire to get food: she can cook something at home, order in take-out food, or go to a restaurant. To cook a meal, Claire needs to know how to make the meal and she must have the necessary ingredients, which might require a trip to the grocery store. She also needs a clean kitchen in which to prepare her meal. Ordering take-out is much simpler; she only has to order and eat the meal. Going to a restaurant requires getting to the restaurant, ordering, eating, and then returning home.

This example is easily encoded in any number of planning systems, and given a specification of Claire's initial state, a planner could generate numerous plans that achieve Claire's goal. Nevertheless, like many of us, Claire has certain preferences concerning where and what she eats that make some plans better than others. It is the definition of these preferences and the generation of these preferred plans that is the focus of this paper.

3.1 A First-Order Preference Language

In this section we present the syntax of a first-order language for expressing preferences about dynamical systems. Our preference language modifies and extends the preference language *PP* recently proposed by Son and Pontelli [13]. Following their work, we provide a hierarchy of preference formulae comprising basic desire formulae, atomic preference formulae, and general preference formulae, as defined below. Subsequent reference to a *preference formula* refers to a general preference formula, which encompasses both basic desire formulae and atomic preference formulae.

Definition 1 (Basic Desire Formula (BDF)). A basic desire formula is a sentence drawn from the smallest set $\mathcal{B}$ where:

1. $\mathcal{F} \subset \mathcal{B}$
2. $\mathcal{R} \subset \mathcal{B}$
3. If $f \in \mathcal{F}$, then **final**$(f) \in \mathcal{B}$
4. If $a \in \mathcal{A}$, then **occ**$(a) \in \mathcal{B}$
5. If φ, φ_1, and φ_2 are in $\mathcal{B}$, then so too are $\neg\psi$, $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$, $(\exists x)\varphi$, $(\forall x)\varphi$, **next**(ψ), **always**(ψ), **eventually**(ψ), and **until**(ψ_1, ψ_2)

BDFs establish preferred situations. By combining BDFs using boolean and temporal connectives, we are able to express a wide variety of properties of situations. We illustrate their use with some sample BDFs from our motivating example.

$$hasIngrnts(spag) \wedge knowsHowToMake(spag) \quad \text{(P1)}$$
$$(\exists x).hasIngrnts(x) \wedge knowsHowToMake(x) \quad \text{(P2)}$$
$$\textbf{final}(kitchenClean) \quad \text{(P3)}$$
$$(\exists x).\textbf{eventually}(\textbf{occ}(cook(x))) \quad \text{(P4)}$$
$$(\exists x).(\exists y).\textbf{eventually}(\textbf{occ}(orderTakeout(x,y))) \quad \text{(P5)}$$
$$(\exists x).(\exists y).\textbf{eventually}(\textbf{occ}(orderRestaurant(x,y))) \quad \text{(P6)}$$
$$\textbf{always}(\neg((\exists x)(\exists y)\textbf{occ}(drive(x,y)) \wedge isSnowing)) \quad \text{(P7)}$$
$$\textbf{always}(\neg((\exists x).\textbf{occ}(eat(x)) \wedge chinese(x))) \quad \text{(P8)}$$

The first BDF, P1, states that in the initial situation Claire has the ingredients and the know-how to cook spaghetti. P2 is more general, expressing that in the initial situation Claire has the ingredients to make something she knows how to make. Observe that fluent formulae that are not inside temporal connectives refer only to the initial situation. P3 states that in the final situation the kitchen is clean. P4 - P6 tell us respectively that at some point in time Claire cooked something, ordered something from take-out, or ordered something at a restaurant. The BDF P7 tells us that at no point does Claire drive while it is snowing. Finally P8 tells us that Claire never eats any chinese food.

While BDFs alone enable us to express many interesting preferences, we cannot express preferences between alternatives. For example, we cannot say that Claire prefers cooking to ordering take-out. To do so, we define Atomic Preference Formulae, following the definition in [13].

Definition 2 (Atomic Preference Formula). An atomic preference formula is a formula $\varphi_0 \gg \varphi_1 \gg ... \gg \varphi_n$, where $n \geq 0$ and each φ_i is a basic desire formula. When $n = 0$, atomic preference formulae correspond to BDFs.

An atomic preference formula expresses a preference over alternatives. For example, Claire can express her preference over what to eat (pizza, followed by spaghetti, followed by crêpes) using P9[2].

$$\textbf{occ}'(eat(pizza)) \gg \textbf{occ}'(eat(spag)) \gg \textbf{occ}'(eat(crêpes)) \quad \text{(P9)}$$

If Claire is in a hurry, tired, or very hungry, she may be more concerned about how long she will have to wait for her meal:

$$P5 \gg P2 \wedge P4 \gg P6 \gg \neg P2 \wedge P4 \quad \text{(P10)}$$

This says that Claire first choice is take-out, followed by cooking if she has the ingredients for something she knows how to make, followed by going to a restaurant, and lastly cooking when it requires a trip to the grocery store.

Again, an atomic preference represents a preference over alternatives, φ_i. That means we are only trying to satisfy one of the BDFs φ_i, and we would like to satisfy the BDF with lowest index possible. Consequently, if Claire eats pizza *and* crêpes, this is no better nor worse with respect to P9 than situations in which Claire eats only pizza, and it is strictly better than situations in which she just eats crêpes. Also note that there is always implicitly one last option, which is to satisfy none of the φ_i, and this option is the least preferred.

[2]For legibility, we abbreviate **eventually**(**occ**(φ)) by **occ**$'(\varphi)$, and we refer to the preference formulae by their labels.

Atomic preference formulae contribute significantly to the expressivity of our preference language, but we still lack a way to combine atomic preferences together. Our third and final class of preference formulae will provide us with several useful methods for combining preference formulae.

Definition 3 (General Preference Formula). A formula Φ is a general preference formula if one of the following holds:

- Φ is an atomic preference formula
- Φ is $\gamma : \Psi$, where γ is a basic desire formula and Ψ is a general preference formula [Conditional]
- $\Phi = !\,\Psi$, for Ψ a general preference formula [Negation]
- Φ is one of
  - $\Psi_0 \,\&\, \Psi_1 \,\&\, ... \,\&\, \Psi_n$ [General And]
  - $\Psi_0 \mid \Psi_1 \mid ... \mid \Psi_n$ [General Or]
  - $\Psi_0 \rhd \Psi_1 \rhd ... \rhd \Psi_n$ [Lex Order]
  - $\Psi_0 \vec{\&} \Psi_1 \vec{\&} ... \vec{\&} \Psi_n$ [Ordered And]

where $n \geq 1$ and each Ψ_i is a general preference formula.

Here are some example general preference formulae:

$$P2 : P4 \quad (P11) \qquad !\,P9 \quad (P12)$$
$$P9 \,\&\, P10 \quad (P13) \qquad P9 \mid P10 \quad (P14)$$
$$P9 \rhd P10 \quad (P15) \qquad P9 \vec{\&} P10 \quad (P16)$$

The preference P11 states that if Claire initially has the ingredients for something she can make, then she should cook something. A negation preference $!\,\Phi$ does the opposite of its component preference Φ. In the case of P12, that would mean Claire's most preferred option is eating something other than pizza, crêpes, or spaghetti, and otherwise she prefers crêpes to spaghetti to pizza. The remaining preferences show the various ways we can combine Claire's food and time preferences. P13 is used when Claire wants both her food and time preferences to be satisfied as much as possible. P14 can be used if she would be content if either of the two were satisfied. P15 tells us that while Claire cares about both her preferences, her food preference is more important. Finally, P16 means that she wants to satisfy both preferences as much as possible, like P13, but if two solutions are "tied", she prefers the solution which is better with respect to the food preference P9. Preferences of this type allow us to lessen incomparability even when we have many preferences of (almost) equal importance.

This concludes our description of the syntax of our preference language. Our language extends and modifies the *PP* language recently proposed by Son and Pontelli [13]. Quantifiers, variables, and non-fluent relations have been added to BDFs. In *PP* it is impossible to talk about arbitrary action or fluent arguments or their properties, and it is difficult or even impossible to express the kinds of preferences given above. Further, we have extended the definition of general preferences to include *Conditional* and *Ordered And* preferences, two compelling preference modes. *PP* does not have any conditional constructs, nor does it provide a way of dealing with incomparability. We have also provided a more intuitive semantics for *General And* and *General Or* preferences. Finally, we differ significantly in our semantics, which follows.

3.2 The Semantics of our Language

We appeal to the situation calculus to define the semantics of our preference language. Preference formulae are interpretted as situation calculus formulae. Further, we associate a weight with a situation term, dependent upon how greatly it deviates from satisfying a preference formula. 0 indicates satisfaction, i represents i steps away from the most preferred situation. Weights were necessary to differentiate situations that would be deemed “incomparable” in less expressive preference languages (e.g., [4]). Preference formulae are evaluated relative to an action theory $\mathcal{D}$. Since preference formulae may refer to properties that hold at various situations in a situation history, we use the notation $\varphi[s, s']$, proposed by Gabaldon [7], to explicitly denote that φ holds in the sequence of situations originating in s and terminating in $s' = do([a_1, \ldots, a_n], s)$. Recall that fluents are represented in situation-suppressed form and that we use the notation $F[s]$ to denote the re-insertion of situation terms.

We now show how to interpret BDFs in the situation calculus. If $f \in \mathcal{F}$, we will simply need to re-insert the situation argument, yielding:

$$f[s', s] = f[s']$$

For $r \in \mathcal{R}$, we have nothing to do as r is already a situation calculus formula, so:

$$r[s', s] = r$$

A BDF $final(f)$ just means that the fluent f holds in the final situation, giving the following:

$$\mathbf{final}(f)[s', s] = f[s]$$

The BDF $\mathbf{occ}(a)$ tells us that the first action executed is a, which can be written as:

$$\mathbf{occ}(a)[s', s] = do(a, s') \sqsubseteq s$$

The boolean connectives and quantifiers are already part of the situation calculus and so require no translation. Finally, we are left with just the temporal connectives, which we interpret in exactly the same way as in [7]:[3]

$$\begin{aligned}
\mathbf{eventually}(\varphi)[s', s] &= (\exists s_1 : s' \sqsubseteq s_1 \sqsubseteq s)\varphi[s_1, s] \\
\mathbf{always}(\varphi)[s', s] &= (\forall s_1 : s' \sqsubseteq s_1 \sqsubseteq s)\varphi[s_1, s] \\
\mathbf{next}(\varphi)[s', s] &= (\exists a).do(a, s') \sqsubseteq s \wedge \varphi[do(a, s'), s] \\
\mathbf{until}(\varphi, \psi)[s', s] &= (\exists s_1 : s' \sqsubseteq s_1 \sqsubseteq s)\{\psi[s_1, s] \wedge \\
&\quad (\forall s_2 : s' \sqsubseteq s_2 \sqsubseteq s_1)\varphi[s_2, s]\}
\end{aligned}$$

Since each BDF is shorthand for a situation calculus expression, a simple model-theoretic semantics follows.

Definition 4 (Basic Desire Satisfaction). Let $\mathcal{D}$ be an action theory. A situation $s = do([a_1, ..., a_n], S_0)$ satisfies a basic desire formula φ just in the case that

$$\mathcal{D} \models \varphi[S_0, s]$$

We define $w_s(\varphi)$ to be the weight of situation s wrt BDF φ. $w_s(\varphi) = 0$ if s satisfies φ, otherwise $w_s(\varphi) = 1$.

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

We can extend this definition to the more general case as follows.

Definition 5. Let $\mathcal{D}$ be an action theory, and let s' and s be two situations such that $s' \sqsubseteq s$. The situations beginning in s' and terminating in s satisfy φ 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 φ. $w_{s',s}(\varphi) = 0$ if φ is satisfied, otherwise $w_{s',s}(\varphi) = 1$.

Clearly Definition 4 is just a special case of Definition 5 since w_s is simply short-hand for $w_{S_0,s}$. In most circumstances, the short-hand w_s notation of Definition 4 will suffice, with the advantage of being easier to read and understand. Consequently, we use it throughout the paper. Nevertheless, in proving properties of our semantics relative to progression, we will revert to the two-situation notation of Definition 5.

Example 1: Consider the plan trajectory $s =$ $do([cleanDishes, cook(crêpes), eat(crêpes), cleanDishes], S_0)$ and the initial database $\mathcal{D}_{S_0} =$
$\{hungry(S_0), hasIngrnts(spag, S_0), at(home, S_0),$
$hasIngrnts(crêpes, S_0), knowsHowToMake(crêpes)\}$.
Then we have the following:

$$\begin{array}{lll}
w_s(P1) = 1 & w_s(P2) = 0 & w_s(P3) = 0 \\
w_s(P4) = 0 & w_s(P5) = 1 & w_s(P6) = 1 \\
w_s(P7) = 0 & w_s(P8) = 0 &
\end{array}$$

Definition 6 (Atomic Preference Satisfaction). Let s be a situation and $\Phi = \varphi_0 \gg \varphi_1 \gg ... \gg \varphi_n$ be an atomic preference formula. Then $w_s(\Phi) = \min\{i : w_s(\varphi_i) = 0\}$, if such an i exists, and $w_s(\Phi) = n + 1$ otherwise.

Keeping D_{S_0} and s as above, we evaluate our two atomic preferences introduced earlier:

$$w_s(P9) = 2 \qquad w_s(P10) = 1$$

Intuitively, the weight of a situation with respect to a preference formula denotes the number of steps away from a most preferred situation. As a consequence, given two atomic preference formulae $p_0 \gg p_1$ and $q_0 \gg q_1 \gg \ldots \gg q_n$, satisfying p_1 in the first formula yields the same weight as satisfying q_1 in the second. Intuition might tell some that there should be a normalization, that satisfying p_1 should have the same weight as satisfying q_n. This is not the intended interpretation. Atomic preference formulae do not represent a complete ordering of all outcomes. For instance, there may be many less preferred p_i's not listed in the formula, particularly since this is first-order. Furthermore, since preferences are qualitative, one cannot assume that the difference in preference between p_0 and p_1 is the same as the difference between q_0 and q_1, or any φ_i and φ_{i+1}.

Definition 7 (General Preference Satisfaction). Let s be a situation and Φ 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

- $w_s(\gamma : \Psi) = \begin{cases} 0 & \text{if } w_s(\gamma) = 1 \\ w_s(\Psi) & \text{otherwise} \end{cases}$
- $w_s(\,!\,\Psi) = \max^w(\Psi) - w_s(\Psi)$
- $w_s(\Psi_0 \,\&\, \Psi_1 \,\&\, ... \,\&\, \Psi_n) = \sum_{i=0}^{n} w_s(\Psi_i)$
- $w_s(\Psi_0 \mid \Psi_1 \mid ... \mid \Psi_n) = \min\{w_s(\Psi_i) : 1 \leq i \leq n\}$
- $w_s(\Psi_0 \triangleright \Psi_1 \triangleright ... \triangleright \Psi_n)$
  $= \sum_{i=0}^{n} [w_s(\Psi_i) \times \prod_{j=i+1}^{n} \max^w(\Psi_j)]$
  $= w_s(\Phi_0) \times (\max^w(\Psi_1) \times ... \times \max^w(\Psi_n)) +$
  $w_s(\Phi_1) \times (\max^w(\Psi_2 \times ... \max^w(\Psi_n)) \ + \ ... +$
  $w_s(\Phi_{n-1}) \times \max^w(\Psi_n) \ + \ w_s(\Phi_n)$

where $\max^w(\Phi)$ is a syntactic notion, denoting the maximum possible weight Φ could ever assign to a situation. ($\max^w(\Phi) = 1$ if Φ is a BDF, $\max^w(\varphi_0 \gg \ldots \gg \varphi_n) = n+1$, etc.). A full inductive definition is found in [3].

Returning to Example 1,

- $w_s(P2 : P4) = w_s(P4) = 0$
- $w_s(\,!\,P9) = \max^w(P9) - w_s(P9) = 3 - 2 = 1$
- $w_s(P9 \,\&\, P10) = 2 + 1 = 3$
- $w_s(P9 \mid P10) = \min(\{2, 1\}) = 1$
- $w_s(P9 \triangleright P10)$
  $= \ w_s(P9) \times \max^w(P10) + \ w_s(P10)$
  $= 2 \times 4 + 1 = 9$

The weight of *Ordered And* formulae $\Psi_0 \,\vec{\&}\, \Psi_1 \,\vec{\&}\, ... \,\vec{\&}\, \Psi_n$ is defined with respect to a weighted-lexicographic order. This order first orders tuples of numbers by the sum of their elements (i.e. by $w_s(\Psi_0 \,\&\, \Psi_1 \,\&\, ... \,\&\, \Psi_n)$) and then applies the traditional lexicographic ordering within each equivalence class. The weight is then just the index in this ordering. The full definition can be found in [3].

Definition 8 (Preferred Situations). A situation s_1 is preferred to a situation s_2 with respect to a preference formula Φ, written $Pref(s_1, s_2, \Phi)$, if $w_{s_1}(\Phi) < w_{s_2}(\Phi)$. Situations s_1 and s_2 are indistinguishable with respect to a preference formula Φ if $w_{s_1}(\Phi) = w_{s_2}(\Phi)$.

4 Planning with Preferences

With a preference language in hand, we return to the problem of planning with preferences.

Definition 9 (Preference-Based Planning Problem). A preference-based planning problem is a tuple $\langle \mathcal{D}, G, \Phi \rangle$, where $\mathcal{D}$ is an action theory, G is the goal, and Φ is a preference formula.

Definition 9 is trivially extended to generate plans that adhere to hard constraints or domain knowledge Φ_c (e.g., [8; 1]) by requiring that $\mathcal{D} \models \Phi_c[S_0, s]$ additionally holds.

Definition 10 (Preferred Plan). Consider a preference-based planning problem $\langle \mathcal{D}, G, \Phi \rangle$ and two plans $\vec{a_1}$ and $\vec{a_2}$. We say that plan $\vec{a_1}$ is preferred to plan $\vec{a_2}$ if and only if $Pref(do(\vec{a_1}, S_0), do(\vec{a_2}, S_0), \Phi)$.

Definition 11 (Ideal Plan). Given a preference-based planning problem, an ideal plan is any plan $\vec{a}$ such that $w_{do(\vec{a}, S_0)}(\Phi) = 0$.

Definition 12 (Optimal Plan). A plan $\vec{a}$ is an optimal plan with respect to a preference-based planning problem $\langle \mathcal{D}, G, \Phi \rangle$ iff:

$$\begin{aligned} \mathcal{D} \quad \models \quad & (\exists s).\, executable(s) \wedge G(s) \wedge s = do(\vec{a}, S_0) \\ & \wedge \neg \exists s'.\, [executable(s') \wedge G(s') \wedge Pref(s', s, \Phi)]. \end{aligned}$$

Definition 13 (k-Optimal Plan). Given a preference-based planning problem $\langle \mathcal{D}, G, \Phi \rangle$ and a length bound k, a k-optimal plan is any plan $\vec{a} = a_1, \ldots, a_l$ where $l \leq k$ and

$$\begin{aligned} \mathcal{D} \quad \models \quad & (\exists s).\, executable(s) \wedge G(s) \wedge s = do(\vec{a}, S_0) \\ & \wedge \neg \exists s'.\, [s' = do([b_1, \ldots, b_m], S_0) \wedge m \leq k \\ & \qquad \wedge executable(s') \wedge G(s') \wedge \ Pref(s', s, \Phi)]. \end{aligned}$$

Definition 14 (q-Satisfactory Plan). Given a preference-based planning problem and a quality bound q, a q-satisfactory plan is any plan $\vec{a}$ such that $w_{do(\vec{a}, S_0)}(\Phi) \leq q$.

4.1 Progression

In the next section we will present an algorithm for planning with preferences, based on forward-chaining planning. As has been done with control knowledge containing linear temporal logic formulae [1; 8], we evaluate our preference formulae by progressing them as we construct our plan. Progression takes a situation and a temporal logic formula (TLF), evaluates the TLF with respect to the state of the situation and generates a new formula representing those aspects of the TLF that remain to be satisfied in subsequent situations. In this section, we define the notion of progression with respect to our preference formulae and prove that the semantics of preference formulae is preserved through progression.

In order to define the progression operator, 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 further add the BDF **occNext**$(a), a \in \mathcal{A}$, to capture the progression of **occ**(a).

Definition 15 (Progression of a Basic Desire Formula). Let s be a situation, and let φ be a basic desire formula. The progression of φ through s, written $\rho_s(\varphi)$, is given by:

- If $\varphi \in \mathcal{F}$, then $\rho_s(\varphi) = \begin{cases} \text{TRUE} & \text{if } \mathcal{D} \models \varphi[s] \\ \text{FALSE} & \text{otherwise} \end{cases}$
- If $\varphi \in \mathcal{R}$, then $\rho_s(\varphi) = \begin{cases} \text{TRUE} & \text{if } \mathcal{D} \models \varphi \\ \text{FALSE} & \text{otherwise} \end{cases}$
- If $\varphi =$ **occ**(a), then $\rho_s(\varphi) =$ **occNext**(a)
- If $\varphi =$ **occNext**(a), then
  $\rho_s(\varphi) = \begin{cases} \text{TRUE} & \text{if } \mathcal{D} \models \exists s'.s = do(a, s') \\ \text{FALSE} & \text{otherwise} \end{cases}$
- If $\varphi =$ **final**(ψ), then $\rho_s(\varphi) = \varphi$
- If $\varphi = \neg\psi$, then $\rho_s(\varphi) = \neg\rho_s(\psi)$
- If $\varphi = \psi_1 \wedge \psi_2$, then $\rho_s(\varphi) = \rho_s(\psi_1) \wedge \rho_s(\psi_2)$
- If $\varphi = \psi_1 \vee \psi_2$, then $\rho_s(\varphi) = \rho_s(\psi_1) \vee \rho_s(\psi_2)$
- If $\varphi = (\exists x)\psi$, then $\rho_s(\varphi) = \bigvee_{c \in \mathcal{C}} \rho_s(\psi^{c/x})$[4]
- If $\varphi = (\forall x)\psi$, then $\rho_s(\varphi) = \bigwedge_{c \in \mathcal{C}} \rho_s(\psi^{c/x})$
- If $\varphi =$ **next**(ψ), then $\rho_s(\varphi) = \psi$
- If $\varphi =$ **always**(ψ), then $\rho_s(\varphi) = \rho_s(\psi) \wedge \varphi$

- If $\varphi = \textbf{eventually}(\psi)$, then $\rho_s(\varphi) = \rho_s(\psi) \vee \varphi$
- If $\varphi = \textbf{until}(\psi_1, \psi_2)$, then
  $\rho_s(\varphi) = (\rho_s(\psi_1) \wedge \varphi) \vee \rho_s(\psi_2)$
- If $\varphi =$ TRUE or $\varphi =$ FALSE, then $\rho_s(\varphi) = \varphi$

Returning to Example 1,

– $\rho_s(occ(cleanDishes)) =$ TRUE

– $\rho_s(\textbf{always}(kitchenClean))$
$= \rho_s(kitchenClean) \wedge \textbf{always}(kitchenClean)$
$=$ FALSE $\wedge\ \textbf{always}(kitchenClean)$

– $\rho_s((\exists x).hasIngrnts(x)) = \bigvee_{c \in \mathcal{C}} \rho_s(hasIngrnts(c))$

Progression of atomic and general preference formulae is defined in a straightforward fashion by progressing the individual BDFs that comprise these more expressive formulae. The full definition can be found in [3]. Note that progression can lead to a potentially exponential increase in the size of a BDF. In practice, we can (and do) greatly reduce the size of progressed formulas by the use of Boolean simplification and bounded quantification, cf. [1]. Definition 15 show us how to progress a preference formula one step, through one situation. We extend this to the notion of iterated progression.

Definition 16 (Iterated Progression). The iterated progression of a preference formula Φ through situation $s = do(\vec{a}, S_0)$, written $\rho^*_s(\Phi)$, is defined by:

$$\rho^*_{S_0}(\Phi) = \rho_{S_0}(\Phi)$$
$$\rho^*_{do(a,s)}(\Phi) = \rho_{do(a,s)}(\rho^*_s(\Phi))$$

Finally we prove that the progression of our preference formulae preserves their semantics, i.e., that our action theory entails a preference formula over the situation history of s iff it entails the progressed formula up to (but not including) s. We will exploit this in proving the correctness of our algorithm in the section to follow.

Theorem 1 (Correctness of Progression). Let $s = do([a_1, \ldots, a_n], S_0)$ be a situation and let φ be a BDF. Then

$$\mathcal{D} \models \varphi[S_0, s] \quad \text{iff} \quad \mathcal{D} \models \rho^*_{s'}(\varphi)[s, s]$$

where $s = do(a_n, s')$.

Proof Sketch: The proof proceeds by induction on the structure of φ. Refer to [3] for details.

From Theorem 1, we can prove that the weight of a situation with respect to a preference formula is equal to the weight of the final situation, disregarding its history, with respect to the progressed preference formula.

Corollary. Let $s = do([a_1, \ldots, a_n], S_0)$ be a situation and let Φ be a preference formula. Then

$$w_s(\Phi) = w_{s,s}(\rho^*_{s'}(\Phi))$$

where $s = do(a_n, s')$.

5 Computing Preferred Plans

In this section, we describe PPLAN a bounded best-first search planner for computing preference-based plans. The PPLAN algorithm is outlined in Figure 1. The code is available at [2] and a more detailed description is in [3].

```
PPLAN(init, goal, pref, maxLength, desiredWt)
frontier ← INITFRONTIER(init, pref)
bestPlanSoFar ← [ ]
bestWtSoFar ← MAXWT(pref)+1
while frontier $\neq \emptyset$ and bestWtSoFar $>$ desiredWt
    current ← REMOVEFIRST(frontier)
    if goal $\subset$ state and weight $<$ bestWtSoFar
        bestPlanSoFar ← partialPlan
        bestWtSoFar ← weight
    end if
    neighbours ← EXPAND(partialPlan, state, progPref)
    frontier ← SORTNMERGEBYVAL(neighbours, frontier)
end while
return bestPlanSoFar, bestWtSoFar
```

EXPAND(*partialPlan*, *state*, *progPref*) returns a list of new nodes to add to the frontier. If *partialPlan* has length equal to *maxLength*, EXPAND returns [ ]. Otherwise, EXPAND determines all the executable actions in *state* and returns a list which contains, for each of these executable actions a,
(*weight*, *newPartialPlan*, *newState*, *newProgPref*).

Figure 1: The PPLAN algorithm.

PPLAN takes as input an initial state *init*, a goal state *goal*, a preference formula *pref*, a length bound on plans *maxLength*, and a bound on plan weight *desiredWt*. The latter is a measure of plan quality, designating the maximum weight for which a plan is considered acceptable. The algorithm returns two outputs: a plan *bestPlanSoFar* and its weight *bestWtSoFar*.

A naive implementation of such a planner would require computing alternative plan trajectories and then evaluating their relative weights. This is computationally explosive, requiring computation of numerous plan trajectories, caching of relevant trajectory state, and redundant evaluation of preference formula weights. Instead, we make use of Theorem 1 to compute weights as we construct plans, progressing the preference formula as we go. Exploiting progressions enables the development of a best-first search strategy that orders search by weight and evaluates preference formulae across shared partial plans. Progression is commonly used to evaluate domain control knowledge in forward chaining planners such as TLPlan [1] and TALPlan [8], where progression of hard constraints prunes the search space. In contrast, we are unable to prune less preferred partial plans, because they may yield the final solution, hence the need for a best-first strategy.

Returning to our algorithm in Figure 1, our plan *frontier* is a list of nodes of the form [*weight*, *partialPlan*, *state*, *pref*], sorted by weight, and then by length. The frontier is initialized to the empty partial plan, its *weight* and *pref* corresponding to the progression and evaluation of the preference formula in the initial state. On each iteration of the **while** loop, PPLAN removes the first node from the frontier and places it in *current*. If the partial plan of *current* satisfies the goal and has lower weight than *bestWtSoFar*, then *bestPlanSoFar*

[4]We assume a finite domain. $t^{c/v}$ denotes the result of substituting the constant c for all instances of the variable v in t.

TEST #	PPLAN	BFS		TEST #	PPLAN	BFS
1	6	9		13	54	87
2	13	19		14	57	90
3	6	10		15	39	102
4	6	9		16	57	90
5	34	34		17	60	42
6	48	50		18	60	42
7	8	12		19	70	87
8	57	90		20	316	FAILS
9	38	113		21	70	4806
10	47	113		22	117	FAILS
11	47	124		23	31	1698
12	55	135		24	274	FAILS

Figure 2: Nodes expanded by **PPLAN** & breadth-first search.

and *bestWtSoFar* are replaced by *current's partialPlan* and *weight* respectively. Next we call the function **EXPAND** with *current's* node arguments as input. If *partialPlan* has length equal to *maxLength*, then the frontier is left as is. Otherwise, **EXPAND** generates a new set of nodes, one for each action executable in *state*. These new nodes are sorted by *weight*, then length and merged with the remainder of the frontier. We exit the **while** loop when we have either reached an empty frontier or we have found a plan with weight less than or equal *desiredWt*. The correctness of **PPLAN** is given in the following theorem.

Theorem 2 (Correctness of PPLAN Algorithm). Given as input a preference-based planning problem $\langle \mathcal{D}, G, \Phi \rangle$, a length bound k, and a quality bound q, **PPLAN** outputs a plan which is either k-optimal or q-satisfactory, provided $\langle \mathcal{D}, G, \Phi \rangle$ is k-solvable, and the empty plan otherwise.

Proof Sketch: The proof proceeds by proving termination and then proving the correct output properties by cases [3].

5.1 Experimental Results

We tested **PPLAN** on 24 instances of the dinner example and 31 instances of the simpler school travel example used in [13][5]. We compared the number of nodes expanded using **PPLAN**'s heuristic best-first search with a breadth-first search (BFS) algorithm. Results for the dinner example are given in Figure 2. Our results illustrate the effectiveness of our preference-weight heuristic in guiding search. As plans grow in length, the efficacy of this heuristic is magnified. It's interesting to note test cases 17 and 18, where **PPLAN** demonstrates poorer performance than BFS. Recall that **PPLAN**'s best-first search explores plans based on weight, *then* length. As a consequence, **PPLAN** can be led astray, investigating a long plan with low weight, whereas the best plan can end up being a shorter plan with higher weight. In our experience with both domains, this behavior occurs infrequently, and the heuristic generally leads to significantly improved performance.

[5]The code, domains and test cases can be found at [2]. Unfortunately, there was no way to get comparative statistics with [13].

6 Summary

In this paper we addressed the problem of preference-based planning. We presented the syntax and semantics of an expressive first-order language for specifying domain-specific, qualitative user preferences. This led to the development of **PPLAN** a best-first search, forward-chaining planner that computes optimal preferred plans relative to quality and length bounds. Our planner can be modified to compute the optimal plan without a quality bound and is trivially extended to include hard user constraints. **PPLAN** is well-suited to online planning and execution, a topic of future work. More generally, our preference language is amenable to integration with a variety of existing planners, and beyond planning, can be used to support arbitrary dynamical reasoning tasks.

Acknowledgments

The authors gratefully acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) Grant # 229717-04. The first author would also like to thank NSERC for the support of a USRA.

References

[1] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 16:123–191, 2000.

[2] M. Bienvenu and S. McIlraith. PPLAN: Code, experiments. http://www.cs.toronto.edu/∼sheila/pplan.

[3] M. Bienvenu and S. McIlraith. Planning with preferences. Manuscript, 2004.

[4] C. Boutilier, R. Brafman, C. Domshlak, H. Hoos, and D. Poole. CP-nets: A tool for representing and reasoning about conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, 21:135–191, 2004.

[5] G. Brewka. Complex preferences for answer set optimization. In D. Dubois, C. Welty, and M. Williams, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004)*, pages 213–223. AAAI Press, 2004.

[6] J. Delgrande, T. Schaub, and H. Tompits. Domain-specific preferences for causual reasoning and planning. In *Proc. KR2004*, pages 673–682, 2004.

[7] A. Gabaldon. Precondition control and the progression algorithm. In *Proc. KR2004*, pages 634–643, 2004.

[8] J. Kvarnström and P. Doherty. TALplanner: A temporal logic based forward chaining planner. *Annals of Mathematics and Artificial Intelligence*, 30:119–169, 2000.

[9] K. Myers and T. Lee. Generating qualitatively different plans through metatheoretic biases. In *Proc. AAAI'99*, pages 570–576, 1999.

[10] M. Puterman. *Markov Decision Processes: Discrete Dynamic Programming*. Wiley, New York, 1994.

[11] R. Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. 2001.

[12] F. Rossi, K. Venable, and N. Yorke-Smith. Temporal reasoning with preferences and uncertainty. In *Proc. IJCAI-03*, 2003.

[13] T. Son and E. Pontelli. Planning with preferences using logic programming. In *Proc. LPNMR 2004*, pages 247–260. 2004.