Planning with Qualitative Temporal Preferences

Meghyn Bienvenu*
IRIT
Université Paul Sabatier
Toulouse, France
meghyn.bienvenu@irit.fr

Christian Fritz
Department of Computer Science
University of Toronto
Toronto, Canada
fritz@cs.toronto.edu

Sheila A. McIlraith
Department of Computer Science
University of Toronto
Toronto, Canada
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 non-Markovian preferences over the evolution of states and actions associated with a plan. The semantics for our first-order preference language is defined in the situation calculus. Unlike other recent temporal preference languages, our preferences are *qualitative* rather than just *ordinal*, affording greater expressivity and less incomparability. We propose an approach to computing preferred plans via bounded best-first search in a forward-chaining planner. Key components of our approach are the exploitation of progression to efficiently evaluate levels of preference satisfaction over partial plans, and development of an admissible evaluation function that establishes the optimality of best-first search. We have implemented our planner **PPLAN** and evaluated it experimentally. Our preference language and planning approach is amenable to integration with several existing planners, and beyond planning, can be used to support arbitrary dynamical reasoning tasks involving preferences.

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, 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 (Myers & Lee 1999).

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 non-Markovian preferences over the evolution of actions and states leading to goal achievement. Our language harnesses much of the expressive power of first-order and linear temporal logic (LTL). We define the semantics of our preference language in the situation calculus (Reiter 2001). Nevertheless, nothing requires that the planner be implemented in the situation calculus. Indeed our planner **PPLAN**, a bounded best-first search planner, is a forward-chaining planner, in the spirit of TLPlan (Bacchus & Kabanza 2000) and TALPlan (Kvarnström & Doherty 2000), that exploits progression of preference formulae to more efficiently compute preferred plans. Experimental results illustrate the efficacy of our best-first heuristic.

Research on qualitative preferences has predominantly focused on less expressive, *static* preferences (e.g., (Boutilier *et al.* 2004)). In the area of dynamic preferences, Son and Pontelli (Son & Pontelli 2004) have developed a language for planning with preferences together with an implementation using answer-set programming. Also notable is the work of Delgrande et al. (Delgrande, Schaub, & Tompits 2004), who have developed a framework for characterizing preferences and properties of preference-based planning. Research on decision-theoretic planning and MDPs also addresses the problem of generating preferred plans (Puterman 1994). Nevertheless, the elicitation of preferences in terms of Markovian numeric utilities makes these approaches less applicable to the types of preferences we examine. We include a comprehensive discussion of related work at the end of the paper.

We begin in Section 2 with 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 via an example which is carried through the paper. With the semantics of our preferences in hand, we return in Section 4 to the general problem of planning with preferences. Next, in Section 5, we introduce progression which we prove preserves our semantics, and we define an admissible best-first heuristic. In Section 6, we describe our implementation of **PPLAN**, proving the correctness of our algorithm and presenting our experimental results. We conclude the paper with a discussion of related work and a summary.

2 Preliminaries

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., $F(\vec{x}, s)$. In this paper, we distin-

*This work was performed while the first author was a student at the University of Toronto.

guish 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 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 under which 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 (Reiter 2001). 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* 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 $s = do(a_n, \ldots, do(a_1, S_0)))$[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 situation $s = do(\vec{a}, S_0)$ as the *plan trajectory* and the sequence of actions $\vec{a} = a_1 a_2 ... a_n$ as the *associated plan*. The *length* of this plan is n. The set of all plans is denoted by Π, and Π^k denotes the subset of plans of length $\leq k$. 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 $\leq k$.

3 Preference Specification

In this section we describe the syntax and semantics of our first-order preference language, illustrated in terms of the following 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 the 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 in (Son & Pontelli 2004). We keep their hierarchy of basic desire formulae, atomic preference formulae, and general preference formulae, to which we add a new class of aggregated preference formulae. Subsequent reference to a *preference formula* refers to an aggregated preference formula, which encompasses basic desire formulae, atomic preference formulae, and general 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. $f \in \mathcal{F}$, then **final**$(f) \in \mathcal{B}$
4. If $a \in \mathcal{A}$, then **occ**$(a) \in \mathcal{B}$
5. If φ_1 and φ_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**(φ_1), **always**(φ_1), **eventually**(φ_1), and **until**(φ_1, φ_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 BDFs from our dinner example.

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

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 cook 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 Claire cooked something, ordered something from take-out, or ordered something at a restaurant. Finally P7 tells us that Claire never eats any chinese food.

While BDFs alone enable us to express some simple user preferences, we cannot express preferences between alternatives. For example, we cannot say that Claire prefers

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

cooking to ordering take-out. To do so, we define Atomic Preference Formulae, which represent preferences over alternatives. Our definition differs from that in (Son & Pontelli 2004) in that we ask the user to annotate the component preferences with a value indicating the level of preference. What we gain in doing so is significantly reduced incomparability, a wider range of preference combination methods, and a more fine-grained representation of the user's desires.

Definition 2 (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 φ_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.

An atomic preference formula expresses a preference over alternatives. In what follows, we let $\mathcal{V} = [0, 1]$ for parsimony, but we could just as easily choose a strictly qualitative set like $\{$*best* $<$ *good* $<$ *indifferent* $<$ *bad* $<$ *worst*$\}$. Returning to our example, the following APF expresses Claire's preference over what to eat (pizza, followed by spaghetti, followed by crêpes):

$$\begin{array}{l}\textbf{eventually}(\textbf{occ}(eat(pizza)))[0] \gg \\ \quad \textbf{eventually}(\textbf{occ}(eat(spag)))[0.4] \gg \\ \quad\quad \textbf{eventually}(\textbf{occ}(eat(crêpes)))[0.5]\end{array} \quad \text{(P8)}$$

We can see that Claire strongly prefers pizza but finds spaghetti and crêpes about equally good. If instead Claire is more concerned about how long she will have to wait for her meal, she may specify the following APF:

$$P5[0] \gg (P2 \wedge P4)[0.2] \gg P6[0.7] \gg (\neg P2 \wedge P4)[0.9] \quad \text{(P9)}$$

This says that Claire's 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. From the values that Claire provided, we can see that she really prefers options that don't involve going out.

Again, an atomic preference represents a preference over alternatives φ_i. We wish to satisfy the BDF φ_i with the lowest index i. Consequently, if Claire eats pizza *and* crêpes, this is no better nor worse with respect to P8 than situations in which Claire eats only pizza, and it is strictly better than situations in which she just eats crêpes. 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.

In order to allow the user to specify more complex preferences, we introduce our third class of preference formulae, which extends our language to conditional, conjunctive, and disjunctive preferences.

Definition 3 (General Preference Formula (GPF)). 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 BDF and Ψ is a general preference formula [Conditional]
- Φ is one of
  - $\Psi_0 \,\&\, \Psi_1 \,\&\, ... \,\&\, \Psi_n$ [General Conjunction]
  - $\Psi_0 \mid \Psi_1 \mid ... \mid \Psi_n$ [General Disjunction]

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

Here are some example general preference formulae:

$$P2 : P4\,(P10) \qquad P8 \mid P9\,(P11) \qquad P8 \,\&\, P9\,(P12)$$

P10 states that if Claire initially has the ingredients for something she can make, then she prefers to cook. Preferences P12 and P11 show two ways we can combine Claire's food and time preferences into a single complex preference. P12 tries to maximize the satisfaction of both of her preferences, whereas P11 is appropriate if she would be content if either of the two were satisfied.

We have just seen two ways that a user's preferences can be combined. Indeed, because users very often have more than a single preference, we feel that it is important to provide a variety of methods of combining different preferences. Moreover, since a user's preferences may be unsatisfiable, our preference language should allow the user to indicate how to relax his preferences in order to find a solution. With this in mind, we introduce our final class of preference formulae which proposes a number of preference aggregation methods.

Definition 4 (Aggregated Preference Formula (AgPF)). A formula Φ is an aggregated preference formula if: 1) Φ is a general preference, or 2) for general preference formulae $\Psi_i, i = 1, \ldots, n$, Φ is one of **lex**$(\Psi_1, ..., \Psi_n)$, **lexand**$(\Psi_1, ..., \Psi_n)$, **lexor**$(\Psi_1, ..., \Psi_n)$, **leximin**$(\Psi_1, ..., \Psi_n)$, or, if $\mathcal{V}$ is numeric, **sum**$(\Psi_1, ..., \Psi_n)$.

We briefly discuss these different aggregation methods, many of which are already well-known in the literature (cf., (Ehrgott 2000)). Preference formulae **lex**$(\Psi_1, ..., \Psi_n)$ rank situations using the standard lexicographic ordering, a combination method which is appropriate when the user's preferences are of greatly different levels of importance. **lexand**$(\Psi_1, ..., \Psi_n)$ (resp. **lexor**$(\Psi_1, ..., \Psi_n)$) refines the ordering defined by $\Psi_0 \,\&\, ... \,\&\, \Psi_n$ (resp. $\Psi_0 \mid ... \mid \Psi_n$) by sorting indistinguishable situations using the lexicograping ordering. For example, **lexand**(Ψ_1, Ψ_2) will try to satisfy both Ψ_1 and Ψ_2, but failing this will prefer to satisfy Ψ_1 over Ψ_2. This is useful in cases where we have preferences of similar yet unequal importance. The aggregation method **leximin** first sorts the levels of satisfaction of the component preferences in non-decreasing order and then applies the lexicographic ordering, a method well-suited to cases where the component preferences are considered equally important. Finally, if $\mathcal{V}$ is numeric, we can simply **sum** the levels of satisfaction of the component preferences, which amounts to maximizing the average level of satisfaction.

This concludes our description of the syntax of our preference language. Our language extends and modifies the *PP* language recently proposed in (Son & Pontelli 2004). Quantifiers, variables, non-fluent relations, a conditional construct, and aggregation operators (AgPF) have been added to our language. In *PP* it is impossible to talk about arbitrary action or fluent arguments or their properties, and difficult or even impossible to express the kinds of preferences given above. *PP*'s APFs are *ordinal* rather than *qualitative* making relative differences between ordered preferences impossible to articulate. Finally, our semantics gives a different, and we argue more natural, interpretation of *General Conjunction*

and *General Disjunction*. Relative to *quantitative* dynamical preferences, we argue that our language is more natural for a user. We elaborate on this point in Section 7.

3.2 The Semantics of our Language

We appeal to the situation calculus to define the semantics of our preference language. BDFs are interpreted as situation calculus formulae and are evaluated relative to an action theory $\mathcal{D}$. In order to define the semantics of more complex preference formulae, which can be satisfied to a certain degree, we associate a qualitative value or *weight* with a situation term, depending upon how well it satisfies a preference formula. Weights are elements of $\mathcal{V}$, with v_{min} indicating complete satisfaction and v_{max} complete dissatisfaction. The motivation for introducing values was that purely ordinal preferences (such as the atomic preference formulae in (Son & Pontelli 2004)) can be combined in only very limited, and not necessarily very natural ways, in addition to leading to great incomparability between outcomes. Replacing ordinal preferences by qualitative preferences allows us to give a more nuanced representation of the user's preferences.

Since BDFs may refer to properties that hold at various situations in a situation history, we use the notation $\varphi[s, s']$ borrowed from (Gabaldon 2004) 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 $F[s]$ denotes the re-insertion of situation term s.

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:

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

The BDF **occ**(a) tells us that the first action executed is a, which can be written as:

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

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 (Gabaldon 2004):[2]

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

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

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

Definition 5 (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 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. $w_{s',s}(\varphi) = v_{min}$ if φ is satisfied, otherwise $w_{s',s}(\varphi) = v_{max}$.

Note that in the special case of $s' = S_0$, we drop s' from the index, i.e. $w_s(\varphi) = w_{S_0,s}(\varphi)$. For readibility, the following definitions are phrased in terms of $w_s(\varphi)$ but can be analogously formulated for the more general case.

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

$w_s(P1) = 1 \quad w_s(P2) = 0 \quad w_s(P3) = 0 \quad w_s(P4) = 0$
$w_s(P5) = 1 \quad w_s(P6) = 1 \quad w_s(P7) = 0$

The weight of an atomic preference formula is simply defined to be the value associated with the first satisfied component BDF:

Definition 6 (Atomic Preference Satisfaction). Let s be a situation and $\Phi = \varphi_0[v_0] \gg \varphi_1[v_1] \gg \ldots \gg \varphi_n[v_n]$ 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.

Evaluation with respect to Example 1 gives $w_s(P8) = 0.5$ and $w_s(P9) = 0.2$.

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:

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

Observe that the interpretations of our generalized boolean connectives emulate their boolean counterparts. Returning to Example 1:

- $w_s(P2 : P4) = w_s(P4) = 0$
- $w_s(P8 \,\&\, P9) = \max\{0.5, 0.2\} = 0.5$
- $w_s(P8 \mid P9) = \min\{0.5, 0.2\} = 0.2$

We conclude this section with the following definition which shows us how to compare two situations with respect to an aggregated preference formula:

Definition 8 (Preferred Situations). A situation s_1 is at least as preferred as a situation s_2 with respect to a preference formula Φ, written $s_1 \succeq_\Phi s_2$, if one of the following holds:

- Φ is a GPF, and $w_{s_1}(\Phi) \leq w_{s_2}(\Phi)$
- $\Phi = \textbf{lex}(\Phi_1, \ldots, \Phi_n)$ and either $w_{s_1}(\Phi_i) = w_{s_2}(\Phi_i)$ for all i or there is some i such that $w_{s_1}(\Phi_i) < w_{s_2}(\Phi_i)$ and for all $j < i$ $w_{s_1}(\Phi_j) = w_{s_2}(\Phi_j)$
- $\Phi = \textbf{lexand}(\Phi_1, \ldots, \Phi_n)$ and either $s_1 \succ_{\Psi_0 \,\&\, \ldots \,\&\, \Psi_n} s_2$ or $s_1 \approx_{\Psi_0 \,\&\, \ldots \,\&\, \Psi_n} s_2$ and $s_1 \succeq_{\textbf{lex}(\Phi_1, \ldots, \Phi_n)} s_2$

- $\Phi = \textbf{lexor}(\Phi_1, ..., \Phi_n)$ and either $s_1 \succ_{\Psi_0 \mid ... \mid \Psi_n} s_2$ or $s_1 \approx_{\Psi_0 \mid ... \mid \Psi_n} s_2$ and $s_1 \succeq_{\textbf{lex}(\Phi_1,...,\Phi_n)} s_2$
- $\Phi = \textbf{leximin}(\Phi_1, ..., \Phi_n)$ and either $|\{i : w_{s_1}(\Phi_i) = v\}| = |\{i : w_{s_2}(\Phi_i) = v\}|$ for all $v \in \mathcal{V}$ or there is some v such that $|\{i : w_{s_1}(\Phi_i) = v\}| > |\{i : w_{s_2}(\Phi_i) = v\}|$ and for all $v' < v$ $|\{i : w_{s_1}(\Phi_i) = v'\}| = |\{i : w_{s_2}(\Phi_i) = v'\}|$
- $\Phi = \textbf{sum}(\Phi_1, ..., \Phi_n)$, and $\sum_{i=1}^{n} w_{s_1}(\Phi_i) \leq \sum_{i=1}^{n} w_{s_1}$

Strict preference ($\succ$) and equivalence ($\approx$) are defined in the standard way.

We remark that the relation $\succeq_\Phi$ defines a total preorder over situations. Thus, any two situations (and hence plans) can be compared in our framework. We also observe that, at least for finite $\mathcal{V}$, our preference relation is definable within the situation calculus language, enabling us to reason about preferences *within* the language.

4 Planning with Preferences

With our 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 with temporally extended goals (e.g., (Kvarnström & Doherty 2000; Bacchus & Kabanza 2000)) by encoding the goal as a BDF Φ_c. A plan is then any $\vec{a}$ such that $\mathcal{D} \models \Phi_c[S_0, do(\vec{a}, S_0)] \wedge executable(do(\vec{a}, S_0))$.

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

Definition 11 (Optimal Plan, k-Optimal Plan). Given a preference-based planning problem, an optimal plan is any plan $\vec{a}$ such that $\nexists \vec{b}.do(\vec{b}, S_0) \succ_\Phi do(\vec{a}, S_0)$. A k-optimal plan is any plan $\vec{a} \in \Pi^k$ such that $\nexists \vec{b} \in \Pi^k.do(\vec{b}, S_0) \succ_\Phi do(\vec{a}, S_0)$.

5 Computing Preferred Plans

Our approach to computing preferred plans is to construct a bounded best-first search forward-chaining planner that takes as input an initial state, a goal, a general preference formula Φ,[3] and a length bound, and outputs an optimal preferred plan relative to that bound. Two key components of our approach are: 1) the use of *progression* to efficiently evaluate how well partial plans satisfy Φ, and 2) the definition of an admissible *evaluation function* that ensures our best-first search is optimal.

5.1 Progression

Progression has been used extensively to evaluate domain control knowledge in forward chaining planners such as TLPlan (Bacchus & Kabanza 2000) and TALPlan (Kvarnström & Doherty 2000), where progression of hard constraints prunes the search space. We use it here to efficiently evaluate our preference formula over candidate partial plans.

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 progression preserves the semantics of preference formulae.

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

Definition 12 (Progression of a BDF). Let s be a situation, and let φ be a BDF. 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 $\varphi = \textbf{occ}(a)$, then $\rho_s(\varphi) = \textbf{occNext}(a)$
- If $\varphi = \textbf{occNext}(a)$, 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 $\varphi = \textbf{next}(\psi)$, then $\rho_s(\varphi) = \psi$
- If $\varphi = \textbf{always}(\psi)$, 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 $\varphi = \neg\psi$, 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 $\varphi = \textbf{final}(\psi)$, then $\rho_s(\varphi) = \varphi$
- If $\varphi =$ TRUE or $\varphi =$ FALSE, then $\rho_s(\varphi) = \varphi$

Returning to Example 1,

$$
\begin{aligned}
&- \rho_s(\textbf{always}(kitchenClean)) \\
&\quad = \rho_s(kitchenClean) \wedge \textbf{always}(kitchenClean) \\
&\quad = \text{FALSE} \\
&- \rho_s((\exists x).hasIngrnts(x)) = \bigvee_{c \in \mathcal{C}} \rho_s(hasIngrnts(c))
\end{aligned}
$$

Progression of APFs and GPFs is defined in a straightforward fashion by progressing their constituent BDFs[5].

Definition 13 (Progression of General Preference Formulae). Let s be a situation, and let Φ be a general preference formula. The progression of Φ up to s is defined by:

[3] For simplicity, we restrict our attention to GPFs, but the algorithm can be trivially extended to AgPFs.

[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. Note that progression of quantified BDFs can result in a significant increase in formula size. In practice, we can (and do) use Boolean simplification and bounded quantification to reduce the size of progressed formulae, cf. (Bacchus & Kabanza 2000).

[5] This is also the case for AgPFs.

- $\rho_s(\varphi_0 \gg \ldots \gg \varphi_n) = \rho_s(\varphi_0) \gg ... \gg \rho_s(\varphi_n)$
- $\rho_s(\gamma : \Psi) = \rho_s(\gamma) : \rho_s(\Psi)$
- $\rho_s(\Psi_0 \,\&\, ... \,\&\, \Psi_n) = \rho_s(\Psi_0) \,\&\, \ldots \,\&\, \rho_s(\Psi_n)$
- $\rho_s(\Psi_0 \mid ... \mid \Psi_n) = \rho_s(\Psi_0) \mid \ldots \mid \rho_s(\Psi_n)$

Definitions 12 and 13 show us how to progress a preference formula one step, through one situation. We extend this to the notion of iterated progression.

Definition 14 (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:

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

To prove our progession theorem, we will make use of a more general form of iterated progression, which takes two situation arguments:

Definition 15 (General Iterated Progression). The iterated progression of a preference formula Φ starting from situation s_1 through situation $s_2 = do(\vec{a}, s_1)$, written $\rho^*_{s_1,s_2}(\Phi)$, is defined by:

$$\rho^*_{s_1,\,s_1}(\Phi) = \rho_{s_1}(\Phi)$$
$$\rho^*_{s_1,\,do(a,\,s_3)}(\Phi) = \rho_{do(a,\,s_3)}(\rho^*_{s_1,\,s_3}(\Phi))$$

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.

Theorem 1 (Correctness of Progression). Let s_1 and $s_2 = do([a_1, \ldots, a_n], s_2)$ be two situations where $n \geq 1$, and let φ be a BDF. Then

$$\mathcal{D} \models \varphi[s_1,\, s_2] \quad \text{iff} \quad \mathcal{D} \models \rho^*_{s_1,\,s_3}(\varphi)[s_2,\, s_2]$$

where $s_2 = do(a_n,\, s_3)$.

Proof Sketch: The proof proceeds by induction on the structural complexity of ϕ. For space considerations, we do not give all of the cases here. Instead, we give a few representative examples. See (Bienvenu & McIlraith 2006) for more details.

Case 1: $\phi = f \in \mathcal{F}$
$\mathcal{D} \models f[s_1,\, s_2]$
iff $\mathcal{D} \models f(s_1)$
iff $\rho_{s_1}(f) = \text{TRUE}$
iff $\rho^*_{s_1,\,s_3}(f) = \text{TRUE}$
iff $\mathcal{D} \models \rho^*_{s_1,\,s_3}(f)[s_2,\, s_2]$

Case 7: $\phi = \psi_1 \wedge \psi_2$ for BDFs ψ_1 and ψ_2
We assume the result for ψ_1 and ψ_2 and show that the result holds for $\psi_1 \wedge \psi_2$.
$\mathcal{D} \models \psi_1 \wedge \psi_2[s_1,\, s_2]$
iff $\mathcal{D} \models \psi_1[s_1,\, s_2]$ and $\mathcal{D} \models \psi_2[s_1,\, s_2]$
iff $\mathcal{D} \models \rho^*_{s_1,\,s_3}(\psi_1)[s_2,\, s_2]$ and $\mathcal{D} \models \rho^*_{s_1,\,s_3}(\psi_2)[s_2,\, s_2]$
iff $\mathcal{D} \models \rho^*_{s_1,\,s_3}(\psi_1) \wedge \rho^*_{s_1,\,s_3}(\psi_2)[s_2,\, s_2]$
iff $\mathcal{D} \models \rho^*_{s_1,\,s_3}(\psi_1 \wedge \psi_2)[s_2,\, s_2]$

Case 12: $\phi = \textbf{always}(\psi)$
We assume the result for ψ and prove that the result also holds for $\textbf{always}(\psi)$. The proof proceeds by induction on n, the difference in length between s_1 and s_2, with base case $n = 1$:
$\mathcal{D} \models \textbf{always}(\psi)[s_1,\, do(a_1,\, s_1)]$
iff $\mathcal{D} \models \psi[s_1,\, do(a_1,\, s_1)] \wedge \psi[do(a_1,\, s_1),\, do(a_1,\, s_1)]$
iff $\mathcal{D} \models \rho^*_{s_1,\,s_1}(\psi)[do(a_1,\, s_1),\, do(a_1,\, s_1)]$
$\wedge\ \textbf{always}(\psi)[do(a_1,\, s_1),\, do(a_1,\, s_1)]$
iff $\mathcal{D} \models \rho_{s_1}(\psi) \wedge \textbf{always}(\psi)[do(a_1,\, s_1),\, do(a_1,\, s_1)]$
iff $\mathcal{D} \models \rho^*_{s_1,\,s_1}(\textbf{always}(\psi))[do(a_1,\, s_1),\, do(a_1,\, s_1)]$
We now suppose that the theorem holds for $n < k$, and we show that it is also true when $s_2 = do([a_1, \ldots, a_k], s_1)$:
$\mathcal{D} \models \textbf{always}(\psi)[s_1,\, s_2]$
iff $\mathcal{D} \models \psi[s_1,\, s_2] \wedge \textbf{always}(\psi)[do(a_1,\, s_1),\, s_2]$
iff $\mathcal{D} \models \rho^*_{s_1,\,s_3}(\psi)[s_2,\, s_2] \wedge$
$\rho^*_{do(a_1,\,s_1),\,s_3}(\textbf{always}(\phi))[s_2,\, s_2]$
iff $\mathcal{D} \models \rho_{s_3}(\ldots \rho_{do(a_1,s_1)}(\rho_{s_1}(\psi) \wedge \textbf{always}(\psi)) \ldots)[s_2,\, s_2]$
iff $\mathcal{D} \models \rho_{s_3}(\ldots \rho_{do(a_1,s_1)}(\rho_{s_1}(\textbf{always}(\psi))) \ldots)[s_2,\, s_2]$
iff $\mathcal{D} \models \rho^*_{s_1,\,s_3}(\textbf{always}(\psi))[s_2,\, s_2]$ □

Using Theorem 1, we can prove that the weight of a general preference formula with respect to a situation (plan trajectory) is equal to the weight of the progressed preference formula with respect to the final situation, disregarding its history.

Corollary. Let $s = do([a_1, \ldots, a_n], S_0)$ be a situation and let Φ be a general preference formula. Then $w_{S_0,s}(\Phi) = w_{s,s}(\rho^*_{s'}(\Phi))$ where $s = do(a_n, s')$.

5.2 An Evaluation Function for Best-First Search

In this section, we propose an admissible evaluation function for best-first search. To this end, we introduce the notion of *optimistic* and *pessimistic* weights of a situation relative to a GPF Φ. These weights provide a bound on the best and worst weights of any successor situation with respect to Φ. As a result, our evaluation function is non-decreasing and will never over-estimate the actual weight, thus enabling us to define an optimal search algorithm.

Optimistic (respectively pessimistic) weights are defined based on optimistic (respectively pessimistic) satisfaction of BDFs. Roughly, 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, namely that anything not yet satisfied will eventually be falsified. The definition of optimistic and pessimistic satisfaction largely follows the definition of (normal) satisfaction of BDFs given earlier. The key difference is in the definition of $\textbf{next}(\phi)$ and $\textbf{occ}(a)$:

$$\textbf{final}(\varphi)[s', s]^{opt} \stackrel{\text{def}}{=} \text{TRUE}$$
$$\textbf{final}(\varphi)[s', s]^{pess} \stackrel{\text{def}}{=} \text{FALSE}$$
$$\textbf{occ}(a)[s', s]^{opt} \stackrel{\text{def}}{=} do(a, s') \sqsubseteq s \vee s' = s$$
$$\textbf{occ}(a)[s', s]^{pess} \stackrel{\text{def}}{=} do(a, s') \sqsubseteq s^{6}$$
$$\textbf{next}(\varphi)[s', s]^{opt} \stackrel{\text{def}}{=} (\exists a).do(a, s') \sqsubseteq s \wedge \varphi[do(a, s'), s]^{opt} \vee s' = s$$
$$\textbf{next}(\varphi)[s', s]^{pess} \stackrel{\text{def}}{=} (\exists a).do(a, s') \sqsubseteq s \wedge \varphi[do(a, s'), s]^{pess}$$

We define the other temporal formulae in terms of **next**.

$$\textbf{eventually}(\varphi)[s',s]^{opt/pess} \stackrel{\text{def}}{=} \varphi[s',s]^{opt/pess} \vee \textbf{next}(\textbf{eventually}(\varphi))[s',s]^{opt/pess}$$

$$\textbf{always}(\varphi)[s',s]^{opt/pess} \stackrel{\text{def}}{=} \varphi[s',s]^{opt/pess} \wedge \textbf{next}(\textbf{always}(\varphi))[s',s]^{opt/pess}$$

$$\textbf{until}(\varphi,\psi)[s',s]^{opt/pess} \stackrel{\text{def}}{=} \psi[s',s]^{opt/pess} \vee (\varphi[s',s]^{opt/pess} \wedge \textbf{next}(\textbf{until}(\varphi,\psi))[s',s]^{opt/pess})$$

For the purpose of creating an admissible evaluation function for planning, we are really only interested in optimistic evaluation. The reason why we also need pessimistic evaluation is simple: the BDF $\neg\varphi$ is optimistically satisfied if and only if φ is not pessimistically satisfied. That is, it is optimistic to assume that there is a way to falsify φ which in turn will satisfy the negation. We thus define:

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

For all other elements of the language, the definitions are the same as for normal BDF satisfaction.

We can now define optimistic and pessimistic weights of BDFs in terms of optimistic and pessimistic BDF satisfaction:

$$w^{opt}_{[s',s]}(\varphi) = \begin{cases} v_{min} & \text{if } \mathcal{D} \models \phi[s',s]^{opt} \\ v_{max} & \text{otherwise} \end{cases}$$

and

$$w^{pess}_{[s',s]}(\varphi) = \begin{cases} v_{min} & \text{if } \mathcal{D} \models \phi[s',s]^{pess} \\ v_{max} & \text{otherwise} \end{cases}$$

For APFs and GPFs the definitions of optimistic and pessimistic weights are straightforward.

For readability, we abbreviate $w^{opt}_{[S_0,s]}$ and $w^{pess}_{[S_0,s]}$ by w^{opt}_s and w^{pess}_s respectively.

Definition 16 (Optimistic/Pessimistic 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

$$w^{opt/pess}_s(\Phi) = \begin{cases} v_i & \text{if } i = \min_j\{\mathcal{D} \models \varphi_j[S_0,s]^{opt/pess}\} \\ v_{max} & \text{if no such } i \text{ exists.} \end{cases}$$

Definition 17 (General Preference Satisfaction). Let s be a situation and Φ be a general preference formula. Then $w^{opt}_s = (\Phi)$, respectively $w^{pess}_s = (\Phi)$, is defined as follows:

- $w^{opt/pess}_s(\varphi_0 \gg \varphi_1 \gg ... \gg \varphi_n)$ is defined above
- $w^{opt/pess}_s(\gamma : \Psi) = \begin{cases} v_{min} & \text{if } w^{opt/pess}_s(\gamma) = v_{max} \\ w^{opt/pess}_s(\Psi) & \text{otherwise} \end{cases}$
- $w^{opt/pess}_s(\Psi_0 \,\&\, \Psi_1 \,\&\, ... \,\&\, \Psi_n) = \max\{w^{opt/pess}_s(\Psi_i) : 1 \le i \le n\}$
- $w^{opt/pess}_s(\Psi_0 \mid \Psi_1 \mid ... \mid \Psi_n) = \min\{w^{opt/pess}_s(\Psi_i) : 1 \le i \le n\}$

The following theorem describes some of the important properties of our optimistic and pessimistic weight functions.

Theorem 2. Let $s_n = do([a_1,...,a_n],S_0), n \ge 0$ be a collection of situations, φ be a BDF, and Φ a general preference formula. Then for any $0 \le i \le j \le k \le n$,

1. $\mathcal{D} \models \varphi[s_i]^{pess} \Rightarrow \mathcal{D} \models \varphi[s_j]$ and $\mathcal{D} \not\models \varphi[s_i]^{opt} \Rightarrow \mathcal{D} \not\models \varphi[s_j]$,
2. $\left(w^{opt}_{s_i}(\Phi) = w^{pess}_{s_i}(\Phi)\right) \Rightarrow w_{s_j}(\Phi) = w^{opt}_{s_i}(\Phi) = w^{pess}_{s_i}(\Phi)$,
3. $w^{opt}_{s_i}(\Phi) \le w^{opt}_{s_j}(\Phi) \le w_{s_k}(\Phi)$ and $w^{pess}_{s_i}(\Phi) \ge w^{pess}_{s_j}(\Phi) \ge w_{s_k}(\Phi)$

Since these definitions are compatible with those defined for progression, we have the following corollary to Theorem 2:

Corollary. Let $s' = do([a_1,\ldots,a_{n-1}],S_0)$, $s = do(a_n,s')$ be situations, $n \ge 1$, φ a BDF. Then $\mathcal{D} \models \varphi[S_0,s]^{opt/pess}$ iff $\mathcal{D} \models \rho^*_{s'}(\varphi)[s,s]^{opt/pess}$.

This corollary states that we can still use progression when computing optimistic and pessimistic weights. Intuitively this is because the optimistic (pessimistic) part of the evaluation is only concerned with the future whereas the progression deals with the past. Since the past won't change, there is no room for optimism or pessimism.

We can now define our evaluation function f_Φ.

Definition 18 (Evaluation function). Let $s = do(\vec{a},S_0)$ be a situation and let Φ be a general preference formula. Then $f_\Phi(s)$ is defined as follows:

$$f_\Phi(s) = \begin{cases} w_s(\Phi) & \text{if } \vec{a} \text{ is a plan} \\ w^{opt}_s(\Phi) & \text{otherwise} \end{cases}$$

From Theorem 2 we see that the optimistic weight is non-decreasing and never over-estimates the real weight. Thus, f_Φ is admissible and when used in best-first search, the search is optimal.

6 Implementation

In this section, we describe **PPLAN** a bounded best-first search planner for computing preferred plans. The **PPLAN** algorithm is outlined in Figure 1. The code and test cases are available at `http://www.cs.toronto.edu/~sheila/pplan`.

PPLAN takes as input an initial state *init*, a goal state *goal*, a general preference formula *pref* [7], and a plan length bound *maxLength*. The algorithm returns two outputs: a plan and its weight with respect to *pref*.

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

[6] It follows that when $s' = s$, $occ(a)[s',s]^{pess} \stackrel{\text{def}}{=}$ FALSE and $\textbf{next}(\varphi)[s',s]^{pess} \stackrel{\text{def}}{=}$ FALSE.

[7] To treat aggregated preference formulae it suffices to associate a *tuple* of optimistic and pessimistic weights with each node and to sort the frontier according to Definition 8.

```
PPLAN(init, goal, pref, maxLength)
frontier ← INITFRONTIER(init, pref)
while frontier ≠ ∅
    current ← REMOVEFIRST(frontier)
    if goal ⊂ state and optW=pessW
        return partialPlan, optW
    end if
    neighbours ← EXPAND(partialPlan, state, progPref)
    frontier ← SORTNMERGEBYVAL(neighbours, frontier)
end while
return [], ∞
```

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 a node
(*optW*, *pessW*,*newPartialPlan*, *newState*, *newProgPref*)
and for each a leading to a goal state, a second node
(*realW*, *realW*,*newPartialPlan*, *newState*, *newProgPref*).

Figure 1: The PPLAN algorithm.

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 progression 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 (Bacchus & Kabanza 2000) and TALPlan (Kvarnström & Doherty 2000), 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 *frontier* is a list of nodes of the form [*optW*, *pessW*, *partialPlan*, *state*, *pref*], sorted by optimistic weight, pessimistic weight, and then by length. The frontier is initialized to 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, PPLAN removes the first node from the frontier and places it in *current*. If the partial plan of *current* satisfies the goal and has *optW=pessW*, then PPLAN returns *current*'s partial plan and weight. Otherwise, we call the function EXPAND with *current's* node arguments as input. If *partialPlan* has length equal to *maxLength* then no new nodes are added to the frontier. Otherwise, EXPAND generates a new set of nodes of the form [*optW*, *pessW*, *partialPlan*, *state*, *pref*], one for each action executable in *state*. For actions leading to goal states, EXPAND also generates a second node of the same form but with *optW* and *pessW* replaced by the actual weight achieved by the plan. The reason that we need two nodes is that on the one hand, we need to record the actual weight associated with the plan that we have found, and on the other hand, to ensure completeness, we need to be able to reach the node's successors. The new nodes generated by EXPAND are then sorted by *optW*, *pessW*, then length and merged with the remainder of the frontier. If we reach the empty frontier, we exit the **while** loop and return the empty plan. The correctness of PPLAN is given in the following theorem.

Theorem 3 (Correctness of PPLAN Algorithm). Given as input a preference-based planning problem $\mathcal{P}$ and a length bound k, PPLAN outputs a k-optimal plan, if $\mathcal{P}$ is k-solvable, and the empty plan otherwise.

Proof Sketch: First, we prove that the algorithm terminates. There are two ways that PPLAN halts: either the first node on the frontier is a plan and has *optW= pessW*, in which case PPLAN returns this plan, or we reach the empty frontier, in which case PPLAN returns the empty plan. Let us then suppose that the first condition is never met. In this case, we will stay in the while loop, expanding one node on each iteration. But since the successor nodes generated by EXPAND always have length one greater than their parent, and since EXPAND returns an empty list whenever a node has a partial plan of length equal to k, we will eventually run out of nodes and reach the empty frontier. Thus, the algorithm always terminates.

Next, we prove that the outputted plan satisfies the conditions of the theorem. This is obvious for the case where $\mathcal{P}$ is not k-solvable as in this case, we will never find a plan and thus will stay in the while loop until we reach the empty frontier, finally outputting the empty plan.

We now treat the case where $\mathcal{P}$ is k-solvable. By definition, this means that there exists at least one plan of length less than or equal k. As PPLAN systematically explores the search space, at some point EXPAND will create a node whose partial plan satisfies the goal and will set *optW* and *optW* to the actual weight. This means that the frontier will contain a node satisfying the conditions of the **if** loop, and hence, at some point, we will enter the **if** loop and return a non-empty plan. It remains to be shown that the plan returned is k-optimal.

Suppose for a contradiction that we return a plan p with weight w which is not k-optimal. This means that there exists a plan p' of length less than k which has a weight than $w' < w$. There are two possibilities: either we have generated a node corresponding to p' and placed it behind p on the frontier, which is contradiction as the frontier is sorted in non-decreasing order by *optW*, or there there is an ancestor node of p' which is behind p in the frontier. But this is also impossible, as according to Theorem 2, any ancestor of p' must have an optimistic weight less than or equal to $w' < w$ (and hence must be before p on the frontier). We have thus shown that if $\mathcal{P}$ is k-solvable, the outputted plan is k-optimal, concluding the proof. □

6.1 Experiments

In Figure 2, we present the results for 60 instances of our dinner domain [8]. We compared the number of nodes

[8]We also ran an early version of PPLAN on the simple school travel example presented in (Son & Pontelli 2004), but we were

expanded using **PPLAN**'s heuristic best-first search with a breadth-first search (BFS) algorithm[9]. Overall, the results are quite positive. In 56 of the 60 test cases, **PPLAN** performs at least as well as breadth-first search, and generally significantly better, often an order of magnitude so. The four cases where BFS outperforms **PPLAN** are all cases where there was a short k-optimal but non-ideal plan. In these cases, **PPLAN** quickly finds the plan, but must continue the search in order to ensure that no other better plan exists. In 2 of 60 test cases, **PPLAN** runs out of memory before finding a plan. This is most likely due to the fact that our implementation is not optimized, but it also speaks to the difficulty of the task. A good way to cope with this problem is to add control knowledge to reduce the search space. In order to test out this idea, we reran **PPLAN** on the test suite, this time pruning all nodes whose partial plans contained two consecutive $drive$ actions or those containing $orderTakeout$, $orderRestaurant$, or $cook$ actions not immediately followed by an eat action. As the results show, adding these simple pieces of control knowledge allows **PPLAN** to terminate in all 60 test cases, generally with far fewer nodes expanded in the process. Taken all together, we feel that these results speak to the effectiveness of our evaluation function in guiding the search but also to the interest of combining this approach with domain-dependent control knowledge.

An independent contribution of this paper is the creation of the dinner domain, a planning domain that can serve as a benchmark for problems in planning with preferences. In addition to affording a number of natural and compelling temporally extended preferences, the dinner domain is easily scaled either by increasing the number of objects involved (adding more restaurants, meals, etc.) or by making the events more complex (e.g., buying groceries, cooking, etc.). A complete axiomatization is available at `http://www.cs.toronto.edu/~sheila/pplan`.

TEST #	1	2	3	4	5	6	7	8	9	10
PPLAN	52	55	171	7	3	29	8	9	8	59
PPLAN^C	22	10	43	7	3	19	8	9	3	7
BFS	432	426	*	61	61	*	61	71	51	*
TEST #	11	12	13	14	15	16	17	18	19	20
PPLAN	64	54	15	8	16	49	10	29	3	29
PPLAN^C	12	10	15	8	16	29	10	19	3	19
BFS	421	495	71	61	495	*	82	*	61	*
TEST #	21	22	23	24	25	26	27	28	29	30
PPLAN	60	55	65	15	7	37	257	597	55	9
PPLAN^C	8	10	12	15	7	22	19137	169	10	3
BFS	*	432	432	408	51	*	*	51	426	61
TEST #	31	32	33	34	35	36	37	38	39	40
PPLAN	29	702	15	22	597	23	57	15	108	585
PPLAN^C	11	163	15	22	169	23	37	15	54	151
BFS	*	61	61	61	505	71	*	426	*	408
TEST #	41	42	43	44	45	46	47	48	49	50
PPLAN	7	68	15	27	115	7	*	13	1254	*
PPLAN^C	7	23	15	31	36	7	8157	15	85	340
BFS	51	408	426	*	*	51	*	60	*	*
TEST #	51	52	53	54	55	56	57	58	59	60
PPLAN	2	28	8	55	55	408	61	51	55	51
PPLAN^C	2	11	3	10	10	119	22	21	10	21
BFS	51	*	51	61	426	408	61	408	426	51

Figure 2: Nodes expanded by **PPLAN**, **PPLAN** augmented by hard constraints (**PPLAN**^{C}), and breadth-first search (**BFS**). The symbol * indicates that the program ran out of memory.

7 Related Work

There is a significant amount of related work on the general topic of representing and reasoning about user preferences. The literature is extensive, much of it originating within the field of economics not artificial intelligence (AI). We discuss the related work in AI below, elaborating further in (Bienvenu & McIlraith 2006).

In comparing our work to that of others, many of the distinctions we raise relate to whether preference formalisms are ordinal, qualitative or quantitative; whether they model temporal preferences or solely static preferences; whether the formalism is propositional or first order; and whether it induces a total order and if not the degree of incomparability in the ordering. In this context, our language is qualitative, models temporal preferences, is first order, and induces a total order. As further criteria for comparison, (Coste-Marquis *et al.* 2004) evaluate some propositional logic-based preference languages with respect to succinctness and expressiveness. We do not discuss this here.

A widely adopted language for studying user preferences in AI is the propositional CP-nets formalism (Boutilier *et al.* 2004). CP-nets enable the description of conditional *ceteris paribus* statements about user preferences (e.g., the user prefers red wine if meat is being served and white wine if fish is being served, *all other things being equal*). User preferences are represented in a graphical notation which is compact and which reflects the conditional independence and dependence of statements. Unlike our formalism, CP-nets is restricted to static, ordinal statements about preferences. As such, CP-nets cannot express temporal preferences, nor can it express relative importance of different preferences. The CP-nets formalism is simple and elegant, however it achieves this at the expense of expressiveness. There is often a high degree of incomparability between different states. This and the lack of temporal expressions makes it poorly suited to the task of planning with preferences.

Other noteworthy work includes that of Brewka on qualitative choice logic (QCL) (Brewka, Benferhat, & Berre 2004). This preference framework is designed to represent preferences over alternatives and induces a complete preorder over models. QCL provides a subset of the expressive power of our preference language and is similar in semantics to a previously proposed ordinal version of the language we describe here. In other related work (Brewka 2004), Brekwa proposes a language for answer set optimization called $\mathcal{PLD}$. The basic elements of $\mathcal{PLD}$ are rules which code context-dependent preferences over answer sets. More

unable to get comparative statistics in order to compare the two approaches.

[9] In order to facilitate the comparison, breadth-first search was passed the k-optimal weight as a parameter and run until it found an plan with this weight (or ran out of memory).

complex preference formulae are formed using different aggregation operators: sum, (ranked) set inclusion, (ranked) cardinality, pareto, and lexicographic order. Finally, the possibilistic logic approach to preferences (Benferhat, Dubois, & Prade 2001) is notable in that it proposes a qualitative preference framework, thus allowing the relative importance of preferences to be specified. All of the approaches we have seen so far are limited in that they do not consider temporal preferences, and hence are unable to express the types of preferences that interest us. Nonetheless, they all present their own advantages, many of which we have incorporated into our own framework.

There have also been several pieces of work related to the topic of planning with preferences. In (Delgrande, Schaub, & Tompits 2004) Delgrande et al. developed a useful framework for characterizing preferences and properties of preference-based planning. The preference language they propose is propositional and distinguishes between choice preferences and temporal preferences, but is less expressive than the language proposed here.

Most noteworthy of the related work is the work of Son and Pontelli (Son & Pontelli 2004) which developed a propositional language, $\mathcal{PP}$, for planning with preferences together with an implementation using answer-set programming. $\mathcal{PP}$ served as a starting point for the development of our language and we adopted their naming hierarchy of BDF, APF, and GPF, augmenting it with AgPf. Despite the similarity in names, there are significant differences between our preference languages, both in terms of syntax and semantics.

In particular, our language is first-order, which affords us far more compact and simple expression of preferences. It also enables the expression of preferences over unnamed objects, which is important for online planning where groundings may not be known a priori. Planning with Web services is a good example, where the execution of the plan can provide further knowledge of objects that a planner has preferences over (e.g., specific flights or hotels in the case of Web travel planning). Furthermore, our language is *qualitative* rather than simply *ordinal*, allowing us to express, for example, that one BDF is strongly preferred over another, as opposed to just providing an ordering over preferences.

At the GPF level, our language includes conditional preferences, which are useful (cf. CP-nets). Like $\mathcal{PP}$ we have the notion of *General And (Conjunction)* and *General Or (Disjunction)*, but we provide a different semantics for these constructs. According to $\mathcal{PP}$'s semantics, General And must be better on all its component preferences and General Or must be better on one component and at least as good on the others. We did not feel that these were natural ways of interpretting conjunction and disjunction. For example, one would expect that fully satisfying one of the component preferences should ensure satisfaction of a disjunction, but this does not follow from the $\mathcal{PP}$ semantics. In contrast our semantics is more in keeping with the boolean connectives that give these constructs their names. Moreover, our semantics induces a complete pre-order, whereas the semantics of $\mathcal{PP}$'s general preferences leads to great incomparability between plans. Finally, at the AgPF level, we provide several further methods for aggregating preferences, which those using or reviewing our work have found to be be compelling and useful, though our claim of usefulness has not been verified by a usability study.

Son and Pontelli have implemented a planner using answer-set programming. We believe our approach to planning is superior because it uses heuristic search to determine an optimal plan rather than computing plans and evaluating their relative merit. We've discussed this matter with the authors but have not been able to make an experimental comparison.

Also on the topic of planning with preferences, Brafman et al. (Brafman & Chernyavsky 2005) recently addressed this problem, specifying qualitative preferences over possible goal states using TCP-nets. A TCP-net is a tradeoff-enhanced CP-net, which allows the user to express priorities between variables. The most notable limitation of their language relative to ours is that they cannot express temporal preferences. Furthermore, since they are using TCP-nets, they suffer with incomparability of states, just as CP-nets do. Their approach to planning is to compile the problem into an equivalent CSP problem, imposing variable instantiation constraints on the CSP solver, according to the TCP-net. This is a promising method for planning, though it is not clear how it will extend to temporal preferences.

Finally, there has been a variety of work that uses quantitative preferences for planning or temporal reasoning. This includes Eiter et al.'s work on answer set planning with respect to plan length and numeric action costs (Eiter *et al.* 2003), work by Rossi and colleagues on reasoning with temporal soft constraints (Rossi, Venable, & Yorke-Smith 2003), PDDL3 (Gerevini & Long 2005) which builds a plan quality metric via a numeric combination of the penalties assigned to basic temporal preferences, and of course the extensive research on decision-theoretic planning such as (Haddawy & Hanks 1992) and MDPs (Puterman 1994). As we mentioned in the introduction, the quantitative nature of these frameworks makes preference elicitation difficult. This is why in our own work we decided to focus on qualitative preferences, which are more expressive than ordinal preferences yet much easier to elicit than quantitative preferences. As a useful middle ground, (Fritz & McIlraith 2006) integrate qualitative and quantitative preferences within an agent programming framework. The authors' express their qualitative preferences in a restricted version of the language proposed here.

8 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 qualitative user preferences. We proved that our semantics is preserved under progression. We also proposed an admissible evaluation function that establishes the optimality of best-first search. This led to the development of **PPLAN**, a best-first search, forward-chaining planner that accepts temporally extended goals and computes optimal preferred plans relative to length bounds. We further proved the correctness of the **PPLAN** algorithm. 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. In future work, we hope to apply our work to other planning problems, including the task of Web service composition (e.g., (McIlraith & Son 2002). We also have interest in applying our work to temporal diagnosis and to goal analysis within requirements engineering. We have developed several extensions to **PPLAN** that integrate hard constraints and that experiment with additional heuristics. In the future we hope to build a successor to **PPLAN** with heuristics that will further guide the planner towards preferred goals. Finally, work has also commenced on applying this framework to agent programming.

Acknowledgements

We would like to thank Jérôme Lang and the anonymous referees for helpful comments. Many thanks also goes to Shirin Sohrabi Araghi for her work on the implementation of **PPLAN**. Finally, we gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC) through grant number 229717-04 and through the Undergraduate Student Research Award (USRA) program.

References

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

Benferhat, S.; Dubois, D.; and Prade, H. 2001. *Applied Intelligence*, volume 14. Kluwer. chapter Towards a Possibilistic Logic Handling of Preferences, 303–317.

Bienvenu, M., and McIlraith, S. 2006. Planning with preferences: Theory and implementation. In preparation.

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

Brafman, R., and Chernyavsky, Y. 2005. Planning with goal preferences and constraints. In *Proceedings of The International Conference on Automated Plann ing and Scheduling*.

Brewka, G.; Benferhat, S.; and Berre, D. L. 2004. Qualitative choice logic. *Artificial Intelligence* 157(1-2). Special Issue on Nonmonotonic Reasoning.

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

Coste-Marquis, S.; Lang, J.; Liberatore, P.; and Marquis, P. 2004. Expressive power and succintness of preference representation languages. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004)*.

Delgrande, J.; Schaub, T.; and Tompits, H. 2004. Domain-specific preferences for causual reasoning and planning. In Dubois, D.; Welty, C.; and Williams, M., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004)*, 673–682. AAAI Press.

Ehrgott, M. 2000. *Multicriteria Optimization*. Berlin: Springer.

Eiter, T.; Faber, W.; Leone, N.; Pfeifer, G.; and Polleres, A. 2003. Answer set planning under action costs. *Journal of Artificial Intelligence Research* 19:25–71.

Fritz, C., and McIlraith, S. 2006. Decision-theoretic GOLOG with qualitative preferences. In *Proceedings of the 10th International Conference on Principles o f Knowledge Representation and Reasoning, Lake District, UK, June*.

Gabaldon, A. 2004. Precondition control and the progression algorithm. In Dubois, D.; Welty, C.; and Williams, M., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004)*, 634–643. AAAI Press.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in pddl3: The language of the fifth international planning competition. Technical report, University of Brescia.

Haddawy, P., and Hanks, S. 1992. Representations for decision-theoretic planning: Utility functions for deadline goals. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR96)*, 71 – 82.

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

McIlraith, S., and Son, T. 2002. Adapting Golog for composition of semantic web services. In *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002)*, 482–493.

Myers, K., and Lee, T. 1999. Generating qualitatively different plans through metatheoretic biases. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'99)*, 570–576.

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

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

Rossi, F.; Venable, K.; and Yorke-Smith, N. 2003. Temporal reasoning with preferences and uncertainty. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*.

Son, T., and Pontelli, E. 2004. Planning with preferences using logic programming. In Lifschitz, V., and Niemela, I., eds., *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-2004)*, number 2923 in Lecture Notes in Computer Science. Springer. 247–260.