Qualitative Dynamical Preferences in the Situation Calculus

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 non-Markovian 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 \textsc{PPlan}, a bounded best-first search planner that computes preferred plans. 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.

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.

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. We define the semantics of our preference language in the situation calculus [8]. Nevertheless, nothing requires that the planner be implemented using deductive plan synthesis in the situation calculus. Indeed our planner \textsc{PPlan}, a bounded best-first search planner, is a forward-chaining planner, in the spirit of TLPlan [1] and TALPlan [6], 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, yielding greater incomparability between outcomes (e.g., [3]). In the area of dynamical preferences, there are several recent and notable pieces of work. Son and Pontelli [10] have developed a language for planning with preferences together with an implementation using answer-set programming. Indeed we leverage their preference language \textsc{PP} in our work. Also notable is the work of Delgrande et al. [4], who have developed a framework for characterizing preferences and properties of preference-based planning. Rossi and colleagues (e.g., [9]) 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 [7]. Nevertheless, the elicitation of preferences in terms of Markovian numeric utilities makes these approaches less applicable to the types of preferences we examine.

2 Preliminaries

The situation calculus is a logical language for specifying and reasoning about dynamical systems [8]. 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., \(F(\mathcal{F}, s)\). 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 \(a \in A\) 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 \(\Sigma\) define the situations, their branching structure, and the situation predecessor relation \(\sqsubseteq. s \sqsubseteq s'\) states that situation \(s\) precedes situation \(s'\) in the situation tree. \(\Sigma\) 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 \(\text{Poss}(a, s)\), the conditions underwhich it’s possible to execute an action \(a\) in situation \(s\).
- axioms \(\mathcal{D}_{S_0}\) describing the initial situation.
atomic preference formulae define preferences over proper-
focus of this paper.

preference language modifies and extends the preference lan-
following their work, we provide a hierarchy of preference for-

ities, and general preference formulae define compositions of

s. Then a planning problem \( (D, G) \) is solvable if there exists a

plan trajectory \( \pi = (s_0, \ldots, s_n) \) such that:

\[
D \models (3_s) \text{executable}(s) \land G(s)
\]

where \( \text{executable}(s) \iff (\forall a, s'). do(a, s') \subseteq s \supset \text{Poss}(a, s') \).

We refer to situation \( s = do(\bar{a}, S_0) \) as the plan trajectory
and the sequence of actions \( \bar{a} = a_1a_2\ldots a_n \) as the associated plan.
The length of this plan is \( n \). The set of all plans is denoted by \( \Pi \), and \( \Pi^k \) denotes the subset of plans of length \( \leq k \). A planning problem \( (D, G) \) is solvable if it has at least
one plan. It is \( k \)-solvable if it has a plan of length \( \leq k \).

\section{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 pref-
ences concerning where and what she eats that make some plans better than others. It is the definition of these prefer-
ences and the generation of these preferred plans that is the
focus of this paper.

\subsection{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 [10]. Following our work, we provide a hierarchy of preference for-
mae. Basic desire formulae define properties of situations, atomic preference formulae define preferences over propri-
ties, and general preference formulae define compositions of

 preferred situation properties. The planner is ultimately given
one general preference formula (subsequently referred to as
simply as a preference formula) relative to which a preferred plan is

Generated.

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

1. \( \mathcal{F} \subset B \)
2. \( \mathcal{R} \subset B \)
3. If \( f \in \mathcal{F} \), then final\( (f) \in B \)
4. If \( a \in \mathcal{A} \), then occ\( (a) \in B \)
5. If \( \varphi, \varphi_1, \varphi_2 \) are in \( B \), then so are \( \neg \varphi, \varphi_1 \land \varphi_2, \varphi_1 \lor \varphi_2 \), \( (\exists x) \varphi \), \( (\forall x) \varphi \), \text{next} (\varphi) \), \text{always} (\varphi) \), \text{eventually} (\varphi) \), and \text{until} (\varphi, \varphi_2) \).

BDFs establish desirable properties of situations. By com-
paring BDFs using boolean and temporal connectives, we are
able to express a wide variety of properties. We illustrate their
use with BDFs from our motivating example:

\[ \begin{align*}
\text{has Ingredients (spag)} \land \text{knows HowToMake (spag)} \quad & (P1) \\
(\exists x). \text{has Ingredients (x)} \land \text{knows HowToMake (x)} \quad & (P2) \\
\text{final (kitchenClean)} \quad & (P3) \\
(\exists x). \text{eventually (occ (cook (x)))} \quad & (P4) \\
(\exists x). (\exists y). \text{eventually (occ (orderTakeout (x, y)))} \quad & (P5) \\
(\exists x). (\exists y). \text{eventually (occ (orderRestaurant (x, y)))} \quad & (P6) \\
\text{always (eventually (occ (eat (x))) \land \text{chinese (x))}) \quad & (P7)
\end{align*} \]

P1 states that in the initial situation Claire has the ingredients
and the know-how to cook spaghetti. P2 is more general, ex-
pressing that in the initial situation Claire has the ingredients
to make something she knows how to make. Observe that
defluent 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 enable description of desirable properties of situations, they do not enable us to express preferences be-
tween alternative properties. For example, we cannot say that
Claire prefers cooking to ordering take-out. To do so, we de-
define Atomic Preference Formulae, following the definition in
[10].

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

An atomic preference formula expresses a preference over alternative properties defined by BDFs. For example, Claire
can express her preference over what to eat (pizza, followed
by spaghetti, followed by crêpes) using P8:

\[ \text{occ (eat (pizza))} \gg \text{occ (eat (spag))} \gg \text{occ (eat (crêpes))} \quad (P8) \]

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:

\[ P5 \gg P2 \land P4 \gg P6 \gg \neg P2 \land P4 \quad (P9) \]

\[ \text{occ (spag)} \land \text{knows HowToMake (spag)} \quad (P1) \]

\[ (\exists x). \text{has Ingredients (x)} \land \text{knows HowToMake (x)} \quad (P2) \]

\[ \text{final (kitchenClean)} \quad (P3) \]

\[ (\exists x). \text{eventually (occ (cook (x)))} \quad (P4) \]

\[ (\exists x). (\exists y). \text{eventually (occ (orderTakeout (x, y)))} \quad (P5) \]

\[ (\exists x). (\exists y). \text{eventually (occ (orderRestaurant (x, y)))} \quad (P6) \]

\[ \text{always (eventually (occ (eat (x))) \land \text{chinese (x))}) \quad (P7) \]

\[ \text{occ (spag)} \land \text{knows HowToMake (spag)} \quad (P1) \]

\[ (\exists x). \text{has Ingredients (x)} \land \text{knows HowToMake (x)} \quad (P2) \]

\[ \text{final (kitchenClean)} \quad (P3) \]

\[ (\exists x). \text{eventually (occ (cook (x)))} \quad (P4) \]

\[ (\exists x). (\exists y). \text{eventually (occ (orderTakeout (x, y)))} \quad (P5) \]

\[ (\exists x). (\exists y). \text{eventually (occ (orderRestaurant (x, y)))} \quad (P6) \]

\[ \text{always (eventually (occ (eat (x))) \land \text{chinese (x))}) \quad (P7) \]

\[ \text{occ (spag)} \land \text{knows HowToMake (spag)} \quad (P1) \]

\[ (\exists x). \text{has Ingredients (x)} \land \text{knows HowToMake (x)} \quad (P2) \]

\[ \text{final (kitchenClean)} \quad (P3) \]

\[ (\exists x). \text{eventually (occ (cook (x)))} \quad (P4) \]

\[ (\exists x). (\exists y). \text{eventually (occ (orderTakeout (x, y)))} \quad (P5) \]

\[ (\exists x). (\exists y). \text{eventually (occ (orderRestaurant (x, y)))} \quad (P6) \]

\[ \text{always (eventually (occ (eat (x))) \land \text{chinese (x))}) \quad (P7) \]

\[ \text{occ (spag)} \land \text{knows HowToMake (spag)} \quad (P1) \]

\[ (\exists x). \text{has Ingredients (x)} \land \text{knows HowToMake (x)} \quad (P2) \]

\[ \text{final (kitchenClean)} \quad (P3) \]

\[ (\exists x). \text{eventually (occ (cook (x)))} \quad (P4) \]

\[ (\exists x). (\exists y). \text{eventually (occ (orderTakeout (x, y)))} \quad (P5) \]

\[ (\exists x). (\exists y). \text{eventually (occ (orderRestaurant (x, y)))} \quad (P6) \]

\[ \text{always (eventually (occ (eat (x))) \land \text{chinese (x))}) \quad (P7) \]
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.

Again, an atomic preference represents a preference over alternative \( \varphi_i \)'s. We wish to satisfy the BDF \( \varphi_i \) 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 \( \varphi_i \), and this option is the least preferred.

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 \( \Phi \) is a general preference formula if one of the following holds:

- \( \Phi \) is an atomic preference formula
- \( \Phi = \gamma : \Psi \), where \( \gamma \) is a basic desire formula and \( \Psi \) is a general preference formula [Conditional]
- \( \Phi = !\Psi \), for \( \Psi \) a general preference formula [Negation]
- \( \Phi \) is one of
  - \( \Psi_0 \land \Psi_1 \land \ldots \land \Psi_n \) [General And]
  - \( \Psi_0 \mid \Psi_1 \mid \ldots \mid \Psi_n \) [General Or]
  - \( \Psi_0 \triangleright \Psi_1 \triangleright \ldots \triangleright \Psi_n \) [Lex Order]

where \( n \geq 1 \) and each \( \Psi_i \) is a general preference formula.

Here are some example general preference formulae:

\[
P2 : P4 \quad (P10) \quad !P8 \quad (P11) \quad P8 \land P9 \quad (P12)
\]

P10 states that if Claire initially has the ingredients for something she can make, then she should cook. !\( \Phi \) does the opposite of \( \Phi \). E.g., P11 states that 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. P12 maximizes the satisfaction of both Claire’s food and time preferences. P13 can be used if she would be content if either of the two were satisfied. P14 tells us that while Claire cares about both her preferences, her food preference takes priority.

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 [110]. Quantifiers, variables, non-fluent relations, and a conditional construct have been added to 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. 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 interpreted as situation calculus formulae. We associate with each situation term a weight between 0 and 1, depending upon how greatly it deviates from satisfying a preference formula. 0 indicates complete satisfaction, whereas 1 represents complete dissatisfaction. Weights were necessary to differentiate situations that would be deemed “incomparable” in less expressive preference languages (e.g., [31]). Preference formulae are evaluated relative to an action theory \( D \). 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 [5] to explicitly denote that \( \varphi \) 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 \( F[s] \) denotes the re-insertion of situation term \( s \).

We interpret BDFs in the situation calculus as follows.

- \( \varphi \in F \), \( \varphi[s,s'] \equiv \varphi[s] \)
- \( \varphi \in R \), \( \varphi[s,s'] \equiv \varphi \)
- \( \text{final}(\varphi)[s,s'] \equiv \varphi[s'] \)
- \( \text{occ}(\varphi)[s,s'] \equiv \text{do}(a, s) \sqsubseteq s' \)
- \( \text{eventually}(\varphi)[s,s'] \equiv (\exists s_1 : s \sqsubseteq s_1 \sqsubseteq s') \varphi[s_1,s'] \)
- \( \text{always}(\varphi)[s,s'] \equiv (\forall s_1 : s \sqsubseteq s_1 \sqsubseteq s') \varphi[s_1,s'] \)
- \( \text{next}(\varphi)[s,s'] \equiv (\exists s_2 : s \sqsubseteq s_2 \sqsubseteq s') (\varphi[s_2,s'] \land (\forall s_1 : s \sqsubseteq s_1 \sqsubseteq s_2) \varphi[s_1,s']) \)

Boolean connectives and quantifiers are already part of the preference formulae.

**Definition 4 (Basic Desire Satisfaction).** Let \( D \) be an action theory. A situation \( s = do([a_1, \ldots, a_n], S_0) \) satisfies a basic desire formula \( \varphi \) just in the case that \( D \models \varphi[S_0, s] \). We define \( w_s(\varphi) \) to be the weight of situation \( s \) wrt BDF \( \varphi \), \( w_s(\varphi) = 0 \) if \( s \) satisfies \( \varphi \), otherwise \( w_s(\varphi) = 1 \).

We can generalize this definition as follows.

**Definition 5.** Let \( D \) be an action theory, and let \( s \) and \( s' \) be situations such that \( s \sqsubseteq s' \). The situations beginning in \( s \) and terminating in \( s' \) satisfy \( \varphi \) just in the case that \( D \models \varphi[s,s'] \). We define \( w_{s,s'}(\varphi) \) to be the weight of the situations originating in \( s \) and ending in \( s' \) wrt BDF \( \varphi \), \( w_{s,s'}(\varphi) = 0 \) if \( \varphi \) is satisfied, otherwise \( w_{s,s'}(\varphi) = 1 \).

**Example 1:** Consider the plan trajectory \( s = do([\text{cleanDishes(he, she, cook(crêpes), eat(crêpes), cleanDishes(he, she, S_0)}]) \) and the initial database \( D_{S_0} = \{ \text{hungry(S_0), hasIngrnts(spag, S_0), at(home, S_0)} \} \)

\[\text{Temporal formulae follow [5], using the abbreviations:} \]
\[
(\exists s_1 : s \sqsubseteq s_1 \sqsubseteq s) \Phi = (\exists s_1 : s \sqsubseteq s_1 \land s_1 \sqsubseteq s \land \Phi)
\]
\[
(\forall s_1 : s \sqsubseteq s_1 \sqsubseteq s) \Phi = (\forall s_1 : \{s \sqsubseteq s_1 \land s_1 \sqsubseteq s \} \supset \Phi)
\]
has Ingrnts\{crêpes, S\}, knows HowToMake\{crêpes\}).

Then we have the following:
\[ w_s(P1) = 1 \quad w_s(P2) = 0 \quad w_s(P3) = 0 \quad w_s(P4) = 0 \]
\[ w_s(P5) = 1 \quad w_s(P6) = 1 \quad w_s(P7) = 0 \]

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

Evaluating weights with respect to Example 1, we get:
\[ w_s(P8) = \frac{2}{3} \quad w_s(P9) = \frac{1}{4} \]

Definition 7 (General Preference Satisfaction I). Let \( s \) be a situation and \( \Phi \) be a general preference formula. Then \( w_s(\Phi) \) is defined as follows:
- \( w_s(\varphi \Rightarrow \psi) = \left\{ \begin{array}{lcl} 0 & \text{if} & w_s(\gamma) = 1 \\ w_s(\psi) & \text{otherwise} \end{array} \right. \)
- \( w_s(! \Phi) = 1 - w_s(\Phi) \)
- \( w_s(\Psi_0 \land \Psi_1 \land \ldots \land \Psi_n) = \max \{ w_s(\Psi_i) : 1 \leq i \leq n \} \)
- \( w_s(\Psi_0 \lor \Psi_1 \lor \ldots \lor \Psi_n) = \min \{ w_s(\Psi_i) : 1 \leq i \leq n \} \)

The weight of Lex Order preferences makes use of the following two definitions.

Definition 8 (Set of Possible Weights). We use \( V(\Phi) \) to denote the set of possible weights of \( \Phi \). This can be defined inductively as follows:
- If \( \Phi \) is a BDF, then \( V(\Phi) = \{ 0, 1 \} \)
- If \( \Phi = \varphi \Rightarrow \psi \) or \( \Phi = ! \psi \), then \( V(\Phi) = \{ 0, 1, \ldots, n+1 \} \)
- If \( \Phi = \varphi \land \psi \) or \( \Phi = \varphi \lor \psi \), then \( V(\Phi) = \bigcup_{i=0}^{n} V(\Psi_i) \)
- If \( \Phi = \varphi_0 \lor \varphi_1 \lor \ldots \lor \varphi_n \), then \( V(\Phi) = \{ \Pi_{i=0}^{n} V(\Psi_i) \} : i = 0, 1, \ldots, \Pi_{i=0}^{n} |V(\Psi_i)|-1 \}

For example, we obtain the following sets of possible weights:
\[ V(P8) = \{ 0, \frac{1}{2}, \frac{2}{3}, 1 \} \]
\[ V(P9) = \{ 0, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, 1 \} \]
\[ V(P13) = V(P8 \lor P9) = V(P8) \cup V(P9) \]
\[ = \{ 0, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, 1 \} \cup \{ 0, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}, 1 \} = \{ 0, \frac{1}{2}, \frac{1}{3}, \frac{2}{3}, \frac{3}{4}, 1 \} \]

Definition 9 (Position of a Weight). Given a preference formula \( \Phi \) and some \( x \in V(\Phi) \), we define the position of \( x \) with respect to \( V(\Phi) \), written \( Pos(x, V(\Phi)) \), to be equal to \( \{ y \in V(\Phi) : y < x \} \).

To illustrate this definition, we determine the positions of weights with respect to the set \( V(P13) = \{ 0, \frac{1}{2}, \frac{1}{3}, \frac{2}{3}, \frac{3}{4}, 1 \} \):
\[ Pos(0, V(P13)) = 0 \quad Pos(\frac{1}{2}, V(P13)) = 2 \]
\[ Pos(\frac{1}{3}, V(P13)) = 5 \quad Pos(\frac{2}{3}, V(P13)) = 3 \]

Definition 10 (General Preference Satisfaction II). Let \( s \) be a situation and let \( \Phi = \Psi_0 \land \Psi_1 \land \ldots \land \Psi_n \). We define
\[ w_s(\Phi) = \sum_{i=0}^{n} \frac{Pos(w_s(\Psi_i), V(\Psi_i)) \times \Pi_{j=1}^{n} |V(\Psi_j)|}{\Pi_{i=0}^{n} |V(\Psi_i)| - 1} \]

Returning to Example 1,
\[ - w_s(P2) = 0 \quad w_s(P3) = 0 \quad w_s(P4) = 0 \]
\[ - w_s(P8 & P9) = \max \{ \frac{2}{3}, \frac{1}{2} \} = \frac{2}{3} \]
\[ - w_s(P8 | P9) = \min \{ \frac{3}{4}, \frac{1}{2} \} = \frac{1}{2} \]
\[ - w_s(P8 > P9) = \frac{Pos(w_s(P8), V(P8)) + Pos(w_s(P9), V(P9))}{\Pi_{i=0}^{n} |V(\Psi_i)|} = \frac{2 \times 5}{4 \times 6 - 1} = \frac{11}{19} \]

Definition 11 (Preferred Situations). A situation \( s_1 \) is preferred to a situation \( s_2 \) with respect to a preference formula \( \Phi \), written \( Pref(s_1, s_2, \Phi) \), if \( w_s(\Phi) < w_{s_2}(\Phi) \). Situations \( s_1 \) and \( s_2 \) are indistinguishable with respect to a preference formula \( \Phi \), written \( s_1 \approx \Phi, \ s_2 \), if \( w_s(\Phi) = w_{s_2}(\Phi) \).

4 Planning with Preferences

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

Definition 12 (Preference-Based Planning Problem). A preference-based planning problem is a tuple \( \langle D, G, \Phi \rangle \), where \( D \) is an action theory, \( G \) is the goal, and \( \Phi \) is a preference formula.

Definition 13 (Preferred Plan). Consider a preference-based planning problem \( \langle D, G, \Phi \rangle \) and plans \( \alpha_1 \) and \( \alpha_2 \). We say that plan \( \alpha_1 \) is preferred to plan \( \alpha_2 \iff Pref(\alpha_1, \alpha_2, \Phi) \).

Definition 14 (Optimal Plan, Ideal Plan). An optimal plan with respect to \( \langle D, G, \Phi \rangle \) is any plan \( \bar{a} \in \Pi \) s.t.
\[ w_{do(\bar{a}, S_0)}(\Phi) = \min \{ w_{do(\bar{a}, S_0)}(\Phi) : \bar{b} \in \Pi \} \]
A plan \( \bar{a} \) is an ideal plan iff \( w_{do(\bar{a}, S_0)}(\Phi) = 0 \).

Definition 15 (k-Optimal Plan). Given \( \langle D, G, \Phi \rangle \) and a length bound \( k \), a \( k \)-optimal plan is any plan \( \bar{a} \in \Pi^k \) s.t.
\[ w_{do(\bar{a}, S_0)}(\Phi) = \min \{ w_{do(\bar{a}, S_0)}(\Phi) : \bar{b} \in \Pi^k \} \]

Definition 16 (q-Satisfactory Plan). Given a preference-based planning problem and a quality bound \( 0 \leq q \leq 1 \), a \( q \)-satisfactory plan is any \( \bar{a} \in \Pi \) such that \( w_{do(\bar{a}, S_0)}(\Phi) \leq q \).

4.1 Progression

In Section 5 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 \([1; 6]\), 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 progression preserves the semantics of preference formulae.

In order to define the progression operator, we add the propositional constants \text{TRUE} and \text{FALSE} to both the situation calculus and to our set of BDFs, where \( \mathcal{D} \vdash \text{TRUE} \) and \( \mathcal{D} \not\vdash \text{FALSE} \) for every action theory \( \mathcal{D} \). We further add the BDF \text{occNext}(\alpha), \alpha \in \mathcal{A}, to capture the progression of \text{occ}(\alpha).
Definition 17 (Progression of a Basic Desire Formula). Let $s$ be a situation, and let $\varphi$ be a basic desire formula. The progression of $\varphi$ through $s$, written $\rho_s(\varphi)$, is given by:

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

Returning to Example 1,

$$\begin{align*}
\rho_s(\text{always(kitchenClean)}) &= \rho_s(\text{kitchenClean}) \land \text{always(kitchenClean)} \\
&= \text{FALSE} \land \text{always(kitchenClean)} \\
\rho_s((\exists x). \text{hasIngrnts}(x)) &= \bigvee_{c \in C} \rho_s(\text{hasIngrnts}(c))
\end{align*}$$

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 [2]. 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 formulae by the use of Boolean simplification and bounded quantification [1]. Definition 17 shows us how to progress a preference formula one step, through one situation. We extend this to the notion of iterated progression.

Definition 18 (Iterated Progression). The iterated progression of a preference formula $\Phi$ through situation $s = do(\bar{a}, S_0)$, written $\rho_s^*(\Phi)$, is defined by:

$$\begin{align*}
\rho_s^*(\text{jointPref}(\Phi)) &= \rho_s(\text{jointPref}(\Phi)) \\
\rho_s^*(\text{pref}(a, \Phi)) &= \rho_s(\text{pref}(a, \Phi)) \quad (a \in \mathcal{A}_s)
\end{align*}$$

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 $\varphi$ be a BDF. Then $D \models [s_0, s]$ iff $D \models \rho_s^*(\varphi)[s, s]$

Proof Sketch: The proof proceeds by induction on the structure of $\varphi$.

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 $\Phi$ 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.

PPLAN(init, goal, pref, maxLength, desiredWt)
frontier $\leftarrow$ INITFRONTIER(init, pref)
bestPlanSoFar $\leftarrow [\]$ 
bestWtSoFar $\leftarrow$ MAXWt(pref)+1

while frontier $\neq \emptyset$ and bestWtSoFar $> desiredWt$

current $\leftarrow$ REMOVEFIRST(frontier)
if goal $\subseteq$ state and weight $<$ bestWtSoFar
bestPlanSoFar $\leftarrow$ partialPlan
bestWtSoFar $\leftarrow$ weight
end if
neighbours $\leftarrow$ EXPAND(partialPlan, state, progPref)
frontier $\leftarrow$ SORTNMERGEVal(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 maxLength, and a quality bound desiredWt, designating the maximum acceptable plan weight. The algorithm returns a plan bestPlanSoFar and its weight bestWtSoFar.

A naive implementation would require computing alternative plan trajectories and then evaluating their relative weights. This is grossly inefficient, 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

---

1We assume a finite domain. $t^{\text{occ}}$ denotes the result of substituting the constant $c$ for all instances of the variable $v$ in $t$.

2Refer to [2] for a more detailed description of the algorithm.
weight and evaluates preference formulae across shared partial plans. Progression is commonly used to evaluate domain control knowledge in forward chaining planners (e.g., [1, 6]) 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 \([\text{weight}, \text{partialPlan}, \text{state}, \text{pref}]\), 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, \textbf{PPLAN} removes the first node from the frontier and places it in \textit{current}. If the partial plan of \textit{current} satisfies the goal and has lower weight than \textit{bestWtSoFar}, then \textit{bestPlanSoFar} and \textit{bestWtSoFar} are replaced by \textit{current’s} partialPlan and weight respectively. Next we call the function \textbf{EXPAND} with \textit{current’s} node arguments as input. If \textit{partialPlan} has length equal to \textit{maxLength}, then the frontier is updated to the empty list. Otherwise, \textbf{EXPAND} generates a new set of nodes, one for each action executable in \textit{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 \textbf{PPLAN} is given in the following theorem.

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

\textbf{Proof Sketch:} The proof proceeds by proving termination and then proving the correct output properties by cases [2].

5.1 Experimental Results

We tested \textbf{PPLAN} on 24 instances of the dinner example and 31 instances of the simpler school travel example used in [10]. We compared the number of nodes expanded using \textbf{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 \textbf{PPLAN} demonstrates poorer performance than BFS. Recall that \textbf{PPLAN}’s best-first search explores plans based on weight then length. As a consequence, \textbf{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, this behavior occurs infrequently, and the heuristic generally leads to significantly improved performance.

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

Figure 2: Nodes expanded by \textbf{PPLAN} & breadth-first search.

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 non-Markovian, qualitative user preferences. We proved that our semantics is preserved under progression. This led to the development of \textbf{PPLAN}, a best-first search, forward-chaining planner that computes optimal preferred plans relative to quality and length bounds. We further proved the correctness of the \textbf{PPLAN} algorithm. Our planner can be modified to compute the optimal plan without a quality bound and is trivially extended to include hard user constraints. 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.

References


