Compiling Qualitative Preferences into Decision-Theoretic Golog Programs: Extended Version with Proofs

Christian Fritz and **Sheila McIlraith**
Department of Computer Science
University of Toronto
Toronto, Ontario. Canada.
{fritz,sheila}@cs.toronto.edu

July 31, 2005

Abstract

Personalization is becoming increasingly important in agent programming, particularly as it relates to the Web. We propose to develop underspecified, task-specific agent programs, and to automatically personalize them to the preferences of individual users. To this end, we propose a framework for agent programming that integrates rich, non-Markovian, qualitative user preferences with quantitative Markovian reward functions. We begin with DT-Golog, a first-order, decision-theoretic agent programming language in the situation calculus. We present an algorithm that compiles qualitative preferences into Golog programs and prove it sound and complete with respect to the space of solutions. To integrate these preferences into DT-Golog we introduce the notion of multi-program synchronization and restate the semantics of the language as a transition semantics. We demonstrate the utility of this framework with an application to personalized travel planning over the Web. To the best of our knowledge this is the first work to combine qualitative and quantiative preferences for temporal reasoning. Further, while the focus of this paper is on the integration of qualitative and quantative preferences, a side effect of this work is realization of the simpler task of integrating qualitative preferences alone into agent programming.

1 Introduction

Personalization is becoming increasingly important to agent programming. Service-sector agent programs such as personal assistants or travel planners are often characterized by a relatively well-defined but under-specified set of tasks that can be realized in a variety of different ways. As with an office admin assistant or a travel agent, these high-level tasks are commissioned by numerous different customers/users. A *good* agent program, like a good office assistant or travel planner must be able to personalize the service they provide to meet the preferences and constraints of the individual.

Consider the oft-used example of travel planning: *Fiona would like to book a trip from Toronto, Canada to Edinburgh, Scotland for work. She'd like to depart between July 25 and 28, returning no sooner than August 5, but no later than August 8. She would prefer not to connect through London Heathrow, as she had a bad experience being stuck at Heathrow when air traffic controllers went on strike last year. She'll need a hotel in Edinburgh, preferrably close to the castle but if the plane arrives late at night, she'd prefer a hotel close to the airport. Fiona needs to economize, so she'd like the cheapest flights and hotel accommodations possible. Nevertheless, she's willing to pay $100 more to get a direct flight. Finally, she has to work July 29 – August 5, so she's willing to spend up to $200 more to maximize sightseeing days before July 29 and/or after August 5.*

This, presumably realistic setting, displays three types of constraints or preferences that are commonplace in many planning and agent programming application domains: hard constraints (when to go and where), qualitative preferences (airport and hotel preferences), and quantitative preferences (financial restrictions).

We approach the problem of personalizable agent programs by developing task-specific, but underspecified agent programs that have sufficient non-determinism to support personalization. Personalization is achieved by integrating these agent programs with the three types of constraints illustrated in our example above. The goal of this paper is to investigate the integration of qualitative and quantitative preferences into agent programming, and specifically into the agent programming language Golog [**?**].

Golog is a first-order agent programming language based on the situation calculus. Golog enables the specification of, potentially nondeterministic, agent programs in the context of a domain-specific action theory. As such, Golog programs impose hard constraints on the possible evolution of the domain. Decision-Theoretic Golog (DT-Golog) [**?**] extends Golog with the ability to solve MDP-like planning problems up to a given horizon and starting in a known initial situation. In so doing, DT-Golog can handle infinite state (situation) spaces while exploiting the underlying power of Golog to restrict the search space.

There is a large body of research on the use of quantitive preferences in automated reasoning. Indeed, decision-theoretic planning via Markov Decision processes (MDP) [**?**] provides an effective means of generating task plans that maximize a user's expected utility. Unfortunately, preferences and constraints must be specified in terms of numeric, Markovian reward functions. Such specifications can be difficult to elicit and don't capture qualitative user preferences. Bacchus et al. [**?**] addressed the Markovian restriction, by enabling the use of non-Markovian rewards. They did so by augmenting state representation with a new set of temporal variables. Nevertheless, they did not allow for qualitative preferences.

Unfortunately, there has been little work on the incorporation of qualitative preferences into planning, save recent work by [**?**, **?**, **?**]. These approaches are able to represent qualitative non-Markovian user preferences, while [**?**, **?**] also propose a means of planning with such preferences.

In [**?**] Domshlak et al. integrate quantitative soft constraints and qualitative preferences expressed using the CP-nets formalism [?]. They approach the problem by approximating the CP-net with soft constraints expressed in a semi-ring formalism. Nevertheless, their focus is on reasoning about preferences, i.e. deciding on an ordering of possible world states, and it is not obvious how their approach applies to planning or agent programming. In particular, the language they use for specifying preferences does not enable the expression of temporally extended preferences, which we believe are essential to the task at hand.

In this paper we address the problem of combining non-Markovian qualitative preferences, expressed in first-order temporal logic, with quantitative decision-theoretic reward functions and hard symbolic constraints. We do so by compiling non-Markovian qualitative preferences into a DT-Golog program, integrating the potentially competing preferences through a multi-program synchronization. The resultant DT-Golog program, maximizes the users expected utility within the most qualitatively preferred plans. We prove the soundness and completeness of our compilation algorithm. To the best of our knowledge, this is the first work to combine qualitative and quantitative preferences for temporal reasoning.

Our work is related to that of Gabaldon [?] who, following previous work by Bacchus and Kabanza [?] and Doherty and Kvarnstrm [?], compiles temporal logic formulae into preconditions of actions in the situation calculus. There, the temporal formulae are hard constraints that serve to reduce the search space. In contrast, we are unable to eliminate any of the search space, since qualitatively less preferred situations may yield the best final solution. Also related is the work of Sardina and Shapiro [?] who integrate qualitative prioritized goals into the IndiGolog programming language. Our approach differs from theirs in serveral ways: our qualitative preference language is richer than their specification of prioritized goals; we compile preferences into a Golog program which is more efficient from a computational perspective; and we enable the integration of both qualitative and quantitative constraints.

In Section 2 we review the situation calculus and Golog. In Section 3 we propose a first-order language for specifying non-Markovian qualitative user preferences. The semantics of the language is described in the situation calculus. Section 4 describes our approach to integrating preferences. It comprises three steps: compilation of non-Markovian qualitative preferences into a Golog program; multi-program synchronization of the resulting Golog program with an existing Golog program; and given this newly synchronized program, a means of defining preferences over different possible subprograms. Included are a soundness and completeness result relating to our compilation, and a new transition semantics for DT-Golog. We have implemented our approach as an extension to Readylog [?], an existing on-line decision-theoretic Golog interpreter. We demonstrate its utility with an application to personalized travel planning over the Web, as discussed in Section 5. We summarize our contributions in Section 6.

2 Situation Calculus and Golog

The situation calculus is a logical language for specifying and reasoning about dynamical systems [?]. In the situation calculus, the *state* of the world is expressed in terms of functions and relations (fluents) relativized to a particular *situation* s, e.g., $F(\vec{x}, s)$. In this paper, we distinguish between the set of fluent predicates, $\mathcal{F}$, and the set of non-fluent predicates, $\mathcal{R}$, representing properties that do not change over time. A situation s is a *history* of the primitive actions, $a \in \mathcal{A}$, performed from a distinguished 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. Details of the form of these axioms can be found in [**?**]. Following convention we will generally refer to fluents in situation-suppressed form, e.g., $at(toronto)$ rather than $at(toronto, s)$.

Golog (e.g., [**?**]) is a high-level logic programming language for the specification and execution of complex actions in dynamical domains. It builds on top of the situation calculus by providing Algol-inspired extralogical constructs for assembling primitive situation calculus actions into complex actions δ. Constructs include the following:

---

a — primitive actions
$\delta_1; \delta_2$ — sequences
$\phi?$ — tests
$(\pi x)\delta(x)$ — nondeterministic choice of arguments
δ^* — nondeterministic iteration
nondet(L) – nondeterministic choice of (complex) action in list, L
if ϕ **then** δ_1 **else** δ_2 **endIf** – conditionals
proc $P(\vec{v})$ δ **endProc** — procedure

These constructs can be used to write programs in the language of a domain theory, e.g.,

buyAirTicket($\vec{x}$);
  if far **then** rentCar($\vec{y}$) **else** bookTaxi($\vec{y}$) **endIf**.

There are two popular semantics for Golog programs: the original evaluation semantics [**?**] and a related single-step transition semantics that was proposed for on-line execution [**?**]. Following the evaluation semantics, complex actions are macros that expand to situation calculus formulae. The abbreviation $Do(\delta, S_0, do(\vec{a}, S_0))$ denotes that the Golog program δ, starting execution in S_0 will legally terminate in situation $do(a_1, do(a_2, \ldots, do(a_n, S_0)))$ [1]. The following are some example macro expansions.

$Do(a, s, s') \stackrel{\text{def}}{=} Poss(a[s], s) \wedge s' = do(a[s], s)$[2]
$Do(?(\varphi), s, s') \stackrel{\text{def}}{=} \varphi[s] \wedge s = s'$.
$Do(\mathbf{nondet}([\sigma_1|\bar{\sigma}], s, s') \stackrel{\text{def}}{=}$
  $Do(\sigma_1, s, s') \vee Do(\mathbf{nondet}(\bar{\sigma}), s, s')$[3]
$Do(\mathbf{nondet}([\sigma_1], s, s') \stackrel{\text{def}}{=} Do(\sigma_1, s, s')$

[1] which we abbreviate to $do(\vec{a}, S_0)$ or $do([a_1, \ldots, a_n], S_0)$.
[2] $a[s]$ denotes the re-insertion of s into fluent arguments of a.
[3] $[a|r]$ denotes a list with first element a, and rest of list r.

Given a domain theory, $\mathcal{D}$ and Golog program δ, program execution must find a sequence of actions $\vec{a}$ such that: $\mathcal{D} \models Do(\delta, S_0, do(\vec{a}, S_0))$. Recall that $\mathcal{D}$ induces a tree of situations rooted at S_0. Requiring that $\mathcal{D}$ entails $Do(\delta, S_0, do(\vec{a}, S_0))$ serves to constrain the situations in the tree to only those situations that are consistent with the expansion of δ.

These hard constraints can reduce the problem size by orders of magnitude. Consider the following estimate of our travel planning example. The full grounded search space involves 365^2 date combinations and 1901 airports. Assuming 10 available flights for every combination, there are more than $4.8 \cdot 10^{12}$ flights. Optimistically assuming that at each destination there are only 10 hotels with 5 room types each, the total number of possible action combinations increases to $6.2 \cdot 10^{21}$. Using a DT-Golog procedure such as the one that follows reduces the number of alternatives to approximately $3 \cdot 3 \cdot 10 \cdot 50 = 4500$ cases that are relevant to Fiona. Such reductions are of particular importance for agent programming on the Web, where the vastness of information creates enormous search spaces.

In this paper we exploit a decision-theoretic variant of Golog called DT-Golog [**?**], which extends Golog to deal with uncertainty in action outcomes and general reward functions. DT-Golog can be viewed alternatively as an extension to Golog, or as a means to give "advice" to a decision-theoretic planner that maximizes expected utility.

For example, our travel planning problem could be described by the following DT-Golog procedure:

```
proc( travel_planner,
[ pickBest( depart_dt, [726..728],
  pickBest( return_dt, [805..807],
   [ searchFlight("YYZ", "EDI", depart_dt, return_dt),
     searchHotel("EDI", depart_dt, return_dt),
     pickBest( bestFlight, allFlights,
               [ reserveFlight(bestFlight),
                 if(not(error),payFlight(bestFlight))]),
     ?(and([not(outflight = none),not(inflight = none)])),
     pickBest( bestHotel, allHotels,
               [ reserveHotel(bestHotel),
                 if(not(error),payHotel(bestHotel))]),
     ?(not(hotel = none)) ])])]).
```

Note the extensive use of the DT-Golog construct `pickBest( Value, Range, Program)` which picks the best value for `Program` from the range of possibilities. E.g., our program picks the best departure and return dates from the specified ranges (726 denotes July 26, etc.), and so on. In this framework the utility theory is specified by action costs (e.g., the cost of purchasing an airline ticket) and Markovian reward functions assigning real-valued rewards to situations. E.g.,

$$
\begin{aligned}
&reward(v, s) \equiv \\
&\quad (at(EDI, s) \wedge date(s) < 729 \vee date(s) > 805) \wedge v = 200) \vee \\
&\quad (\neg\, (at(EDI, s) \wedge (date(s) < 729 \vee date(s) > 805)) \wedge v = 0)
\end{aligned}
$$

This says that the reward, v is 200 if we are in Edinburgh before July 29 or after August 5, and v is 0 otherwise.

But Fiona cannot easily specify all her preferences as numeric Markovian rewards. A rich qualitative preference language that exploits temporal logic should help!

3 Preference Language

To personalize agent programs, we use a subset of a rich first-order language for expressing non-Markovian user preferences recently proposed in [**?**]. The semantics of this language is defined in the situation calculus.

3.1 Syntax

In this section we present the syntax of a first-order language for expressing qualitative, non-Markovian user preferences. Our language is a subset of the preference language we proposed in [**?**], which is a modification and extension of Son and Pontelli's *PP* language [**?**]. Constraints on the properties of situations are expressed by Basic Desire Formulae (BDF). BDFs are combined into Qualitative Preference Formulae[4], using a preference ordering, $\vec{\&}$.

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

1. $\mathcal{F} \cup \mathcal{R} \subset \mathcal{B}$, where $\mathcal{F}$ is the set of fluents and $\mathcal{R}$ is the set of non-fluent relations;
2. If $a \in \mathcal{A}$, the set of primitive actions, then $\textbf{occ}(a) \in \mathcal{B}$, stating that action a occurs;
3. If $f \in \mathcal{F}$, then $\textbf{final}(f) \in \mathcal{B}$;
4. If ψ, ψ_1, ψ_2 are in $\mathcal{B}$, then so are $\neg\psi$, $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$, conditional $\psi_1 : \psi_2$ (equivalent to $(\psi_1 \wedge \psi_2) \vee \neg\psi_1$), $(\exists x)\psi$, $(\forall x)\psi$, **next**(ψ), **always**(ψ), **eventually**(ψ), and **until**(ψ_1, ψ_2).

BDFs establish desired properties of situations. The first three BDF forms are evaluated with respect to the initial situation unless embedded in a temporal connective. By combining BDFs using boolean and temporal connectives, we are able to express a variety of properties of situations. In our travel example:

$$\begin{aligned}&\textbf{always}[(\exists y, z)(\text{bookedflight(y)} \wedge \text{arrivesLate}(y) \wedge \\ &\qquad \neg\text{closeToAirport}(z) : \neg\textbf{occ}(\text{bookhotel}(z)))] \qquad (1)\\ &\textbf{always}(\neg\, at(LHR)) \qquad (2)\end{aligned}$$

Again, BDFs enable a user to define preferred situations. To express preferences among alternatives, we define the notion of qualitative preference formulae.

Definition 2 (Qualitative Preference Formula)**.** Φ is a qualitative preference formula if one of the following holds:

- Φ is a basic desire formula
- $\Phi = \Psi_1 \vec{\&} \Psi_2$, with $\Psi_{1,2}$ qualitative preference formulae.

$\vec{\&}$ is an *Ordered And* preference. We wish to satisfy both Ψ_1 and Ψ_2, but if that is not possible, we prefer to satisfy Ψ_1 over Ψ_2. Note that this is enough to also express conditional preferences of the form "if a then I prefer b over c", as this can be transformed to $(a : b) \vec{\&} c$ which has the same semantics: if a holds, then I want to satisfy both b and c with a preference for b. If a does not hold, $a : b$ is immediately satisfied and it only remains to satisfy c. Qualitative preference formulae may be arbitrarily long.

[4] Subsequently referred to as *preference formulae*.

3.2 Semantics

Following our recent work [**?**], preference formulae are interpreted as situation calculus formulae and are evaluated relative to an action theory $\mathcal{D}$. Since BDFs may refer to properties that hold over fragments of a situation history, we use the notation $\varphi[s, s']$, proposed in [**?**], to explicitly denote that φ holds in the sequence of situations originating in s and terminating in $s' = do(\vec{a}, s)$. BDFs are interpreted in the situation calculus as follows:

$$\varphi \in \mathcal{F}, \varphi[s, s'] \stackrel{\text{def}}{=} \varphi[s]$$

$$\varphi \in \mathcal{R}, \varphi[s, s'] \stackrel{\text{def}}{=} \varphi$$

$$\textbf{final}(\varphi)[s, s'] \stackrel{\text{def}}{=} \varphi[s']$$

$$\textbf{occ}(a)[s, s'] \stackrel{\text{def}}{=} do(a, s) \sqsubseteq s' \wedge Poss(a[s], s)$$

$$\textbf{eventually}(\varphi)[s, s'] \stackrel{\text{def}}{=} (\exists s_1 : s \sqsubseteq s_1 \sqsubseteq s')\varphi[s_1, s']^5$$

$$\textbf{always}(\varphi)[s, s'] \stackrel{\text{def}}{=} (\forall s_1 : s \sqsubseteq s_1 \sqsubseteq s')\varphi[s_1, s']$$

$$\textbf{next}(\varphi)[s, s'] \stackrel{\text{def}}{=} (\exists a).do(a, s) \sqsubseteq s' \wedge \varphi[do(a, s), s']$$

$$\begin{aligned}\textbf{until}(\varphi, \psi)[s, s'] \stackrel{\text{def}}{=} (\exists s_2 : s \sqsubseteq s_2 \sqsubseteq s')\{\psi[s_2, s'] \;\wedge \\ (\forall s_1 : s \sqsubseteq s_1 \sqsubset s_2)\varphi[s_1, s']\}\end{aligned}$$

The boolean connectives are already defined in the situation calculus. Since each BDF is shorthand for a situation calculus expression, a simple model-theoretic semantics follows.

Definition 3. Let $\mathcal{D}$ be an action theory, and let s and s' be two situations such that $s \sqsubseteq s'$. A basic desire formula φ is satisfied by the situation beginning in s and terminating in s' just in the case that $\mathcal{D} \models \varphi[s, s']$.

Intuitively a qualitative preference formula $\Phi = \Psi_1 \vec{\&} \Psi_2$ partitions the space of situations into four equivalence classes of preferred situations, in decreasing order of preference: (1) those satisfying both Ψ_1 and Ψ_2, (2) those only satisfying Ψ_1, (3) those only satisfying Ψ_2, and (4) those satisfying neither. The semantics of qualitative preference formulae are defined in a subsequent section using Golog constructs. Their semantics follows from the semantics of Golog.

4 Adding Preferences to DT-Golog

BDFs are the building blocks of our qualitative preference formulae. Like Golog programs, BDFs impose constraints on situations. As such, it is natural to integrate BDFs into Golog by translating them into (generally non-deterministic) Golog programs. Preference over the enforcement of BDFs is expressed by qualitative preference formulae. These preferences can be realized in Golog by the multi-program synchronization

[5] Temporal formulae follow [**?**], using the abbreviations:
$(\exists s_1 : s \sqsubseteq s_1 \sqsubseteq s')\Phi \equiv (\exists s_1)\{s \sqsubseteq s_1 \wedge s_1 \sqsubseteq s' \wedge \Phi\}$ and $(\forall s_1 : s \sqsubseteq s_1 \sqsubseteq s')\Phi \equiv (\forall s_1)\{[s \sqsubseteq s_1 \wedge s_1 \sqsubseteq s'] \supset \Phi\}$

of BDF-induced Golog programs with the original agent program, and by prioritized execution of the resultant nondeterministic programs in a manner consistent with the defined preferences.

Synchronization of BDF-induced Golog programs with DT-Golog programs [**?**] results in a natural integration of agent programming under both qualitative preferences and quantitative utility theory. Since qualitative and quantitative expressions of preference are not immediately comparable, one has to decide how to rank them in case they are contradictory, i.e. favour different plans. In this paper we rank qualitative preferences over quantitative ones. As a result, we first try to find the quantitatively best plan within the set of most preferred plans, and only if no such plan exists, broaden our scope to less qualitatively preferred plans. Nevertheless, a different ordering or even several 'layers' would be easy to realize in the presented framework.

The outline of our approach is as follows: (1) compile BDFs into Golog programs such that any successful execution of that program will result in a situation that satisfies the BDF, (2) define multi-program synchronization to couple the execution of two programs so as to combine a given agent program with the compilation result, (3) based on this, define preferences over different subprograms.

4.1 Compilation

This section describes how we compile BDFs into Golog programs. The compilation works by progression up to a given horizon. At each progression step, the mechanism produces a set whose elements consist of a possible program step that can be performed without violating the BDF, and a possibly modified BDF that remains to be satisfied. Recursively these remaining BDFs are processed. As a progression step may return more than one branch (program-step/remaining-formula combination), compilation produces a tree, where branches are linked using nondeterministic choice. This tree describes the set of all possible program traces, i.e. situations of the situation calculus, that satisfy the BDF.

Example 1. Consider the following BDF: $\textbf{always}(\text{happy}) \wedge \textbf{final}(\text{rich})$ and assume $\mathcal{A}$ is a list of all primitive actions in our domain theory. Then the following program describes all possible sequences of length ≤ 2 that satisfy this BDF:

$$\begin{aligned}&\textbf{nondet}([(\text{happy} \wedge \text{rich})?,\\&\quad[\text{happy}?; \textbf{nondet}(\mathcal{A}); \textbf{nondet}([(\text{happy} \wedge \text{rich})?,\\&\qquad[\text{happy}?; \textbf{nondet}(\mathcal{A}); (\text{happy} \wedge \text{rich})?]\,])\,]])\end{aligned}$$

That is, either I am happy and rich already, or I am happy, take some action and then am happy and rich, or again I am happy and take another step. In the end I always have to be happy and rich. Any successful execution of this Golog program will satisfy the BDF.

Again, BDFs define desired properties of situations. As such, the maintenance of BDFs restricts the set of actions that may be taken in a situation. This insight is key to our compilation approach. We call the constraints required to enforce our BDFs *situation constraints*. We express a situation constraint in Golog by a test φ? that enforces a fluent/nonfluent and/or a nondeterministic choice of the actions available in the current situation. In many cases, this is all actions, $\mathcal{A}$.

Recall that in Golog $\varphi?$ states that the formula φ has to hold in the current situation and that $\textbf{nondet}(L)$ is the non-deterministic choice among the elements of the list L. For example, the only possible next steps for $\textbf{nondet}([a, b])$ are taking action a or taking action b. Thus, assuming the current situation is s, the set of possible successor situations are restricted to $\{do(a, s), do(b, s)\}$. The scope of situation constraints can be expanded over several situations by using temporal expressions. In the example, the constraint of being happy is extended over all situations using **always**. Observe that several BDFs are contributing situation constraints to the same situation. To combine several situation constraints we define the function χ. Note that the BDFs ψ are treated as syntactic entities in the context of our compilation and are syntactically manipulated accordingly.

- $\chi(\psi_1?, \psi_2?) = (\psi_1 \wedge \psi_2)?$
- $\chi(\psi?, \textbf{nondet}(L)) = (\psi?; \textbf{nondet}(L))$
- $\chi(\textbf{nondet}(L_1), \textbf{nondet}(L_2)) = \textbf{nondet}(L_1 \cap L_2)$
- $\chi((\psi_1?; \textbf{nondet}(L_1)), (\psi_2?; \textbf{nondet}(L_2)))$
  $= ((\psi_1 \wedge \psi_2)?; \textbf{nondet}(L_1 \cap L_2))$

plus its reflexive completion, where the ψ's are formulae of the situation calculus and the L's are lists of actions. In our example, the temporal extent of **always** and **final** overlap. In these situations, the situation constraints imposed by the two BDFs are combined using χ.

Let $\mathcal{A}$ be the set of actions in our domain, $\mathcal{F}$ the set of fluents, $\mathcal{R}$ the set of non-fluent predicates, then, formally the compilation of a basic desire formula ψ is defined using the predicate $\mathcal{C}$: $\mathcal{C}(\psi, SC, \psi')$ holds iff SC is a situation constraint whose execution will not violate preference ψ, and further ψ' is a BDF that needs to be satisfied in the future. In the following we use STOP as a shorthand for $\nexists a.\textbf{occ}(a)$. $\mathcal{C}$ is defined by the following set of axioms.

- $\mathcal{C}(f, f?, \text{TRUE}), \forall f \in \mathcal{F} \cup \mathcal{R}$
- $\mathcal{C}(\textbf{occ}(a), \textbf{nondet}([a]), \text{TRUE}), \forall a \in \mathcal{A}$
- $\mathcal{C}(\textbf{final}(f), SC, \psi') \equiv$
  $(SC = (f?, \textbf{nondet}([])) \wedge \psi' = \text{STOP})$
  $\vee\ (SC = \textbf{nondet}(\mathcal{A}) \wedge \psi' = \textbf{final}(f))$[6]
- $\mathcal{C}(\psi_1 \wedge \psi_2, SC, \psi') \equiv$
  $\mathcal{C}(\psi_1, SC_1, \psi_1') \wedge \mathcal{C}(\psi_2, SC_2, \psi_2')$
  $\wedge\ SC = \chi(SC_1, SC_2) \wedge \psi' = \psi_1' \wedge \psi_2'$
- $\mathcal{C}(\psi_1 \vee \psi_2, SC, \psi') \equiv \mathcal{C}(\psi_1, SC, \psi') \vee \mathcal{C}(\psi_2, SC, \psi')$
- $\mathcal{C}(\psi_1 : \psi_2, SC, \psi') \equiv \mathcal{C}((\psi_1 \wedge \psi_2) \vee \neg\psi_1, SC, \psi')$
- $\mathcal{C}((\exists x)\psi, SC, \psi') \equiv \mathcal{C}(\bigvee_{c \in \mathbf{C}}(\psi^{c/x}), SC, \psi')$[7]

[6] $\textbf{nondet}([])$ states that no action may be taken. Together with the remaining BDF STOP, it enforces immediate program termination.

[7] 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, and $\mathbf{C}$ denotes the set of constants.

- $\mathcal{C}((\forall x)\psi, SC, \psi') \equiv \mathcal{C}(\bigwedge_{c\in\mathbf{C}}(\psi^{c/x}), SC, \psi')$
- $\mathcal{C}(\mathbf{next}(\psi), \mathbf{nondet}(\mathcal{A}), \psi)$
- $\mathcal{C}(\mathbf{always}(\psi), SC, \psi') \equiv$
  $(\mathcal{C}(\psi, SC, \psi'') \wedge \psi'\text{=STOP} \wedge\ (\psi''\text{=STOP} \vee \psi''\text{=TRUE}))$
  $\vee\ (\mathcal{C}(\psi \wedge \mathbf{next}(\mathbf{always}(\psi)), SC, \psi')$
- $\mathcal{C}(\mathbf{eventually}(\psi), SC, \psi') \equiv$
  $\mathcal{C}(\psi \vee \mathbf{next}(\mathbf{eventually}(\psi)), SC, \psi')$
- $\mathcal{C}(\mathbf{until}(\psi_1, \psi_2), SC, \psi') \equiv$
  $\mathcal{C}(\psi_2 \vee (\psi_1 \wedge \mathbf{next}(\mathbf{until}(\psi_1, \psi_2))), SC, \psi')$
- $\mathcal{C}(\text{TRUE}, SC, \text{TRUE}) \equiv$
  $SC = \mathbf{nondet}([]) \vee SC = \mathbf{nondet}(\mathcal{A})$

Negation requires special treatment. Golog finds situations, i.e. action sequences, that satisfy a program, but to address negation it is not obvious how the complement, that is the situations that do *not* satisfy the program, would be computed. We address this by pushing the negation down to the atomic level. For parsimony we only show some less obvious cases:

- $\mathcal{C}(\neg f, (\neg f)?, \text{TRUE}), \forall f \in \mathcal{F} \cup \mathcal{R}$
- $\mathcal{C}(\neg\mathbf{occ}(a), SC, \psi') \equiv$
  $(SC = \mathbf{nondet}([]) \wedge \psi' = \text{STOP})$
  $\vee\ (SC = \mathbf{nondet}(\mathcal{A} \setminus \{a\}) \wedge \psi' = \text{TRUE}), \forall a \in \mathcal{A}.$
- $\mathcal{C}(\neg\mathbf{always}(\psi), SC, \psi') \equiv \mathcal{C}(\mathbf{eventually}(\neg\psi), SC, \psi')$
- $\mathcal{C}(\neg\mathbf{until}(\psi_1, \psi_2), SC, \psi') \equiv$
  $\mathcal{C}((\neg\psi_2 \wedge (\neg\psi_1 \vee \mathbf{next}(\neg\mathbf{until}(\psi_1, \psi_2))))$
  $\vee\ \mathbf{always}(\neg\psi_2)), SC, \psi')$

Based on $\mathcal{C}$ we can define the following (second-order) formula that relates a BDF ψ to a Golog program P such that every successful execution of P results in a situation that satisfies ψ where h is the maximal number of actions in any such execution.

$$
\begin{aligned}
\Xi(\psi, P, h) \equiv\, & (\psi = \text{TRUE} \wedge P = (\mathbf{nondet}(\mathcal{A}))^*) \\
& \vee (\psi = \text{STOP} \wedge P = nil) \\
& \vee (h = 0 \wedge \exists x.\mathcal{C}(\psi, P, x) \wedge \exists \varphi. P = ?(\varphi)) \\
& \vee (h > 0 \wedge \psi \neq \text{TRUE} \wedge \psi \neq \text{STOP} \wedge \mathcal{C}(\psi, SC, \psi') \ \wedge \\
& \quad \Xi^*(\psi', \mathcal{P}, h-1) \wedge P = SC; \mathbf{nondet}(\mathcal{P})\ ) \\
\Xi^*(\psi, \mathcal{P}, h) \equiv\, & \mathcal{P} = \{\ P \ \mid\ \Xi(\psi, P, h)\ \}
\end{aligned}
$$

A constructive proof of $\exists\mathcal{P}.\Xi^*(\psi, \mathcal{P}, h)$ then, as a side-effect, provides the program $P_\psi^h = \mathbf{nondet}(\mathcal{P})$ that describes all possible execution traces, i.e. situations, of length $\leq\ h$ that satisfy the BDF. These definitions lead to a Prolog implementation, able to conduct the constructive proof, producing the corresponding Golog program (cf. Section 5). Some optimization of the generated code is advisable, but for parsimony we omit the rather technical details of this here.

Soundness The soundness of our compilation method follows from the semantics of our preference language.

Theorem 1. (Soundness) Let ψ be a basic desire formula and P_ψ^h be the corresponding program for horizon h. Then for any situation $s_h = do([a_1, a_2, \dots, a_h], s)$ such that $\mathcal{D} \models Do(P_\psi^h, s, s_h)$, it holds that $\mathcal{D} \models \psi[s, s_h]$.
Proof Sketch: The proof proceeds by double induction over the structure of basic desire formulae and the length of the situation term. The base case for the structural induction is:

- $f \in \mathcal{F}$: as we have $\mathcal{C}(f, ?(f), \text{TRUE})$ and by hypothesis know that $\mathcal{D} \models Do(P_\psi^h, s, s_h)$ we have from the definition of Do (Golog semantics) that $f[s]$ and thus $f[s, s_h]$;
- $\mathbf{occ}(a)$: $\mathcal{C}(\mathbf{occ}(a), \mathbf{nondet}([a]), \text{TRUE})$ enforces that $a_1 = a$ and thus $\mathbf{occ}(a)[s, s_h]$;
- $\neg f \in \mathcal{F}$: as we have $\mathcal{C}(f, (\neg f)?, \text{TRUE})$ and by hypothesis know that $\mathcal{D} \models Do(P_\psi^h, s, s_h)$ we have from the definition of Do that $\neg f[s]$ and thus $\neg f[s, s_h]$;
- $\neg\mathbf{occ}(a)$: $\mathcal{C}(\mathbf{occ}(a), \mathbf{nondet}(\mathcal{A} \setminus \{a\}), \text{TRUE})$ enforces that $a_1 \neq a$ and thus $\neg\mathbf{occ}(a)[s, s_h]$;

Completeness Completeness likewise follows from the semantics of our preference language. This establishes that all situations that satisfy the BDF are preserved.

Theorem 2. (Completeness) Let ψ be a basic desire formula and P_ψ^h be the corresponding program for horizon h. Then for any situation $s_h = do([a_1, a_2, \dots, a_h], s)$ such that $\mathcal{D} \models \psi[s, s_h]$ it holds that $\mathcal{D} \models Do(P_\psi^h, s, s_h)$.
Proof Sketch: The proof is established by induction.

Details of both proofs are presented in [**?**].

4.2 Multi-Program Synchronization

Now that we have a Golog program enforcing satisfaction of a BDF, we want to combine this with a pre-existing agent program or another BDF-induced program to eventually provide a semantics for our qualitative preference formulae. To this end, we define multi-program synchronization.

Roughly, we understand two programs to execute synchronously if they traverse the same sequence of situations. Thus, at each step we need to find a common successor situation for both programs. This can be done efficiently by determining the successors of both individually and then intersecting the results. It is however *not* efficient if both programs are evaluated completely first. This motivates the use of a transition semantics as opposed to the evaluation semantics originally used to define DT-Golog.

A transition semantics for Golog was first introduced in [**?**] where, for the same reasons as above, it was used to define the concurrent execution of two programs. Roughly, a transition semantics is axiomatized through two predicates $\mathit{Trans}(\sigma, s, \sigma', s')$ and $\mathit{Final}(\sigma, s)$. The former defines for a program σ and a situation s the set of possible successor configurations (σ', s') according to the action theory. The later defines whether a program is *final*, i.e. successfully terminated, in a certain situation. For instance, for the program $a_1; a_2$, that is the sequence of actions a_1 and a_2, and a situation s, $\mathit{Trans}(a_1; a_2, s, a_2, do(a_1, s))$ describes the only possible transition and is only possible, if the action a_1 is possible in situation s according to the action theory. Using the transitive closure of *Trans*, denoted *Trans**, one can define a new Do predicate as follows:

$Do(\delta, s, s') \stackrel{\text{def}}{=} \exists \delta'.\mathit{Trans}^*(\delta, s, \delta', s') \wedge \mathit{Final}(\delta', s')$.

As is shown in [**?**], this definition is equivalent to the original Do. Thus, all results for the one semantics hold equally for the other.

In transition semantics we can formally define the synchronization of two programs σ_1, σ_2 by a new Golog construct **sync**(σ_1, σ_2):

$$\begin{aligned}
&\mathit{Trans}(\ \mathbf{sync}(\sigma_1, \sigma_2),\ s,\ \mathbf{sync}(\sigma_1', \sigma_2'),\ s'\ ) \equiv \\
&\qquad (\mathit{Trans}(\sigma_1, s, \sigma_1', s') \wedge \mathit{Trans}(\sigma_2, s, \sigma_2', s')) \\
&\qquad \vee (s' = s \wedge (\ (\mathit{Trans}(\sigma_1, s, \sigma_1', s) \wedge \sigma_2' = \sigma_2) \\
&\qquad\qquad\qquad \vee (\sigma_1' = \sigma_1 \wedge \mathit{Trans}(\sigma_2, s, \sigma_2', s))\ ) \\
&\mathit{Final}(\ \mathbf{sync}(\sigma_1, \sigma_2),\ s\ ) \equiv \mathit{Final}(\sigma_1, s) \wedge \mathit{Final}(\sigma_2, s)
\end{aligned}$$

The program **sync**(σ_1, σ_2) can perform a transition in a situation s to a new situation s' iff both programs σ_1 and σ_2 can perform a transition to s' or when $s' = s$ and one of σ_1 and σ_2 can do a transition that does not affect the situation, for example evaluating a test. In both cases, the program that remains to be run will be the synchronous execution of the two remaining subprograms (σ_1', σ_2'). To synchronize more than two programs we can use nesting, so for instance **sync**$(\sigma_1,$ **sync**$(\sigma_2, \sigma_3))$ would synchronize three programs.

The following theorem follows immediately from above definitions.

Theorem 3. Let σ_a, σ_b be two Golog programs. Then for any S', $\mathcal{D} \models Do(\sigma_a, S_0, S')$ and $\mathcal{D} \models Do(\sigma_b, S_0, S')$ if and only if $\mathcal{D} \models Do(\mathbf{sync}(\sigma_a, \sigma_b), S_0, S')$.

The theorem states that if there is a situation S' that describes a legal execution in both programs starting in S_0, then this is also a legal execution for the synchonization of the two programs. Further, the inverse also holds, saying that any legal execution of the synchonization is also legal for the two individual programs.

4.2.1 A Decision-Theoretic Transition Semantics

As stated above, DT-Golog is defined using an evaluation semantics and that does not suit our requirements. Thus, we have to redefine DT-Golog in an equivalent transition semantics, or, seen differently, extend the available transition semantics to decision-theoretic planning. The semantics follows intuitively from the established relationship between the two semantics. In this section we provide an overview of our new DT-Golog transition semantics. Unfortunately, space precludes us from stating all but an example of the necessary definitions:

$$\begin{aligned}
&\mathit{BestTrans}(\ [\mathbf{nondet}([\sigma]) \mid \sigma'], s, d, \pi, v, prob, \mathcal{B}, \mathcal{D}) \equiv \\
&\qquad \mathcal{B} = [(\ [\sigma|\sigma'], s, d, [\pi, v, prob])] \wedge \mathcal{D} = [] \\
&\mathit{BestTrans}(\ [\mathbf{nondet}([\sigma_1 \mid \sigma_2]) \mid \sigma'], s, d, \pi, v, prob, \mathcal{B}, \mathcal{D}) \equiv \\
&\qquad \mathit{BestTrans}(\ [\mathbf{nondet}(\sigma_2) \mid \sigma'], s, d, \pi_2, v_2, prob_2, \mathcal{B}_2, \mathcal{D}_2) \wedge \\
&\qquad \mathcal{B} = [(\ [\sigma_1|\sigma'], s, d, [\pi_1, v_1, prob_1]) \mid \mathcal{B}_2] \wedge \\
&\qquad \mathcal{D} = [([\pi, v, prob], [\pi_1, v_1, prob_1], [\pi_2, v, prob_2]) \mid \mathcal{D}_2]
\end{aligned}$$

where $\pi_1, v_1, prob_1, \pi_2, v_2, prob_2$ are new variables, s is the situation, d the recursion depth, π the policy, v the value, and $prob$ the termination probability. $\mathcal{B}$ is the list of possible successor configurations/branches and $\mathcal{D}$ a list of decisions to be made once all contained values have been determined. Roughly, the tuple $([\pi, v, prob], [\pi_1, v_1, prob_1], [\pi_2, v, prob_2])$ says that $[\pi, v, prob]$ is equal to $[\pi_1, v_1, prob_1]$ or $[\pi_2, v_2, prob_2]$ depending on which is better. However, this cannot be decided until both branches have been evaluated.

4.3 Expressing Preference in DT-Golog

In previous sections we showed how to compile BDFs into *hard* constraints, realized as Golog programs. To make these constraints *soft* and to rank these constraints to eventually create ordered preferences we need to introduce two more Golog constructs:

• **withPref**$(\sigma_{prog}, P^h_\psi)$: run program σ_{prog} and try to synchronously run P^h_ψ, the result of compiling BDF ψ. This is implemented by creating two branches one with the remaining program **sync**$(\sigma_{prog}, P^h_\psi)$ and one with σ_{prog}. We devise the interpreter so that the first branch will is explored first. Only if it fails is the second branch explored.

• **pref**$(P^h_{\psi_1}, P^h_{\psi_2})$: Let ψ_1, ψ_2 be two BDFs and $P^h_{\psi_1}$, $P^h_{\psi_2}$ their corresponding Golog programs as acquired by compilation. Then **pref**$(P^h_{\psi_1}, P^h_{\psi_2})$ gives semantics to the qualitative preference formula $\psi_1 \vec{\&} \psi_2$ by creating three branches of decreasing preference: **sync**$(P^h_{\psi_1}, P^h_{\psi_2})$, $P^h_{\psi_1}$, and $P^h_{\psi_2}$. Again, the later branches are only explored if no plan is found for the first. Formally this intuition is captured by extending *BestTrans* such that it defines clusters of branches (and corresponding decisions) of equal degree of preference. Then all previously seen Golog constructs return exactly one cluster of (possibly multiple) branches and the above two constructs return two, respectively three clusters:

$$
\begin{aligned}
&\mathit{BestTrans}(\ [\mathbf{withPref}(\sigma_{prog}, P^h_\psi) \mid \sigma'], s, d, \pi, v, prob,\\
&\qquad [C_1, C_2], [[\,], [\,]]) \equiv\\
&\quad C_1 = [([\mathbf{sync}(\sigma_{prog}, P^h_\psi) \mid \sigma'], s, d, [\pi, v, prob])] \wedge\\
&\quad C_2 = [([\sigma_{prog} \mid \sigma'], s, d, [\pi, v, prob])]\\
&\mathit{BestTrans}(\ [\mathbf{pref}(P^h_{\psi_1}, P^h_{\psi_2}) \mid \sigma'], s, d, \pi, v, prob,\\
&\qquad [C_1, C_2, C_3], [[\,], [\,], [\,]]) \equiv\\
&\quad C_1 = [([\mathbf{sync}(P^h_{\psi_1}, P^h_{\psi_2}) \mid \sigma'], s, d, [\pi, v, prob])] \wedge\\
&\quad C_2 = [([P^h_{\psi_1} \mid \sigma'], s, d, [\pi, v, prob])] \wedge\\
&\quad C_3 = [([P^h_{\psi_2} \mid \sigma'], s, d, [\pi, v, prob])]
\end{aligned}
$$

The preference over former clusters is formally defined in the evaluation strategy of clusters and branches:

$$
\begin{aligned}
&\textit{BestTrans}^*(B,h) \equiv B = (\sigma, s, d, [\pi, v, prob]) \wedge \\
&\quad (d \leq h \wedge \textit{BestTrans}(\sigma, s, d, \pi, v, prob, \vec{\mathcal{B}}, \vec{\mathcal{D}}) \wedge \\
&\quad \textit{BestTrans}^*_{\text{quali}}(\vec{\mathcal{B}}, \vec{\mathcal{D}}, prob, h)) \\
&\quad \vee (d > h \wedge \pi = nil \wedge Reward(r)[s] \wedge v = r \wedge prob = 1) \\
&\textit{BestTrans}^*_{\text{quali}}([\mathcal{B}], [\mathcal{D}], prob, h) \equiv \\
&\quad \textit{BestTrans}^*_{\text{quanti}}(\mathcal{B}, h) \wedge \textit{MakeDecisions}(\mathcal{D}) \\
&\textit{BestTrans}^*_{\text{quali}}([\mathcal{B} \mid \vec{\mathcal{B}}], [\mathcal{D} \mid \vec{\mathcal{D}}], prob, h) \equiv \\
&\quad (\textit{BestTrans}^*_{\text{quanti}}(\mathcal{B}, h) \wedge \textit{MakeDecisions}(\mathcal{D}) \wedge prob > 0) \\
&\quad \vee \textit{BestTrans}^*_{\text{quali}}(\vec{\mathcal{B}}, \vec{\mathcal{D}}, prob, h) \\
&\textit{BestTrans}^*_{\text{quanti}}([\,], h) \equiv \text{TRUE} \\
&\textit{BestTrans}^*_{\text{quanti}}([B_1 \mid \mathcal{B}], h) \equiv \\
&\quad \textit{BestTrans}^*(B_1, h) \wedge \textit{BestTrans}^*_{\text{quanti}}(\mathcal{B}, h)
\end{aligned}
$$

In $\textit{BestTrans}^*_{\text{quali}}$ the clusters are evaluated starting with the first and proceeding with the next only if the termination probability $prob$ for the first cluster is zero. Thus, we first explore only that partition of the situation space that is most preferred and only if within this partition no valid plan is found, less preferred partitions are considered. The predicate *MakeDecisions*($\mathcal{D}$) makes decisions as described above, once all variables in $\mathcal{D}$ have been instantiated.

A Qualitative Preference Formulae Ψ is compiled by compiling the contained BDFs and replacing any occurence of $\vec{\&}$ by a **pref** construct. We denote the compilation by $\Omega(\Psi, P^h_\Psi, h)$.

Example 2. Consider the following Golog program for determining transprotation: $\sigma =$ **nondet**$([drive, plane])$ and the preference $\Psi =$ **final**$(eco)\ \vec{\&}$ **always**$(\neg($**occ**$(drive)))$[8]. Compilation with horizon one, $\Omega(\Psi, P^1_\Psi, 1)$, of the preference will provide us with the following Golog program P^1_Ψ:

pref(**nondet**$([eco?, [$**nondet**$([plane, drive]); eco?]])$,
  nondet$([nil,$ **nondet**$([drive])]))$

This will be combined with the original Golog program, outlining the rough planning problem, by **withPref**(σ, P^1_Ψ). When fed into our interpreter this will produce different clusters of decreasing preference, which will be explored for a solution one at a time. The most preferred is:

sync(**nondet**$([plane, drive])$,
 sync(**nondet**$([[eco?], [$**nondet**$([plane, drive]), eco?]])$,
  nondet$([nil,$ **nondet**$([plane])])))$

Let's assume that driving is the only economical means of transporation. Then obviously we cannot satisfy both desires and in fact after the only possible synchronized transition, doing $plane$, the test $eco?$ fails. We thus move on to the next best cluster:

sync(**nondet**$([plane, drive])$,

[8] eco is a fluent stating that the transportation is economical.

nondet$([[eco?], [$**nondet**$([plane, drive]), eco?]]))$.
There are two possible synchronized transitions here, either doing $plane$ or $drive$, however, only the second will let the test $eco?$ succeed and will thus be chosen.

The following corollary follows from the soundness and completeness of our compilation, Theorem 3, and the correctness of above decision-theoretic transition semantics.

Corollary 1. Let σ be an arbitrary Golog program, Ψ a qualitative preference formula, and $h \in \mathbb{N}$ a horizon. Let further P_{Ψ}^{h} be such that $\Omega(\Psi, P_{\psi}^{h}, h)$. Then any constructive proof of

$$\begin{aligned}\mathcal{D} \models \exists \pi, v, prob.& \\ &\mathit{BestTrans}^{*}((\mathbf{withPref}(\sigma, P_{\Psi}^{h}), S_0, 0, [\pi, v, prob]), h)\end{aligned}$$

as a side-effect returns a policy[9] π which has the following properties:
• any successful execution of π leads to a situation that is most preferred among all possible situations, i.e., the set of situations of length $\leq h$ which describe a legal execution trace for σ according to the action theory $\mathcal{D}$ and there is no situation s' in this set that is more preferred;
• π maximizes the expected reward according to the utility theory.

In other words, π is the best we can do with respect to satisfying the hard constraints in the first place, generating the most *qualitatively* preferred plan in second place, and finally maximizing the *quantitative* expected reward in third place.

5 Implementation and Application

As noted previously, we have implemented the approach reported in this paper as an extension to Readylog [**?**]. We have also turned our travel agency example into a working application by creating wrappers for the flight and hotel pages of Yahoo!-Travel. Recall the planning procedure from Section 2. The actions `searchFlight(From, To, OutDate, ReturnDate)` and `searchHotel(Destination, CheckinDate, CheckoutDate)` realize the querying and wrapping of the relevant Web pages[10].

With respect to the quality of the results generated from our implementation, our theoretical results and correctness of the implementation (which we do not prove) ensure that the travel plan generated is optimized with respect to a user's quantitative preferences, within the best realization of their qualitative preferences. No benchmarks exist for the empirical evaluation of our system, nor was it our objective to optimize our implementation. Nevertheless, as an illustration of the power of our system, we argue that our implementation enables a level of customization of travel planning (and more generally, agent programming), heretofore unattainable in an automated system. For

[9]i.e. a Golog program without any non-deterministic choices
[10]Technically speaking these are so-called *sensing actions*, but space preclude a thorough discussion of this issue. The interested reader is referred to the literature, e.g. [**?**].

example, in the described case, for each of the 9 date combinations there are over 90 hotels with about 5 room types each and 9 flights. To gather all relevant information, the system issues more than 800 queries to Yahoo!-Travel, considers 36450 combinations, and returns the most preferred travel plan. Manually this would not be feasible and existing systems, although allowing customization to a certain extent, cannot account for the complex preferences a customer may have. *We* now can! We intend to make this application available as a service at our website.

6 Summary and Discussion

Motivated by the need to personalize agent programs to meet individual users' preferences and constraints, we addressed the problem of integrating non-Markovian qualitative user preferences with quantitative decision-theoretic planning in Golog. We approached the problem by compiling preferences into Golog programs using a notion of multi-program synchronization which we introduced. This required the redefinition of DT-Golog using a transition semantics which, as a nice side-effect, enables the implementation of more efficient and any-time solution algorithms. We proved the soundness and completeness of our compilation. The resulting system is able to handle infinite state spaces and allows for an efficient programmatic restriction of planning tasks using Golog's procedural expressiveness. Also, this is, to the best of our knowledge, the first work on integrating qualitative and quantitative preferences for temporal reasoning. We implemented our approach, and as a demonstration of its utility developed a customizable travel planner for the Web. The results in this paper are applicable to both symbolic and decision-theoretic agent programming systems, and may be used not only for the personalization of agent programs, but also for the realization of defeasible control strategies for planning.

A Proofs

A.1 Lemmata

Lemma 1. Let $\mathcal{S}$ be the set of all situations in a given action theory. In the proofs we will exploit the following property of the BDF semantics: Let ψ_1, ψ_2 be two BDFs and let $S_1(s), S_2(s)$ be two sets of situations such that $S_i(s) = \{s' \in \mathcal{S} \mid \mathcal{D} \models \psi_i[s, s']\}$, i.e. the set of all situations that, rooting in s, satisfy the BDF. Then for any situation s' $\mathcal{D} \models (\psi_1 \wedge \psi_2)[s, s']$ iff $s' \in S_1(s) \cap S_2(s)$.

A.2 Soundness

In the following proofs we use the semantics of Golog as defined in [**?**]. Further we use $\star^h$ as a shorthand for the nondeterministic repetition of $nondet(\mathcal{A})$ with a maximum of h repetitions. If h is omitted, the repetition is of arbitrary length.

Intuitively, soundness states that any program execution will result in a situation that also satisfies the BDF.

Theorem 4. (Soundness) Let ψ be a basic desire formula and P_ψ^h be the corresponding program for horizon h. Then for any situation $s_n = do([a_1, a_2, \ldots, a_n], s)$[11] such that $\mathcal{D} \models Do(P_\psi^h, s, s_n)$ it holds that $\mathcal{D} \models \psi[s, s_n]$.[12]

Proof: The proof proceeds by double induction over the structure of basic desire formulae and the length of the situation term. The base cases are as follows:

- for the structural induction:
  - $f \in \mathcal{F} \cup \mathcal{R}$: as we have $\mathcal{C}(f, ?(f), \text{TRUE})$ thus $P_\psi^h = ?(f); \star^h$ and by hypothesis know that $\mathcal{D} \models Do(P_\psi^h, s, s_n)$ we have from the definition of Do (Golog semantics) $\mathcal{D} \models f[s]$ and thus $\mathcal{D} \models f[s, s_n]$;
  - $\mathbf{occ}(a)$: With $\mathcal{C}(\mathbf{occ}(a), \mathbf{nondet}([a]), \text{TRUE})$ we have $P_\psi^h = nondet([a]); \star^{h-1}$ which enforces that $a_1 = a$ and thus $\mathcal{D} \models \mathbf{occ}(a)[s, s_n]$;
  - $\neg f \in \mathcal{F} \cup \mathcal{R}$: similar as above we have $\mathcal{C}(f, (\neg f)?, \text{TRUE})$ and by hypothesis know that $\mathcal{D} \models Do(P_\psi^h, s, s_n)$ we have from the definition of Do that $\mathcal{D} \models \neg f[s]$ and thus $\mathcal{D} \models \neg f[s, s_n]$;
  - $\neg\mathbf{occ}(a)$: $\mathcal{C}(\neg\mathbf{occ}(a), SC, \psi') \equiv (SC = \mathbf{nondet}([]) \wedge \psi' = \text{STOP}) \vee (SC = \mathbf{nondet}(\mathcal{A} \setminus \{a\}) \wedge \psi' = \text{TRUE}), \forall a \in \mathcal{A}$, enforces that either $n = 0$, i.e. no action happens, or $a_1 \neq a$. In both cases a does not happen and thus $\mathcal{D} \models \neg\mathbf{occ}(a)[s, s_n]$;

  As these cases are independent of the situation s they equally hold for all $s_i, 0 \leq i \leq n$.

- for the induction over the situation term we consider only the final situation s_n: As from there no more actions take place, we are only to look at the case of horizon zero. The definition of Ξ for $h = 0$ has three possible cases:

$$\begin{aligned} \Xi(\psi, P, 0) \equiv\, & (\psi = \text{TRUE} \wedge P = (\mathbf{nondet}(\mathcal{A}))^*) \\ & \vee (\psi = \text{STOP} \wedge P = nil) \\ & \vee (\exists x. \mathcal{C}(\psi, P, x) \wedge \exists \varphi. P = ?(\varphi)) \end{aligned}$$

  - For $\psi = \text{TRUE}$ we have trivially $\mathcal{D} \models \text{TRUE}[s, s]$.
  - For $\psi = \text{STOP} = \nexists a.\mathbf{occ}(a)$ we have as a tautology $\mathcal{D} \models \nexists a.\mathbf{occ}(a)[s, s]$.
  - The third case follows from the base case of the structural induction.

For the induction step we assume the theorem holds for f, $\mathbf{occ}(a)$, $\neg f$, and $\neg\mathbf{occ}(a)$ for any situation $s_i, 0 \leq i \leq n$, as well as over all situations $s_m, j \leq m \leq n$, for a certain $j, 0 < j \leq n$. We show the step from atomic formulae (only comprised of $f \in \mathcal{F} \cup \mathcal{R}$ and $\mathbf{occ}(a)$ and their negation) to general BDFs and from situation s_j to s_{j-1}.

For a BDF ψ let SC_ψ be the situation constraint and ψ' the remaining formula as defined by $\mathcal{C}(\psi, SC_\psi, \psi')$. Then the induction step

[11] We use the notation $do([a_1, a_2, \ldots, a_n], s)$ as a shorthand for $do(a_n, do(\ldots, do(a_2, do(a_1, s))\ldots))$.

[12] Which includes the special case $s = s_0$.

- $\psi = \textbf{final}(f)$: then $SC_\psi = (f?, \textbf{nondet}([])) \wedge \psi' = \text{STOP}$ or $SC_\psi = \textbf{nondet}(\mathcal{A}) \wedge \psi' = \textbf{final}(f))$. As $Do(\textbf{nondet}([]), s, s_n)$ holds only for $s = s_n$ this can only be the case for $j - 1 = n$ for which the proposition immediately holds by induction hypothesis. In the second case, no situation constraints are raised for s_{j-1} and ψ' holds on $[s_j, s_n]$ by induction hypothesis.

- $\psi = \psi_1 \wedge \psi_2$: then $SC_\psi = \chi(SC_{\psi_1}, SC_{\psi_2})$ and by construction of χ and by induction hypothesis it follows that $\mathcal{D} \models \psi_1[s_{j-1}, s_n]$ and $\mathcal{D} \models \psi_2[s_{j-1}, s_n]$ and with Lemma 1 also $\mathcal{D} \models \psi[s_{j-1}, s_n]$.

- $\psi = \psi_1 \vee \psi_2$: then $SC_\psi = SC_{\psi_1}$ or $SC_\psi = SC_{\psi_2}$. By induction hypothesis it follows that $\mathcal{D} \models \psi_1[s_{j-1}, s_n]$ or $\mathcal{D} \models \psi_2[s_{j-1}, s_n]$ and thus the proposition.

- $\psi = \textbf{next}(\varphi)$: $SC_\psi = \textbf{nondet}(\mathcal{A})$ and $\psi' = \textbf{next}(\varphi)$. The proposition follows by induction hypothesis.

- $\psi = \textbf{always}(\varphi)$: then either $\mathcal{C}(\varphi, SC, \psi'') \wedge \psi'{=}\text{STOP} \wedge (\psi''{=}\text{STOP} \vee \psi''{=}\text{TRUE})$ or
$\mathcal{C}(\varphi \wedge \textbf{next}(\textbf{always}(\varphi)), SC, \psi')$. In the former case we know from $\mathcal{C}(\varphi, SC, \psi'')$ together with induction hypothesis that $\mathcal{D} \models \varphi[s_{j-1}, s_j]$ and from $\psi'{=}\text{STOP}$ that $j = n$. Thus it follows that $\mathcal{D} \models \psi[s_{j-1}, s_n]$. In the latter case by induction hypothesis we have $\mathcal{D} \models \varphi[s_{j-1}, s_n]$ and $\mathcal{D} \models \textbf{always}(\varphi)[s_j, s_n]$. It follows $\mathcal{D} \models (\forall s_1 : s_{j-1} \sqsubseteq s_1 \sqsubseteq s_n)\varphi[s_1, s_n]$.

- $\psi = \textbf{eventually}(\varphi)$: from $\mathcal{C}(\varphi \vee \textbf{next}(\textbf{eventually}(\varphi)), SC, \psi')$ we have together with induction hypothesis that either $\mathcal{D} \models \varphi[s_{j-1}, s_n]$ or $\mathcal{D} \models \textbf{eventually}(\varphi)[s_j, s_n]$. In either case we have $\mathcal{D} \models (\exists s_1 : s_{j-1} \sqsubseteq s_1 \sqsubseteq s_n)\varphi[s_1, s_n]$

- $\psi = \textbf{until}(\psi_1, \psi_2)$: thus either $\mathcal{C}(\psi_2, SC, \psi')$ or $\mathcal{C}(\psi_1 \wedge \textbf{next}(\textbf{until}(\psi_1, \psi_2)), SC, \psi')$. In the former case we know from induction hypothesis that $\mathcal{D} \models \psi_2[s_{j-1}, s_n]$ and thus $\mathcal{D} \models (\exists s_2 : s_{j-1} \sqsubseteq s_2 \sqsubseteq s_n)\{\psi[s_2, s_n] \wedge (\forall s_1 : s_{j-1} \sqsubseteq s_1 \sqsubset s_2)\varphi[s_1, s_n]\}$ holds with $s_2 = s_{j-1}$. In the latter case we get $\mathcal{D} \models \psi_1[s_{j-1}, s_n]$ and by induction hypothesis $\mathcal{D} \models \textbf{until}(\psi_1, \psi_2)[s_j, s_n]$. Thus it follows that $\mathcal{D} \models (\exists s_2 : s \sqsubseteq s_2 \sqsubseteq s')\{\psi[s_2, s'] \wedge (\forall s_1 : s \sqsubseteq s_1 \sqsubset s_2)\varphi[s_1, s]\}$.

Quantifiers and conditional are macros, defined based on the above constructs. For these the theorem follows from the equivalence in their definition in the preference semantics.

□

A.3 Completeness

Intuitively completeness says that the generated program minimally restricts the set of situations, i.e. no situation that satisfies the BDF is ruled out.

Theorem 5. (Completeness) Let ψ be a basic desire formula and P_ψ^h be the corresponding program for horizon h. Then for any situation $s_n = do([a_1, a_2, \ldots, a_n], s)$ with $n \leq h$ such that $\mathcal{D} \models \psi[s, s_n]$ it holds that $\mathcal{D} \models Do(P_\psi^h, s, s_n)$.

Proof: The proof again is established by induction over the structure of BDFs. First note that $\mathcal{D} \models Do(\star, s, s_n)$ holds for any situation $s_n = do([a_1, a_2, \ldots, a_n], s), n \geq 0$ of arbitrary actions a_i, where for $n = 0$ we understand $s_0 = s$. The base cases for the induction are:

- $f \in \mathcal{F} \cup \mathcal{R}$: By assumption we have $\mathcal{D} \models f[s, s_n]$ and thus by definition $\mathcal{D} \models f[s]$. Also by definition of Ξ and $\mathcal{C}$ we have $P^h_\psi = f?; \star$. Further $Do(f?; \star, s, s') \stackrel{\text{def}}{=} \exists s^*. Do(f?, s, s^*) \wedge Do(\star, s^*, s')$ is satisfied with $s^* = s$ and $s' = s_n$.
- **occ**(a): The assumption together with the definition of the semantics of BDFs entail $do(a, s) \sqsubseteq s_n \wedge Poss(a[s], s)$ and from compilation $P^h_\psi =$ **nondet**$(a); \star$ which is equivalent to $P^h_\psi = a; \star$. Again $Do(a; \star, s, s') \stackrel{\text{def}}{=} \exists s^*. Do(a, s, s^*) \wedge Do(\star, s^*, s')$ is satisfied with $s^* = s_1 = do(a, s)$ and $s' = s_n$.
- $\neg f \in \mathcal{F} \cup \mathcal{R}$: similar as above we have by assumption $\mathcal{D} \models \neg f[s, s_n]$ and by definition $\mathcal{D} \models \neg f[s]$. Also by definition of Ξ and $\mathcal{C}$ we have $P^h_\psi = \neg f?; \star$. Further $Do(\neg f?; \star, s, s') \stackrel{\text{def}}{=} \exists s^*. Do(\neg f?, s, s^*) \wedge Do(\star, s^*, s')$ is satisfied with $s^* = s$ and $s' = s_n$.
- $\neg$**occ**(a): By definition we have $do(a, s) \not\sqsubseteq s_n \vee \neg Poss(a[s], s)$ and either $P^h_\psi =$ **nondet**$(\mathcal{A} \setminus \{a\}); \star$ or $P^h_\psi =$ **nondet**$([\,]); nil$. For the former case $Do(\textbf{nondet}(\mathcal{A} \setminus \{a\}); \star, s, s') \stackrel{\text{def}}{=} \exists s^*. Do(\textbf{nondet}(\mathcal{A} \setminus \{a\}), s, s^*) \wedge Do(\star, s^*, s')$ is satisfied for any $s^* = s_1 = do(b, s)$ with $b \neq a$ and $s' = s_n$. The latter case is true for $n = 0$.

The induction step is as follows:

- $\psi =$ **final**(f): The program as described by compilation is of the form:

$$
\begin{aligned}
P^h_\psi = \textbf{nondet}([\quad & [f?; \textbf{nondet}([\,])], \\
& [\textbf{nondet}(\mathcal{A}); \textbf{nondet}([\quad [f?; \textbf{nondet}([\,])], \\
& \qquad\qquad\qquad\qquad\quad [\textbf{nondet}(\mathcal{A}); \ldots]\,])\,]\,])
\end{aligned}
$$

  which is a compact representation of:

$$
\begin{aligned}
\textbf{nondet}([\quad & [f?; \textbf{nondet}([\,])], \\
& [\textbf{nondet}(\mathcal{A}); f?; \textbf{nondet}([\,])], \\
& [\textbf{nondet}(\mathcal{A}); \textbf{nondet}(\mathcal{A}); f?; \textbf{nondet}([\,])], \\
& \vdots \\
& [\underbrace{\textbf{nondet}(\mathcal{A}); \ldots; \textbf{nondet}(\mathcal{A})}_{h}; f?; \textbf{nondet}([\,])]\ \ ])
\end{aligned}
$$

By assumption we know that $\mathcal{D} \models f[s_n]$ and from above we have that for any $n \leq h$ there is an alternative $\sigma_n = [\underbrace{\textbf{nondet}(\mathcal{A}); \ldots; \textbf{nondet}(\mathcal{A})}_{n}; f?; \textbf{nondet}([\,])]$.

In combination this implies $\mathcal{D} \models Do(\sigma_n, s, s_n)$ and further, by definition of $Do(\textbf{nondet}(\ldots), s, s')$, $\mathcal{D} \models Do(P^h_\psi, s, s_n)$.

- $\psi = \psi_1 \wedge \psi_2$: From induction hypothesis we know the proposition holds for ψ_1 and ψ_2, i.e. for any situation s_n as above such that $\mathcal{D} \models \psi_i[s, s_n]$ it also holds that for the corresponding program $P^h_{\psi_i}$ we have $\mathcal{D} \models Do(P^h_{\psi_i}, s, s_n)$. We can think of a program generated from our compilation as a tree whose nodes are situation constraints and any successful execution of the program is a path from the root to one of the leaves and describes a situation. Following Lemma 1, we are interested in the intersection of the sets of situations that describe successful executions of the individual programs $P^h_{\psi_1}, P^h_{\psi_2}$. This set can be desccribed by the conjunction of above mentioned trees. That is, starting at the root, we combine the situation constraints of the individual programs in all possible ways, thus creating a new tree. Any path from the root to one of the leaves in the new tree, will describe a situation which satisfies the cunjunction of the two BDFs ψ_1, ψ_2.

  Using Lemma 1 and induction hypothesis it is sufficient to show that any situation s_n that is a successful execution of both programs $P^h_{\psi_1}, P^h_{\psi_2}$ is also a successful execution of the combined program P^h_ψ. The axiom for compiling conjunction combines the situation constraints SC_1, SC_2 raised by compiling the two subformulas using the function χ and conjoining the remaining BDFs ψ'_1, ψ'_2. By case distinction, we show that if s_n satisfies the individual situation contrains, i.e. $\mathcal{D} \models \exists s'.Do(SC_i, s, s') \wedge s' \sqsubseteq s_n, i \in \{1, 2\}$, then it also satisfies the combined one:

  - $\chi(\psi_1?, \psi_2?) = (\psi_1 \wedge \psi_2)?$: By assumption we have $\mathcal{D} \models \exists s'.Do(\psi_1?, s, s') \wedge s' \sqsubseteq s_n$ and $\mathcal{D} \models \exists s'.Do(\psi_2?, s, s') \wedge s' \sqsubseteq s_n$. In both cases, by definition of Do, $s' = s$, which as a whole entails $\mathcal{D} \models \exists s'.Do(\psi_1?, s, s') \wedge Do(\psi_2?, s, s') \wedge s' \sqsubseteq s_n$ and thus again by definition of Do: $\mathcal{D} \models \exists s'.Do((\psi_1 \wedge \psi_2)?, s, s') \wedge s' \sqsubseteq s_n$.
  - $\chi(\psi?, \textbf{nondet}(L)) = (\psi?; \textbf{nondet}(L))$: By assumption we know $\mathcal{D} \models \exists s'.Do(\psi?, s, s') \wedge s' \sqsubseteq s_n$ and $\mathcal{D} \models \exists s''.Do(\textbf{nondet}(L), s, s'') \wedge s'' \sqsubseteq s_n$. From the definition of Do we know that $s' = s$. Thus $\mathcal{D} \models Do(\psi?, s, s) \wedge \exists s''.Do(\textbf{nondet}(L), s, s'') \wedge s'' \sqsubseteq s_n \equiv \exists s''.Do([\psi?; \textbf{nondet}(L)], s, s'') \wedge s'' \sqsubseteq s_n$, the proposition.
  - $\chi(\textbf{nondet}(L_1), \textbf{nondet}(L_2)) = \textbf{nondet}(L_1 \cap L_2)$: By assumption $\mathcal{D} \models \exists s'.Do(\textbf{nondet}(L_1), s, s') \wedge s' \sqsubseteq s_n$ and $\mathcal{D} \models \exists s''.Do(\textbf{nondet}(L_2), s, s'') \wedge s'' \sqsubseteq s_n$. Notice that there is only one $s' = do(a, s)$ such that $s' \sqsubseteq s_n$. Thus s' and s'' have to be identical and further a has to be in both L_1 and L_2. Thus we have that $\mathcal{D} \models \exists s'.Do(\textbf{nondet}(L_1 \cap L_2), s, s') \wedge s' \sqsubseteq s_n$.
  - $\chi((\psi_1?; \textbf{nondet}(L_1)), (\psi_2?; \textbf{nondet}(L_2))) = ((\psi_1 \wedge \psi_2)?; \textbf{nondet}(L_1 \cap L_2))$: See last item.

As we know s_n is a path in both trees, for $P^h_{\psi_1}$ and $P^h_{\psi_2}$, we can take the corresponding situation constraints along these paths and combine them as above. The combination will, as shown above, be satisfied by s_n and will be a path in the combined tree. It follows the proposition: $\mathcal{D} \models Do(P^h_\psi, s, s_n)$.

- $\psi = \psi_1 \vee \psi_2$: By induction hypothesis we know that either $\mathcal{D} \models Do(P^h_{\psi_1}, s, s_n)$ or $\mathcal{D} \models Do(P^h_{\psi_2}, s, s_n)$. Thus

  $$\mathcal{D} \models Do(P^h_{\psi_1}, s, s_n) \vee Do(P^h_{\psi_2}, s, s_n) \equiv Do(\mathbf{nondet}([P^h_{\psi_1}, P^h_{\psi_2}]), s, s_n)$$

  With $\mathbf{nondet}([P^h_{\psi_1}, P^h_{\psi_2}])$ being the compilation of ψ we get the proposition.

- $\psi = \mathbf{next}(\varphi)$: By assumption we have that $\mathcal{D} \models \mathbf{next}(\varphi)[s, s_n]$. From the semantics of **next** we further know: $\mathbf{next}(\varphi)[s, s_n] \stackrel{\text{def}}{=} (\exists a \in \mathcal{A}).do(a, s) \sqsubseteq s_n \wedge \varphi[do(a, s), s_n]$. By induction hypothesis we can assume $\mathcal{D} \models Do(P^{h-1}_\varphi, do(a, s), s_n)$. The compilation of ψ is defined as $P^h_\psi = [\mathbf{nondet}(\mathcal{A}); P^{h-1}_\varphi]$. Clearly $\mathcal{D} \models Do(\mathbf{nondet}(\mathcal{A}), s, do(a, s))$ for any action $a \in \mathcal{A}$. We thus get the proposition:

  $$\begin{aligned} \mathcal{D} \quad \models \quad & \exists a \in \mathcal{A}.Do(\mathbf{nondet}(\mathcal{A}), s, do(a, s)) \wedge Do(P^{h-1}_\varphi, do(a, s), s_n) \\ & \equiv Do([\mathbf{nondet}(\mathcal{A}); P^{h-1}_\varphi], s, s_n) \\ & \equiv Do(P^h_\psi, s, s_n). \end{aligned}$$

- $\psi = \mathbf{always}(\varphi)$: By assumption we know $\mathbf{always}(\varphi)[s, s_n] \stackrel{\text{def}}{=} (\forall s' : s \sqsubseteq s' \sqsubseteq s_n)\varphi[s', s_n]$. This is equivalent to $\varphi[s, s_n] \wedge \varphi[s_1, s_n] \wedge \cdots \wedge \varphi[s_n, s_n]$ with $s_i \sqsubseteq s_n$. This in turn is equivalent to $(\varphi[s, s_n] \wedge \mathbf{next}(\mathbf{always}(\varphi))[s, s_n]) \vee (\varphi[s, s] \wedge \neg\exists a.\mathbf{occ}(a)[s, s_n])$, where in the latter disjunct we have as a consequence $s = s_n$. The compilation is defined as $P^h_\psi = \mathbf{nondet}([P^h_1, P^h_2])$ with P^h_1 the compilation of $\varphi[s, s_n] \wedge \mathbf{next}(\mathbf{always}(\varphi))[s, s_n]$ and P^h_2 the compilation of $\varphi[s, s] \wedge \neg\exists a.\mathbf{occ}(a)[s, s_n]$. The proposition follows by induction hypothesis.

- $\psi = \mathbf{eventually}(\varphi)$: By assumption we have

  $$\begin{aligned} \mathcal{D} \quad & \models \quad \mathbf{eventually}(\varphi)[s, s_n] \\ & \stackrel{\text{def}}{=} \quad (\exists s_1 : s \sqsubseteq s_1 \sqsubseteq s_n)\varphi[s_1, s_n] \\ & \equiv \quad \varphi[s, s_n] \vee \mathbf{next}(\mathbf{eventually}(\varphi))[s, s_n]. \end{aligned}$$

  The compilation of $\mathbf{eventually}(\varphi)$ is defined as the compilation of $\varphi \vee \mathbf{next}(\mathbf{eventually}(\varphi))$. The proposition follows by induction hypothesis.

- $\psi = \mathbf{until}(\psi_1, \psi_2)$: By assumption we have $\mathcal{D} \models \mathbf{until}(\psi_1, \psi_2)[s, s_n]$ and by definition

  $$\begin{aligned} \mathbf{until}(\psi_1, \psi_2)[s, s_n] \quad & \stackrel{\text{def}}{=} \quad (\exists s_2 : s \sqsubseteq s_2 \sqsubseteq s_n)\{\psi_2[s_2, s_n] \wedge (\forall s_1 : s \sqsubseteq s_1 \sqsubset s_2)\psi_1[s_1, s]\} \\ & \equiv \quad \psi_2[s, s_n] \vee (\psi_1[s, s_n] \wedge \mathbf{next}(\mathbf{until}(\psi_1, \psi_2))[s, s_n]). \end{aligned}$$

  The compilation of $\mathbf{until}(\psi_1, \psi_2)$ is defined as the compilation of $\psi_2 \vee (\psi_1 \wedge \mathbf{next}(\mathbf{until}(\psi_1, \psi_2)))$. The proposition follows by induction hypothesis.

This completes our proof of completeness.

$\square$