Decision-Theoretic GOLOG with Qualitative Preferences

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

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 expressed in a linear temporal logic with quantitative Markovian reward functions. We begin with DTGOLOG, 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 DTGOLOG 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 quantitative preferences for agent programming. Further, while the focus of this paper is on the integration of qualitative and quantitative preferences, a side effect of this work is realization of the simpler task of integrating qualitative preferences alone into agent programming as well as the generation of GOLOG programs from LTL formulae.

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:

Example 1. *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, preferably 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).

In this paper, we develop a system that allows the user to express all these kinds of constraints and preferences. In particular 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 in agent programming. Then, parameterized, task-specific, non-deterministic agent programs can be developed for performing standard tasks, like travel planning over the Web, and we can personalize them with the proposed kinds of preferences and constraints. We do so by compiling qualitative temporal preferences into (extended-)DTGOLOG programs (Boutilier *et al.* 2000), a decision-theoretic variant of the agent programming language GOLOG. The resultant program maximizes the users expected utility within the set of 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 in an agent programming setting. There is of course extensive work on reasoning with quantitative dynamical preferences as exemplified by Markov Decision Processes (MDPs). There has also been significant work on *static* qualitative preferences, exemplified by the popular ceteris paribus CP-nets formalism (e.g., (Boutilier *et al.* 2004; Domshlak *et al.* 2003)). In contrast to our preference language, this language only deals with static preferences and does not yield

a total order over preferences as our preference language does. There has been relatively little work on the incorporation of qualitative preferences into dynamical systems (e.g., (Bienvenu, Fritz, & McIlraith 2006; Son & Pontelli 2004; Delgrande, Schaub, & Tompits 2004)).

In the next section we review the situation calculus and GOLOG. This is followed by a description of the preference language we use. In Section 4, we present our approach to integrating qualitative preferences into DTGOLOG. It comprises three steps: compilation of non-Markovian qualitative preferences into GOLOG programs; combining the resulting GOLOG programs with an existing GOLOG program via synchronization; and given this newly synchronized program, a means of expressing preferences over different possible subprograms. Included are a soundness and completeness result relating to our compilation, and a new transition semantics for DTGOLOG. We have implemented our approach as an extension to READYLOG(Ferrein, Fritz, & Lakemeyer 2004), an existing on-line decision-theoretic GOLOG interpreter. We demonstrate its utility with an application to personalized travel planning over the Web in Section 5. Prior to concluding, we discuss some related work.

2 Situation Calculus and GOLOG

The situation calculus is a logical language for specifying and reasoning about dynamical systems (Reiter 2001). In the situation calculus, the *state* of the world is expressed in terms of functions and relations (fluents) relativized to a particular *situation* s, e.g., $F(\vec{x}, s)$. In this paper, we 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 thus inducing a tree of situations rooted in S_0. Following convention we will generally refer to fluents in situation-suppressed form, e.g., $at(toronto)$ rather than $at(toronto, s)$.

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. Details of the form of these axioms can be found in (Reiter 2001).

GOLOG (Levesque *et al.* 1997) is a high-level logic programming language for the specification and execution of complex actions in dynamical domains. It builds on top of the situation calculus by providing Algol-inspired extralogical constructs for assembling primitive situation calculus actions into complex actions (*programs*) δ. Constructs include:

a — primitive actions
$\delta_1; \delta_2$ — sequences
$\phi?$ — tests
$(\pi x)\delta(x)$ — nondeterministic choice of arguments
δ^* — nondeterministic iteration
ndet(L) – nondeterministic choice of sub-program 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 (Reiter 2001) and a related single-step transition semantics that was proposed for on-line execution (De Giacomo, Lespérance, & Levesque 2000). 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]
$Do(?(\varphi), s, s') \stackrel{\text{def}}{=} \varphi[s] \wedge s = s'$.
$Do(\textbf{ndet}([\delta_1|\bar{\delta}], s, s') \stackrel{\text{def}}{=} Do(\delta_1, s, s') \vee Do(\textbf{ndet}(\bar{\delta}), s, s')$[3]
$Do(\textbf{ndet}([], s, s') \stackrel{\text{def}}{=} s = s'$

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 DTGOLOG 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 build upon a decision-theoretic variant of GOLOG called DTGOLOG (Boutilier *et al.* 2000), which extends GOLOG to deal with uncertainty in action outcomes and general reward functions. DTGOLOG 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. In particular we are using an on-line interpreter called READYLOG, which combines the advantages of several previous extensions and implementations. As with other on-line interpreters, READYLOG offers a special search-construct that invokes a switch to an off-line planning mode. This off-line planning determines the optimal

[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 remaining list r

policy (conditional plan) for a given non-deterministic program, i.e. it makes the non-deterministic choices based on the expected utility of the alternatives. The policy is then executed in the real world.

As an example, our travel planning problem could be described by the following READYLOG 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( bestF, foundFlights, bookFlight(bestF)),
    pickBest( bestH, foundHotels, bookHotel(bestH)) ]))]).
```

We use airport codes to represent locations: YYZ – Toronto, LHR – London Heathrow, EDI – Edinburgh. If enclosed in the above mentioned search-construct, the interpreter first finds the best conditional plan for the procedure, and then executes it in the real world. Note the extensive use of the READYLOG construct **pickBest**(Value, Range, Program) which picks the best value for Program from the range of possibilities Range relating to Value, a variable appearing in the program. 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.,
$reward(v, s) \equiv$

$(at(EDI, s) \wedge date(s) < 729 \vee date(s) > 805) \wedge v = 200) \vee$

$(\neg\,(at(EDI, s) \wedge (date(s) < 729 \vee date(s) > 805)) \wedge v = 0)$

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

But we cannot realistically expect the customers of a travel agency to provide us with all their preferences in this Markovian and numeric form. It seems more natural to also elicit qualitative preferences with temporal extent, as we exemplified in our earlier example. We will start by defining a suitable language for this purpose.

3 Preference Language

To personalize agent programs, we use a rich first-order language for expressing qualitative, non-Markovian user preferences. Our language is a subset of the preference language we proposed in (Bienvenu, Fritz, & McIlraith 2006), which is a modification and extension of Son and Pontelli's *PP* language (Son & Pontelli 2004). The semantics of this language is defined in the situation calculus.

3.1 Syntax

Constraints on the properties of situations are expressed by *basic desire formulae* (BDF):

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 **occ**$(a) \in \mathcal{B}$, stating that action a occurs;
3. If $f \in \mathcal{F}$, then **final**$(f) \in \mathcal{B}$;
4. If ψ, ψ_1, ψ_2 are in $\mathcal{B}$, then so are $\neg\psi$, $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$, $\psi_1 \rightarrow \psi_2[\equiv \neg\psi_1 \vee \psi_2$; conditional], $(\exists x)\psi$, $(\forall x)\psi$, **next**(ψ), **always**(ψ), **eventually**(ψ), and **until**(ψ_1, ψ_2).

BDFs establish desired properties of situations, which are action histories. 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:

$$\textbf{always}\big[(\exists y, z)((\text{flight(y)} \wedge \text{arrivesLate}(y) \wedge \neg\text{closeToAirport}(z)) \rightarrow \neg\textbf{occ}(\text{bookhotel}(z)))\big] \quad (1)$$

$$\textbf{always}(\neg\, at(\text{LHR})) \quad (2)$$

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 [4] (QPF))**.** Ψ is a qualitative preference formula if one of the following holds:

- Ψ is a Basic Desire Formula
- $\Psi = \psi \,\vec{\&}\, \Psi'$, with ψ a BDF and Ψ' another Qualitative Preference Formula.

$\vec{\&}$ is an *Ordered And* preference. We wish to satisfy both ψ and Ψ', but if that is not possible, we prefer to satisfy ψ over Ψ'. 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 represented by $(a \rightarrow b) \,\vec{\&}\, (a \rightarrow c)$ which has the intended semantics.

Note that every QPF can be expanded to the ordered-and of its BDFs: $\Psi = \psi_1 \,\vec{\&}\, \psi_2 \,\vec{\&}\, \ldots \,\vec{\&}\, \psi_n$.

Throughout this paper we will follow the notational conventions of using the symbol ψ for BDFs, Ψ for QPFs, and δ for GOLOG programs.

3.2 Semantics

Following recent work (Bienvenu, Fritz, & McIlraith 2006), 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 $\psi[s, s']$, proposed in (Gabaldon 2004), to explicitly denote that ψ holds on the sequence of situations originating in s and terminating in $s' = do(\vec{a}, s)$. Note that this does not say that ψ holds on all situations in this interval, nor that it only holds in one. It says that all temporal expressions occurring in ψ are limited to this interval. If no temporal expressions are used, then that means that ψ holds on the first situation s.

As noted previously, fluents are represented in situation-suppressed form and $f[s]$ denotes the re-insertion of situation terms s in fluent f. BDFs are interpreted in the situation calculus as follows:

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

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

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

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

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

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

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

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

$\textbf{until}(\psi_1,\psi_2)[s,s'] \stackrel{\text{def}}{=} (\exists s_2 : s \sqsubseteq s_2 \sqsubseteq s').$
$\psi_2[s_2,s'] \wedge (\forall s_1 : s \sqsubseteq s_1 \sqsubset s_2)\psi_1[s_1,s']$

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 feasible situations (all contained actions are possible) such that $s \sqsubseteq s'$. A BDF ψ is satisfied on the interval of situations s to s' if and only if $\mathcal{D} \models \psi[s,s']$. We say ψ is satisfied by situation s if it is satisfied on $[S_0, s]$.

Semantics of Qualitative Preference Formulae

Let $\Psi = \psi_1 \vec{\&} \psi_2 \vec{\&} \ldots \vec{\&} \psi_n$ be a qualitative preference formula, ψ_i be BDFs, and let $index_s([\psi_1,..,\psi_n])$ denote the index of the first BDF in $\psi_1,..,\psi_n$ that is not satisfied by situation s or $n+1$ if no such index exists. Situation s is preferred to situation s' w.r.t. to Ψ iff $index_s([\psi_1,..,\psi_n]) > index_{s'}([\psi_1,..,\psi_n])$, or $index_s([\psi_1,..,\psi_n]) = index_{s'}([\psi_1,..,\psi_n]) = i$ and s is preferred to s' with respect to $\Psi' = \psi_{i+1} \vec{\&} \ldots \vec{\&} \psi_n$. That is, we aim to satisfy the longest possible prefix of BDFs composing Ψ.

4 Adding Preferences to DTGOLOG

Recall that BDFs are the building blocks of our qualitative preference formulae. Further note that BDFs impose constraints on situations just as GOLOG programs do. Thus, to integrate a QPF into a DTGOLOG program, we will first transform its constituent BDFs into small GOLOG programs and combine them back together to reflect the preference semantics of the QPF. To combine multiple GOLOG programs, we define the notion of multi-program synchronization, and to reflect the relative preference of one GOLOG program over another, we define their strict preference in DTGOLOG.

Synchronization of preference-induced GOLOG programs with DTGOLOG 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 try to find the quantitatively best plan within the set of most qualitatively 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. We compile each of the BDFs comprising our QPF into a GOLOG program such that any successful execution of the resulting program will result in a situation that satisfies the BDF that generated it;
2. We then define multi-program synchronization to couple the execution of multiple programs so as to intersect the sets of situations the programs describe;
3. Based on this, we define preferences over different subprograms both to rank several preferences and to add defeasible preferences to a given program.

4.1 Compiling BDFs into GOLOG programs

This section describes how we compile BDFs, the building blocks of our QPF into GOLOG programs. The compilation works by progression up to a given horizon. At each progression step, the mechanism produces a set of tuples consisting of 1) a program step that can be performed without violating the BDF, and 2) the BDF that remains to be satisfied. We continue to progress the latter until we arrive at the trivial BDF (TRUE) or reach the compilation horizon. As a progression step may produce more than one program-step/remaining-formula combination, (branch), compilation produces a tree. The branches are linked using nondeterministic choice. This tree describes the set of all situations that satisfy the BDF under the given domain theory.

Example 2. Consider the BDF

$$\textbf{always}(\neg\text{at(LHR)}) \wedge \textbf{eventually}(\textbf{occ}(\text{sight-seeing}))$$

and assume $\mathcal{A}$ is the list of all primitive actions in our domain theory. Then the following program describes all possible sequences of length ≤ 2 that satisfy this BDF:

$$\delta^2_\psi = \textbf{ndet}(\ [\ \neg\text{at(LHR)?; sight-seeing; } \neg\text{at(LHR)?;}$$
$$\textbf{ndet}([\ nil,\ \textbf{ndet}(\mathcal{A}); \neg\text{at(LHR)?}\ ]),$$
$$\neg\text{at(LHR)?; } \textbf{ndet}(\mathcal{A}); \neg\text{at(LHR)?;}$$
$$\text{sight-seeing; } \neg\text{at(LHR)?}\ ]\ )$$

That is, either I check that I am not at London Heathrow, then do sight-seeing and then either stop (nil) or do any other action, repeatedly verifying that I am still not at Heathrow. Or, again testing that I am never at Heathrow, I do any action and then go sight-seeing. 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 $\varphi?$ that enforces a formula, and/or a nondeterministic choice of the actions allowable in the current situation. In many cases, this is the set of all actions $\mathcal{A}$.

[5] We use the following abbreviations (Gabaldon 2004): $(\exists s_1 : s \sqsubseteq s_1 \sqsubseteq s')\Phi \equiv (\exists s_1)\{s \sqsubseteq s_1 \wedge s_1 \sqsubseteq s' \wedge \Phi\}$ and $(\forall s_1 : s \sqsubseteq s_1 \sqsubseteq s')\Phi \equiv (\forall s_1)\{[s \sqsubseteq s_1 \wedge s_1 \sqsubseteq s'] \supset \Phi\}$

Recall that in GOLOG $\varphi?$ states that the formula φ has to hold in the current situation and that **ndet**(L) is the nondeterministic choice among the elements of the list L. For example, the only possible next steps for **ndet**$([a,b])$ are taking action a or taking action b. Thus, assuming the current situation is s, the set of possible successor situations is 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 never being at Heathrow is extended over all situations using **always**. Observe that by this several BDFs are contributing situation constraints to the same situation. To combine these coinciding situation constraints we define the function χ as

$$\chi(\varphi_1?,\ \varphi_2?) = (\varphi_1 \wedge \varphi_2)?$$
$$\chi(\varphi?,\ \textbf{ndet}(l)) = \varphi?; \textbf{ndet}(l)$$
$$\chi(\textbf{ndet}(l_1),\ \textbf{ndet}(l_2)) = \textbf{ndet}(l_1 \cap l_2)$$
$$\chi(\varphi_1?; \textbf{ndet}(l_1),\ \varphi_2?; \textbf{ndet}(l_2)) = (\varphi_1 \wedge \varphi_2)?; \textbf{ndet}(l_1 \cap l_2)$$

plus its reflexive completion, where the φ's are formulae of the situation calculus and the l's are lists of actions. In our example, the temporal extent of **always**(¬at(LHR)) and **eventually**(**occ**(sight-seeing)) overlap. In these situations, the imposed situation constraints (¬at(LHR)? and sight-seeing) are combined, in this case by simple sequencing (second rule above).

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 BDF ψ, and further ψ' is the progressed BDF that needs to be satisfied in the future. Note that BDFs are treated as syntactic entities in the context of our compilation and accordingly are only syntactically manipulated. In the following we use STOP as a shorthand for $\nexists a.\textbf{occ}(a)$. $\mathcal{C}$ is defined through the following set of axioms:

$$\mathcal{C}(f, f?, \text{TRUE}), \forall f \in \mathcal{F} \cup \mathcal{R}$$
$$\mathcal{C}(\textbf{occ}(a), \textbf{ndet}([a]), \text{TRUE}), \forall a \in \mathcal{A}$$
$$\mathcal{C}(\textbf{final}(f), sc, \psi') \equiv (sc = f?; \textbf{ndet}([]) \wedge \psi' = \text{STOP}) \vee (sc = \textbf{ndet}(\mathcal{A}) \wedge \psi' = \textbf{final}(f))$$
$$\mathcal{C}(\psi_1 \wedge \psi_2, sc, \psi') \equiv \mathcal{C}(\psi_1, sc_1, \psi_1') \wedge \mathcal{C}(\psi_2, sc_2, \psi_2') \wedge sc = \chi(sc_1, sc_2) \wedge \psi' = (\psi_1' \wedge \psi_2')$$
$$\mathcal{C}(\psi_1 \vee \psi_2, sc, \psi') \equiv \mathcal{C}(\psi_1, sc, \psi') \vee \mathcal{C}(\psi_2, sc, \psi')$$
$$\mathcal{C}(\psi_1 \rightarrow \psi_2, sc, \psi') \equiv \mathcal{C}((\psi_1 \wedge \psi_2) \vee \neg\psi_1, sc, \psi')$$
$$\mathcal{C}(\textbf{next}(\psi), \textbf{ndet}(\mathcal{A}), \psi)$$
$$\mathcal{C}(\textbf{always}(\psi), sc, \psi') \equiv (\mathcal{C}(\psi, sc, \psi'') \wedge \psi'\text{=STOP} \wedge\ (\psi''\text{=STOP} \vee \psi''\text{=TRUE})) \vee (\mathcal{C}(\psi \wedge \textbf{next}(\textbf{always}(\psi)), sc, \psi')$$
$$\mathcal{C}(\textbf{eventually}(\psi), sc, \psi') \equiv \mathcal{C}(\psi \vee \textbf{next}(\textbf{eventually}(\psi)), sc, \psi')$$
$$\mathcal{C}(\textbf{until}(\psi_1, \psi_2), sc, \psi') \equiv \mathcal{C}(\psi_2 \vee (\psi_1 \wedge \textbf{next}(\textbf{until}(\psi_1, \psi_2))), sc, \psi')$$
$$\mathcal{C}(\text{TRUE}, sc, \text{TRUE}) \equiv sc = \textbf{ndet}([]) \vee sc = \textbf{ndet}(\mathcal{A})$$

The situation constraint **ndet**$([\,])$ states that no action may be taken. This enforces that the program, that we are generating, terminates, or, speaking in terms of situations, that situations grow no longer.

Further we allow negation of BDFs, $\neg\psi$, but this requires special treatment: GOLOG finds situations, i.e. action sequences, that satisfy a program, but for negation it is not obvious how the complement, that is the set of situations that do *not* satisfy the program, would be computed. We address this by pushing the negation down to the atomic level (fluents and action occurrence), thereby avoiding the above problem. For parsimony we only show some less obvious cases:

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

The actual compilation then proceeds by repeatedly applying these axioms to a given BDF, ψ, obtaining situation constraints that, when taken together, form a GOLOG program δ_ψ^h, that exactly describes the same set of situations as the BDF. Non-determinism in above definitions directly translates into non-determinism in GOLOG:

$$\textbf{Compile}(\psi, h, \delta_\psi^h) \equiv \exists l. \forall \delta' (\delta' \in l \Leftrightarrow \textbf{Prog}(\psi, h, \delta')) \wedge \delta_\psi^h = \textbf{ndet}(l)$$
$$\textbf{Prog}(\psi, 0, \delta) \equiv \exists x. \mathcal{C}(\psi, \delta, x) \wedge \delta = \varphi?$$
$$\textbf{Prog}(\text{STOP}, h, nil)$$
$$\textbf{Prog}(\text{TRUE}, h, \textbf{ndet}(\mathcal{A}))$$
$$\textbf{Prog}(\psi, h, \delta) \equiv h \neq 0 \wedge \psi \neq \text{STOP} \wedge \mathcal{C}(\psi, sc, \psi') \wedge \textbf{Compile}(\psi', h-1, \delta')\ \wedge\ \delta = sc; \delta'$$

These definitions can be directly implemented in Prolog, where $\forall$ is realized using *findall/3*. Then for a given BDF ψ and horizon h the query **Compile**(ψ, h, δ_ψ^h) produces the GOLOG program δ_ψ^h with the desired properties. Some optimization of the generated code is advisable, but for parsimony we omit these rather technical details here.

Recall that $\mathcal{D}$ denotes an action theory in the situation calculus and that $Do(\delta, s, s')$ states that GOLOG program δ starting in situation s can successfully terminate in situation s'. We prove the soundness and completeness of our compilation method with respect to the semantics stated above:

Theorem 1. (Soundness) Let ψ be a basic desire formula and δ_ψ^h be the corresponding program for horizon h (i.e. **Compile**(ψ, h, δ_ψ^h)). Then for any situation $s_n = do([a_1, a_2, \ldots, a_n], s)$ with $n \leq h$ such that $\mathcal{D} \models Do(\delta_\psi^h, s, s_n)$, it holds that $\mathcal{D} \models \psi[s, s_n]$.

Theorem 2. (Completeness) Let ψ be a basic desire formula and δ_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 $\mathcal{D} \models Do(\delta_\psi^h, s, s_n)$.

The proofs for both theorems can be found in the Appendix. Intuitively, soundness states that any program execution will result in a situation that also satisfies the BDF, while completeness establishes that all situations that satisfy the BDF are preserved.

4.2 Multi-Program Synchronization

Now that we have GOLOG programs that enforce satisfaction of each of the BDFs in our QPF, we want to combine these together with a pre-existing agent program to eventually provide a semantics for 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 DTGOLOG.

A transition semantics for GOLOG was first introduced in (De Giacomo, Lespérance, & Levesque 2000) 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 $\mathit{Trans}(\delta, s, \delta', s')$ and $\mathit{Final}(\delta, s)$. Given an action theory, a program δ and a situation s, *Trans* defines the set of possible successor configurations (δ', s') according to the action theory. *Final* 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, $\mathit{Trans}(a_1; a_2, a, a_2, do(a_1, s))$ describes the only possible transition that is possible if the action a_1 is possible in situation s according to the action theory. $do(a_1, s)$ is the transition and a_2 is the remaining program to be executed. 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 (De Giacomo, Lespérance, & Levesque 2000), this definition is equivalent to the original Do.

Using the transition semantics we can formally define the synchronization of two programs δ_1, δ_2 by a new GOLOG construct $\mathbf{sync}(\delta_1, \delta_2)$:

$$\begin{aligned}
&\mathit{Trans}(\ \mathbf{sync}(\delta_1, \delta_2),\ s,\ \mathbf{sync}(\delta_1', \delta_2'),\ s'\ ) \equiv \\
&\quad (\mathit{Trans}(\delta_1, s, \delta_1', s') \wedge \mathit{Trans}(\delta_2, s, \delta_2', s')) \\
&\quad \vee \big(s' = s \wedge (\ (\mathit{Trans}(\delta_1, s, \delta_1', s) \wedge \delta_2' = \delta_2) \\
&\qquad\qquad \vee (\delta_1' = \delta_1 \wedge \mathit{Trans}(\delta_2, s, \delta_2', s))\ )\big) \\
&\mathit{Final}(\ \mathbf{sync}(\delta_1, \delta_2),\ s\ ) \equiv \mathit{Final}(\delta_1, s) \wedge \mathit{Final}(\delta_2, s)
\end{aligned}$$

That is the program $\mathbf{sync}(\delta_1, \delta_2)$ can perform a transition in a situation s to a new situation s' iff both programs δ_1 and δ_2 can perform a transition to s' or when $s' = s$ and one of δ_1 and δ_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 (δ_1', δ_2'). To synchronize more than two programs we can use nesting, so for instance $\mathbf{sync}(\delta_1, \mathbf{sync}(\delta_2, \delta_3))$ would synchronize three programs.

The following theorem follows immediately from the above definitions.

Theorem 3. Let δ_a, δ_b be two GOLOG programs. Then for any situation S, $\mathcal{D} \models Do(\delta_a, S_0, S)$ and $\mathcal{D} \models Do(\delta_b, S_0, S)$ if and only if $\mathcal{D} \models Do(\mathbf{sync}(\delta_a, \delta_b), S_0, S)$.

The theorem states that a situation S describes a legal execution for both programs, if and only if it describes a legal execution of the synchronization. Thus, synchronizing two programs *intersects* the sets of situations they describe.

A Decision-Theoretic Transition Semantics As stated above, DTGOLOG is defined using an evaluation semantics and that does not suit our requirements. Thus, we have redefined DTGOLOG in an equivalent transition semantics, or, seen differently, extended the existing GOLOG transition semantics to decision-theoretic planning.

A thorough presentation of DTGOLOG and the complete set of definitions of our transition semantics is out of the scope of this paper. This, rather technical, section is thus intended for the benefit of readers familiar with the original DTGOLOG work as described in (Boutilier *et al.* 2000) and only provides an overview of the necessary changes.

The transition semantics is defined via the predicate $\mathit{BestTrans}(\delta, s, d, (\pi, v, prob), \mathcal{N}, \mathcal{D})$, where δ is a program, s a situation, d an integer indicating the current depth in the search tree, π, v, and $prob$ the optimal policy, the expected utility, and the termination probability respectively, and $\mathcal{N}$, $\mathcal{D}$ are lists of *nodes* and *decisions*. The latter two lists constitute the fundamental difference between the evaluation semantics and the transition semantics. In an evaluation semantics the interpreter traverses the tree of possible action- and action-outcome sequences, called the *decision-tree*, in a depth-first manner: nodes are expanded right when they are created and local decisions can be made directly following the expansion of all sub-nodes. In contrast, transition semantics does not force any particular expansion strategy. Instead, nodes can be expanded in an arbitrary order. Thus, we need to do some bookkeeping of unexpanded nodes (often called *frontier*) and the open decisions, which are to be made once all sub-nodes have been expanded. The two list $(\mathcal{N}, \mathcal{D})$ mentioned above serve this purpose.

For example a transition over a non-deterministic choice among sub-programs is then defined as follows:

$$\begin{aligned}
&\mathit{BestTrans}(\ \mathbf{ndet}([\delta]); \delta', s, d, (\pi, v, prob), \mathcal{N}, \mathcal{D}) \equiv \\
&\quad \mathcal{N} = [\ (\delta; \delta', s, d, (\pi, v, prob))\ ]\ \wedge\ \mathcal{D} = [] \\
&\mathit{BestTrans}(\ \mathbf{ndet}([\delta_1 \mid \delta_2]); \delta', s, d, (\pi, v, prob), \mathcal{N}, \mathcal{D}) \equiv \\
&\quad \exists(\pi_1, v_1, prob_1, \pi_2, v_2, prob_2). \\
&\quad \mathit{BestTrans}(\ \mathbf{ndet}(\delta_2); \delta', s, d, (\pi_2, v_2, prob_2), \mathcal{N}_2, \mathcal{D}_2) \wedge \\
&\quad \mathcal{N} = [\ (\delta_1; \delta', s, d, (\pi_1, v_1, prob_1)) \mid \mathcal{N}_2]\ \wedge \\
&\quad \mathcal{D} = [max((\pi_1, v_1, prob_1), (\pi_2, v_2, prob_2), (\pi, v, prob)) | \mathcal{D}_2]
\end{aligned}$$

where $\mathcal{N}$ is the list of possible successor configurations (nodes) and $\mathcal{D}$ the list of decisions to be made once the values of all sub-nodes have been determined.

Roughly, the expression $max((\pi_1, v_1, prob_1), (\pi_2, v_2, prob_2), (\pi, v, prob))$ states that $(\pi, v, prob)$ is equal to the 'maximum' of $(\pi_1, v_1, prob_1)$ and $(\pi_2, v_2, prob_2)$, where the ordering is defined by a mixture of termination probability ($prob$) and value (v) as in (Boutilier *et al.* 2000). However, at the time of expansion of the node these values

are not yet known and thus the decision has to be delayed until all sub-nodes have been expanded, i.e. in all sub-trees the horizon has been reached and the actual values have been propagated back.

The definitions for the other program constructs (loops, conditionals etc.) are rather straightforward so we omit them here for reasons of space.

4.3 Expressing Preference in DTGOLOG

So far we have shown how to compile BDFs into *hard* constraints, realized as GOLOG programs. By this we achieve what TLPlan (Bacchus & Kabanza 2000) does for classical planning and Gabaldon (Gabaldon 2004) does in the situation calculus, namely the integration of domain control knowledge to prune the search space. To make these constraints *soft* and to rank them to express ordered preferences we need to introduce yet another construct into GOLOG: $\mathbf{lex}(\delta_1, \delta_2)$ states that program δ_1 is (lexicographically) preferred to δ_2. This is implemented, roughly, by first trying δ_1 and only if it fails[6] the second program, δ_2, is considered. Formally this intuition is captured by extending the just introduced predicate *BestTrans* so that it defines clusters of nodes (and corresponding decisions) of equal degree of preference. Recall that a node is a tuple $(\delta, s, d, (\pi, v, prob))$ where δ is the remaining GOLOG program, s the current situation, d the current search depth, π a policy (variable), v the value (variable), and $prob$ the termination (variable). The former three (δ, s, d) are always instantiated (input), whereas $(\pi, v, prob)$ get instantiated only when the corresponding branch of the search tree ends (output). Decisions consist of asserting which tuple $(\pi, v, prob)$ to choose at choice points (cf. previous section). All previously seen GOLOG constructs return exactly one cluster of (possibly multiple) nodes and the above construct returns two clusters:

$$\begin{aligned}&\mathit{BestTrans}(\ [\mathbf{lex}(\delta_1, \delta_2) \mid \delta'], s, d, \pi, v, prob, [C_1, C_2], [[\,], [\,]]) \equiv\\ &\quad C_1 = [\ ([\delta_1 \mid \delta'], s, d, (\pi, v, prob))\ ] \wedge\\ &\quad C_2 = [\ ([\delta_2 \mid \delta'], s, d, (\pi, v, prob))\ ]\end{aligned}$$

So what used to be a (flat) list of nodes $\mathcal{N}$ is now a list of lists of nodes, sorted by qualitative preference. The preference for earlier clusters is formally defined in the evaluation strategy of clusters and nodes. These definitions form the main loop for planning by decision-tree search under the newly introduced transition semantics:

Recall the following notation: N denotes a node, $\mathcal{N}$ a list of nodes, and $\vec{\mathcal{N}}$ denotes a cluster (list) of lists of nodes.

$$\begin{aligned}&\mathit{BestTrans}^*(N, h) \equiv \qquad \textit{nodes}\\ &\quad N = (\delta, s, d, (\pi, v, prob)) \wedge\\ &\quad \big(\exists \mathcal{N}, \mathcal{D}.d \leq h \wedge \mathit{BestTrans}(\delta, s, d, (\pi, v, prob), \vec{\mathcal{N}}, \vec{\mathcal{D}}) \wedge\\ &\quad\ \mathit{BestTrans}^*_{\text{quali}}(\vec{\mathcal{N}}, \vec{\mathcal{D}}, prob, h)\big)\\ &\quad \vee (d > h \wedge \pi = nil \wedge Reward(v)[s] \wedge prob = 1)\end{aligned}$$

$$\begin{aligned}&\mathit{BestTrans}^*_{\text{quali}}([\mathcal{N}], [\mathcal{D}], prob, h) \equiv \qquad \textit{clusters}\\ &\quad \mathit{BestTrans}^*_{\text{quanti}}(\mathcal{N}, h) \wedge \mathit{MakeDecisions}(\mathcal{D})\\ &\mathit{BestTrans}^*_{\text{quali}}([\mathcal{N} \mid \vec{\mathcal{N}}], [\mathcal{D} \mid \vec{\mathcal{D}}], prob, h) \equiv\\ &\quad (\mathit{BestTrans}^*_{\text{quanti}}(\mathcal{N}, h) \wedge \mathit{MakeDecisions}(\mathcal{D}) \wedge prob > 0)\ \vee\\ &\quad \mathit{BestTrans}^*_{\text{quali}}(\vec{\mathcal{N}}, \vec{\mathcal{D}}, prob, h)\\ &\mathit{BestTrans}^*_{\text{quanti}}([], h) \equiv \text{TRUE} \qquad \textit{node-lists}\\ &\mathit{BestTrans}^*_{\text{quanti}}([N_1 | \mathcal{N}], h) \equiv \mathit{BestTrans}^*(N_1, h) \wedge\\ &\qquad\qquad \mathit{BestTrans}^*_{\text{quanti}}(\mathcal{N}, h)\end{aligned}$$

In $\mathit{BestTrans}^*_{\text{quali}}$ 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, are less preferred partitions considered. The predicate *MakeDecisions*($\mathcal{D}$) makes decisions as described above, once all variables in $\mathcal{D}$ have been instantiated, i.e. when all associated sub-nodes have been processed.

To compile a qualitative preference formula $\Psi = \psi \,\vec{\&}\, \Psi'$ we first compile the sub-formulae ψ, Ψ' into programs δ^h_ψ and $\delta^h_{\Psi'}$ and then combine them into:

$$\delta^h_\Psi = \mathbf{lex}(\mathbf{sync}(\delta^h_\psi, \delta^h_{\Psi'}), \mathbf{lex}(\delta^h_\psi, \delta^h_{\Psi'}))$$

That is, it is most preferred to perform the synchronized execution of both programs, which corresponds to the satisfaction of both sub-formulae. If that is not possible, the former program is preferred over the latter. This reflects the intended semantics of our QPFs.

Example 3. Consider the following GOLOG program for determining transportation: $\delta = \mathbf{ndet}([drive, fly])$ and the preference $\Psi = \neg\mathbf{final}(poor)\ \vec{\&}\ \mathbf{always}(\neg(\mathbf{occ}(drive)))$. Compilation with horizon one of this QPFs provides us with the following GOLOG program δ^1_Ψ assuming that the only action available in our domain are fly, drive, and walk:

$$\begin{aligned}&\mathbf{lex}(\ \mathbf{sync}(\ \mathbf{ndet}([\neg poor?, \mathbf{ndet}([fly, drive, walk]); \neg poor?]),\\ &\qquad \mathbf{ndet}([nil, \mathbf{ndet}([fly, walk])])\ ),\\ &\quad \mathbf{lex}(\ \mathbf{ndet}([\neg poor?, \mathbf{ndet}([fly, drive, walk]); \neg poor?]),\\ &\qquad \mathbf{ndet}([nil, \mathbf{ndet}([fly, walk])])))\end{aligned}$$

This will be combined with the original GOLOG program δ, the rough planning problem to be addressed, by $\mathbf{lex}(\mathbf{sync}(\delta, \delta^1_\Psi), \delta)$. 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 contains the program:

$$\begin{aligned}&\mathbf{sync}(\ \mathbf{ndet}([drive, fly]),\\ &\quad \mathbf{sync}(\ \mathbf{ndet}([[\neg poor?], [\mathbf{ndet}([fly, drive, walk]), \neg poor?]]),\\ &\qquad \mathbf{ndet}([nil, \mathbf{ndet}([fly, walk])])))\end{aligned}$$

Let's assume that driving and walking are the only affordable means of transportation. Then obviously we cannot satisfy both desires and in fact after the only possible synchronized transition, doing fly, the test $\neg poor?$ fails. We thus move on to the next best cluster:

$$\begin{aligned}&\mathbf{sync}(\mathbf{ndet}([fly, drive]),\\ &\quad \mathbf{ndet}([[\neg poor?], [\mathbf{ndet}([fly, drive, walk]), \neg poor?]])).\end{aligned}$$

There are two possible synchronized transitions here, either doing fly or $drive$, however, only the second will make the second test of $\neg poor?$ succeed and will thus be chosen.

[6] Note that we have not defined 'failure' of a program. For this application we understand a program in READYLOG to fail if its termination probability is zero. But more conservative standpoints, for example requiring the probability to be above a certain threshold, could be easily realized.

The following corollary follows from Theorems 2 and 3, and the correctness of our decision-theoretic transition semantics.

Corollary 1. Let δ be an arbitrary DTGOLOG program, Ψ a Qualitative Preference Formula, and $h \in \mathbb{N}$ a horizon. Further, let δ^h_Ψ be the compilation of Ψ for horizon h. Then a constructive proof of

$$\mathcal{D} \models \exists \pi, v, prob.$$
$$\mathit{BestTrans}^* \Big(\big(\mathbf{lex}(\mathbf{sync}(\delta, \delta^h_\Psi), \delta), S_0, 0, (\pi, v, prob) \big), h \Big)$$

as a side-effect produces a policy[7] π which has the following properties:

i) any successful execution of π leads to a situation that is most preferred among all possible situations, which is the set of situations of length $\leq h$ which describe a legal execution trace for δ according to the action theory $\mathcal{D}$;

ii) π maximizes the expected reward within this partition of the situation space 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 (Ferrein, Fritz, & Lakemeyer 2004). 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 page 3. The actions `searchFlight(From, To, OutDate, ReturnDate)` and `searchHotel(Destination, CheckinDate, CheckoutDate)` realize the querying and wrapping of the relevant Web pages[8].

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 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.

[7] a DTGOLOG program without any non-deterministic choices

[8] Technically speaking these are so-called *sensing actions*, but space precludes a thorough discussion of this issue. The interested reader is referred to the literature, e.g. (Reiter 2001).

6 Related Work

Arguably, the most widely accepted formalism for expressing and reasoning about *qualitative user preferences* within the field of artificial intelligence is CP-nets (Boutilier *et al.* 2004; Domshlak *et al.* 2003). The purpose of this work is to reason about user preferences when only *ceteris paribus* statements of preference over the values of domain features are available, i.e. statements of the form "a red car is better than a blue car" assuming that all other domain features stay the same for either choice. Generally, this does not produce a full ordering of configurations which makes it impractical for our purposes. Further it is not clear how this *static* approach could be incorporated efficiently into a planning, i.e. a *dynamic* task.

In (Domshlak *et al.* 2003) 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 agent programming.

Recently, Brafman et al. (Brafman & Chernyavsky 2005) presented an approach to planning with goal preferences and constraints where they use TCP-nets to rank possible goal states. A TCP-net is a tradeoff-enhanced CP-net, which allows the user to express priority of preferences over some variables relative to those over others. They approach the planning problem by compiling it into an equivalent CSP problem and imposing variable instantiation constraints on the CSP solver, according to the TCP-net. Our work differs from theirs largely by the fact that we explicitly deal with temporal preferences, as opposed to just static preferences over solely the goal state. Going back to our introductory travel example, Brafman's approach would, for instance, not be able to handle the preference of never being at London Heathrow.

Our work is related to that of Gabaldon (Gabaldon 2004) who, following previous work by Bacchus and Kabanza (Bacchus & Kabanza 2000) and Doherty and Kvarnström (Doherty & Kvarnström 2001), 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 and thus cannot serve as defeasible preferences. Also, it is not obvious how several preferences could be combined in Gabaldon's approach, a task that we easily solve at the level of programs in Section 4.

Also related is the work of Sardina and Shapiro (Sardina & Shapiro 2003) who integrate qualitatively prioritized goals into the IndiGolog programming language. Our approach differs from theirs in several 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 (Bienvenu, Fritz, & McIlraith 2006) the authors deal with the problem of classical planning with qualitative temporal preferences only, but with a richer language, which subsumes ours. The planning is realized as heuristic search. To that end the authors propose an admissible evaluation function and a best-first search algorithm that optimally satisfies the user's preferences. Through the specification of a semantic preserving progression of preference formulae, the evaluation necessary at each step of the search is drastically reduced.

7 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. This required the definition of multi-program synchronization and the redefinition of DTGOLOG in a transition semantics. We proved the soundness and completeness of our compilation. The resulting system is able to perform planning under hard constraints, Markovian quantitative, and non-Markovian qualitative preferences and in so doing is, to the best of our knowledge, the first system to integrate qualitative and quantitative preferences into agent programming. 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

In the following proofs we use the evaluation semantics of GOLOG as defined in (Levesque *et al.* 1997). Further we use $\star^h$ as a shorthand for the non-deterministic repetition of $\mathbf{ndet}(\mathcal{A})$ with a maximum of h repetitions. If h is omitted, the repetition is of arbitrary length. 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.

Lemma 1. 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.1 Soundness

Proof of Theorem 1: 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 $\delta^h_\psi = ?(f); \star^h$ and by assumption know that $\mathcal{D} \models Do(\delta^h_\psi, s, s_n)$ it follows from the definition of Do that $\mathcal{D} \models f[s]$ and thus $\mathcal{D} \models f[s, s_n]$;
  - $\mathbf{occ}(a)$: With $\mathcal{C}(\mathbf{occ}(a), \mathbf{ndet}([a]), \text{TRUE})$ we have $\delta^h_\psi = \mathbf{ndet}([a]); \star^{h-1}$ 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 to above we have $\mathcal{C}(f, (\neg f)?, \text{TRUE})$ and by hypothesis know that $\mathcal{D} \models Do(\delta^h_\psi, s, s_n)$ thus it follows from the definition of Do that $\mathcal{D} \models \neg f[s]$ and thus $\mathcal{D} \models \neg f[s, s_n]$;
  - $\neg\mathbf{occ}(a)$: $\mathcal{C}(\neg\mathbf{occ}(a), sc, \psi') \equiv (sc = \mathbf{ndet}([]) \wedge \psi' = \text{STOP}) \vee (sc = \mathbf{ndet}(\mathcal{A} \setminus \{a\}) \wedge \psi' = \text{TRUE}), \forall a \in \mathcal{A}$, 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 the base case is defined for the final situation s_n: Since no further actions occur from s_n, we need only look at the case of horizon zero. The definition of $\mathbf{Prog}(\psi, h, \delta)$ for $h = 0$ has three possible cases:

  $$\mathbf{Prog}(\psi, 0, \delta) \equiv \exists x.\mathcal{C}(\psi, \delta, x) \wedge \delta = \varphi?$$
  $$\mathbf{Prog}(\text{STOP}, h, nil)$$
  $$\mathbf{Prog}(\text{TRUE}, h, \mathbf{ndet}(\mathcal{A}))$$

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

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, as well as over all intervals of situations $[s_m, s_n], j \leq m \leq n$, 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

- $\psi = \mathbf{final}(f)$: then $sc_\psi = f?; \mathbf{ndet}([\,]) \wedge \psi' = \text{STOP}$ or $sc_\psi = \mathbf{ndet}(\mathcal{A}) \wedge \psi' = \mathbf{final}(f)$. As $Do(\mathbf{ndet}([]), s, s_n)$ 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.
- $\psi = \mathbf{next}(\varphi)$: $sc_\psi = \mathbf{ndet}(\mathcal{A})$ and $\psi' = \mathbf{next}(\varphi)$. The proposition follows by induction hypothesis.

- $\psi =$ **always**(φ): then either $\mathcal{C}(\varphi, sc, \psi'') \wedge \psi' =$STOP$\wedge$ ($\psi'' =$ STOP $\vee$ $\psi'' =$TRUE) or $\mathcal{C}(\varphi \wedge$ **next**(**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 $\psi' =$ 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 $\mathcal{D} \models$ **always**$(\varphi)[s_j, s_n]$. It follows $\mathcal{D} \models (\forall s_1 : s_{j-1} \sqsubseteq s_1 \sqsubseteq s_n)\varphi[s_1, s_n]$.
- $\psi =$ **eventually**(φ): from $\mathcal{C}(\varphi \vee$ **next**(**eventually**$(\varphi)), sc, \psi')$ we have together with induction hypothesis that either $\mathcal{D} \models \varphi[s_{j-1}, s_n]$ or $\mathcal{D} \models$ **eventually**$(\varphi)[s_j, s_n]$. In either case we have $\mathcal{D} \models (\exists s_1 : s_{j-1} \sqsubseteq s_1 \sqsubseteq s_n)\varphi[s_1, s_n]$
- $\psi =$ **until**(ψ_1, ψ_2): thus either $\mathcal{C}(\psi_2, sc, \psi')$ or $\mathcal{C}(\psi_1 \wedge$ **next**(**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 $\mathcal{D} \models$ **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.2 Completeness

Proof of Theorem 2: The proof is again 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 **Prog** and $\mathcal{C}$ we have $\delta^h_\psi = f?;\star$. Further $Do(f?;\star, s, s_n) \stackrel{\text{def}}{=} \exists s^*.Do(f?, s, s^*) \wedge Do(\star, s^*, s_n)$ is satisfied with $s^* = s$.
- **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 $\delta^h_\psi =$ **ndet**$([a]);\star$ which is equivalent to $\delta^h_\psi = a;\star$. Again $Do(a;\star, s, s_n) \stackrel{\text{def}}{=} \exists s^*.Do(a, s, s^*) \wedge Do(\star, s^*, s_n)$ is satisfied with $s^* = s_1 = do(a, s)$.
- $\neg f \in \mathcal{F} \cup \mathcal{R}$: similar to above we have by assumption that $\mathcal{D} \models \neg f[s, s_n]$ and by definition $\mathcal{D} \models \neg f[s]$. Also by definition of **Prog** and $\mathcal{C}$ we have $\delta^h_\psi = \neg f?;\star$. Further $Do(\neg f?;\star, s, s_n) \stackrel{\text{def}}{=} \exists s^*.Do(\neg f?, s, s^*) \wedge Do(\star, s^*, s_n)$ is satisfied with $s^* = s$.
- $\neg$**occ**(a): By definition we have $do(a, s) \not\sqsubseteq s_n \vee \neg Poss(a[s], s)$ and either $\delta^h_\psi =$ **ndet**$(\mathcal{A} \setminus \{a\});\star$ or $\delta^h_\psi =$ **ndet**$([\,]); nil$. For the former case $Do($**ndet**$(\mathcal{A}\setminus\{a\});\star, s, s_n) \stackrel{\text{def}}{=} \exists s^*.Do($**ndet**$(\mathcal{A}\setminus\{a\}), s, s^*) \wedge Do(\star, s^*, s_n)$ is satisfied for any $s^* = s_1 = do(b, s)$ with $b \neq a$. 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:

$$\delta^h_\psi = \textbf{ndet}([\quad [f?; \textbf{ndet}([\,])], \quad [\textbf{ndet}(\mathcal{A}); \textbf{ndet}([\quad [f?; \textbf{ndet}([\,])], \quad [\textbf{ndet}(\mathcal{A}); \ldots] \,]) \,] \,])$$

  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 $\delta_n = [\underbrace{\textbf{ndet}(\mathcal{A}); \ldots; \textbf{ndet}(\mathcal{A})}_{n}; f?; \textbf{ndet}([\,])]$ in δ^h_ψ. In combination this implies $\mathcal{D} \models Do(\delta_n, s, s_n)$ and further, by definition of $Do($**ndet**$(\ldots), s, s')$, $\mathcal{D} \models Do(\delta^h_\psi, s, s_n)$.
- $\psi = \psi_1 \wedge \psi_2$: 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 $\delta^h_{\psi_i}$ we have $\mathcal{D} \models Do(\delta^h_{\psi_i}, s, s_n)$. 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 $\delta^h_{\psi_1}, \delta^h_{\psi_2}$. This set can be described 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 conjunction of the two BDFs ψ_1, ψ_2.

  Using Lemma 1 and induction hypothesis it is sufficient to show that any situation s_n which is a successful execution of both programs $\delta^h_{\psi_1}, \delta^h_{\psi_2}$ is also a successful execution of the combined program δ^h_ψ. The axiom for compiling conjunction combines the situation constraints sc_1, sc_2 raised by compiling the two sub-formulae 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 constrains, 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 implies $\mathcal{D} \models \exists s'.Do(\psi_1?, s, s') \wedge Do(\psi_2?, s, s') \wedge s' \sqsubseteq s_n$ and thus again by definition of Do : $\mathcal{D} \models \exists s'.Do((\psi_1 \wedge \psi_2)?, s, s') \wedge s' \sqsubseteq s_n$.
  - $\chi(\psi?,$ **ndet**$(L)) =$ **ndet**$(\psi?; L)$: By assumption we know $\mathcal{D} \models \exists s'.Do(\psi?, s, s') \wedge s' \sqsubseteq s_n$ and $\mathcal{D} \models \exists s''.Do($**ndet**$(L), s, s'') \wedge s'' \sqsubseteq s_n$. From the definition of Do we know that $s' = s$. Thus $\mathcal{D} \models Do(\psi?, s, s) \wedge \exists s''.Do($**ndet**$(L), s, s'') \wedge s'' \sqsubseteq s_n$ which is equivalent to $\exists s''.Do([\psi?;$ **ndet**$(L)], s, s'') \wedge s'' \sqsubseteq s_n$, the proposition.
  - $\chi($**ndet**$(L_1),$ **ndet**$(L_2)) =$ **ndet**$(L_1 \cap L_2)$: By assumption $\mathcal{D} \models \exists s'.Do($**ndet**$(L_1), s, s') \wedge s' \sqsubseteq s_n$ and $\mathcal{D} \models \exists s''.Do($**ndet**$(L_2), s, s'') \wedge s'' \sqsubseteq s_n$. Sure enough 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 $\mathcal{D} \models \exists s'.Do($**ndet**$(L_1 \cap L_2), s, s') \wedge s' \sqsubseteq s_n$.

– $\chi((\psi_1?; \mathbf{ndet}(L_1)), (\psi_2?; \mathbf{ndet}(L_2))) = ((\psi_1 \wedge \psi_2)?; \mathbf{ndet}(L_1 \cap L_2))$: See last item.

As we know s_n is a path in both trees, for $\delta^h_{\psi_1}$ and $\delta^h_{\psi_2}$, 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: $\mathcal{D} \models Do(\delta^h_\psi, s, s_n)$.

- $\psi = \psi_1 \vee \psi_2$: By induction hypothesis we know that either $\mathcal{D} \models Do(\delta^h_{\psi_1}, s, s_n)$ or $\mathcal{D} \models Do(\delta^h_{\psi_2}, s, s_n)$. Thus $\mathcal{D} \models Do(\delta^h_{\psi_1}, s, s_n) \vee Do(\delta^h_{\psi_2}, s, s_n) \equiv Do(\mathbf{ndet}([\delta^h_{\psi_1}, \delta^h_{\psi_2}]), s, s_n)$. With $\mathbf{ndet}([\delta^h_{\psi_1}, \delta^h_{\psi_2}])$ 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 $\mathcal{D} \models Do(\delta^{h-1}_\varphi, do(a, s), s_n)$. The compilation of ψ is defined as $\delta^h_\psi = \mathbf{ndet}(\mathcal{A}); \delta^{h-1}_\varphi$. Clearly $\mathcal{D} \models Do(\mathbf{ndet}(\mathcal{A}), s, do(a, s))$ for some action $a \in \mathcal{A}$. We thus get the proposition: $\mathcal{D} \models \exists a \in \mathcal{A}.Do(\mathbf{ndet}(\mathcal{A}), s, do(a, s)) \wedge Do(\delta^{h-1}_\varphi, do(a, s), s_n) \equiv Do(\mathbf{ndet}(\mathcal{A}); \delta^{h-1}_\varphi, s, s_n) \equiv Do(\delta^h_\psi, s, s_n)$.
- $\psi = \mathbf{always}(\varphi)$: 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 $\delta^h_\psi = \mathbf{ndet}([\delta^h_1, \delta^h_2])$ with δ^h_1 the compilation of $\varphi[s, s_n] \wedge \mathbf{next}(\mathbf{always}(\varphi))[s, s_n]$ and δ^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 $\mathcal{D} \models \mathbf{eventually}(\varphi)[s, s_n] \stackrel{\text{def}}{=} (\exists s_1 : s \sqsubseteq s_1 \sqsubseteq s_n)\varphi[s_1, s_n] \equiv \varphi[s, s_n] \vee \mathbf{next}(\mathbf{eventually}(\varphi))[s, s_n]$. 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 $\mathcal{D} \models \mathbf{until}(\psi_1, \psi_2)[s, s_n]$ which by definition is $(\exists s_2 : s \sqsubseteq s_2 \sqsubseteq s_n)\{\psi_2[s_2, s_n] \wedge (\forall s_1 : s \sqsubseteq s_1 \sqsubset s_2)\psi_1[s_1, s]\} \equiv \psi_2[s, s_n] \vee (\psi_1[s, s_n] \wedge \mathbf{next}(\mathbf{until}(\psi_1, \psi_2))[s, s_n])$. 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. □

References

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

Bienvenu, M.; Fritz, C.; and McIlraith, S. 2006. Planning with qualitative temporal preferences. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning, Lake District, UK, June*.

Boutilier, C.; Reiter, R.; Soutchanski, M.; and Thrun, S. 2000. Decision-theoretic, high-level agent programming in the situation calculus. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000), Austin, TX*, 355–362.

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

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

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

Delgrande, J.; Schaub, T.; and Tompits, H. 2004. Domain-specific preferences for causal reasoning and planning. In *Proceedings of the 9th International Conference on Principles of Knowledge Representation and Reasoning (KR'04) Whistler, BC, Canada*, 673–682.

Doherty, P., and Kvarnström, J. 2001. TALplanner: A temporal logic based planner. *AI Magazine*. Fall Issue.

Domshlak, C.; Rossi, F.; Venable, B.; and Walsh, T. 2003. Reasoning about soft constraints and conditional preferences: complexity results and approximation techniques. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August*.

Ferrein, A.; Fritz, C.; and Lakemeyer, G. 2004. On-line decision-theoretic Golog for unpredictable domains. In *Proceedings of the 27th German Conference on Artificial Intelligence*.

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

Levesque, H. J.; Reiter, R.; Lesperance, Y.; Lin, F.; and Scherl, R. B. 1997. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming* 31(1-3):59–83.

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

Sardina, S., and Shapiro, S. 2003. Rational action in agent programs with prioritized goals. In *Proceedings of Autonomous Agents and Multi-Agent Systems Conference (AAMAS), Melbourne, Australia. July*, 417–424.

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