Declarative Preferences in Reactive BDI Agents

Mostafa Mohajeri Parizi[1], Giovanni Sileno[1], and Tom van Engers[1]

Complex Cyber Infrastructure, Informatics Institute, University of Amsterdam, the Netherlands
{m.mohajeriparizi,g.sileno,vanengers}@uva.nl

Abstract. Current agent architectures implementing the *belief-desire-intention* (BDI) model consider agents which respond reactively to internal and external events by selecting the first-available plan. Priority between plans is hard-coded in the program, and so the reasons why a certain plan is preferred remain in the programmer's mind. Recent works that attempt to include explicit preferences in BDI agents treat preferences essentially as a rationale for planning tasks to be performed at run-time, thus disrupting the reactive nature of agents. In this paper we propose a method to include declarative preferences (i.e. concerning states of affairs) in the agent program, and to use them in a manner that preserves reactivity. To achieve this, the plan prioritization step is performed offline, by (a) generating all possible outcomes of situated plan executions, (b) selecting a relevant subset of situation/outcomes couplings as representative summary for each plan, (c) sorting the plans by evaluating summaries through the agent's preferences. The task of generating outcomes in several conditions is performed by translating the agent's procedural knowledge to an ASP program using discrete-event calculus.

Keywords: BDI agents · CP-nets · Preferences · Belief-desire-intention · Answer set programming

1 Introduction

In the last decades several attempts have been made to move from machine-oriented views of programming towards concepts and abstractions that more closely reflect the way in which humans conceive the world. In particular, the *belief-desire-intention* framework (BDI) [30], building upon a theory of mind [5], has been introduced to provide a basis for the implementation of computational agents that exhibit rational behaviour, using the same representations that we typically use to address human behaviour. In the decision-making literature, instead, particular attention is given to the role of preferences: any model of agency involving decision-making is deemed to abide the agent's preferences [28]. This does not imply that any model of agency will rely on explicit preferences, rather it affirms the general principle that when there are multiple goals that should be achieved (or multiple ways to achieve a certain goal or even multiple sets of states that can be reached) the best course of action is the one that abides the most to the agent's preferences [28]. In practice, preferences can vary from the implicit "maximize utility" of optimizing agents [27] to explicit preferences specified in a preference representation language [7,34]. Unexpectedly, none of the main BDI languages presented in the literature support explicit preferences.

The present work proposes an approach for adding explicit declarative preferences (i.e. preferences about states of affairs possibly holding in the world) into BDI agent scripts. The novel contribution consists in presenting an offline method aiming to preserve the reactive executable nature of BDI agents. Declarative preferences are *compiled* together with the procedural knowledge and knowledge about primitive actions specified in the program into prioritized procedural knowledge. The resulting script is usable by any (AgentSpeak(L)-like) BDI interpreter such as Jason [3] or AgentScriptCC [25] without any modification to the reasoning cycle. The compilation approach has been selected to provide programmers with a higher abstraction model without compromising performance in execution. Indeed, our target use case is to embed purpose and constraints to programs—using intentional agents as controllers of given programs—for applications running on data-sharing infrastructures.

The paper is structured as follows. Section 2 provides an overview on related literature. Section 3 contains the proposed approach as the core of our contribution. Section 4 presents an illustrative example of application. Finally section 5 contains the discussion and a note on future developments.

2 Background

BDI Agents Agents specified following a BDI framework are represented by three mental attitudes. Beliefs are facts that the agent believes to be true. Desires capture the motivational dimension of the agent, typically conflated with the more concrete form of *goals*, representing procedures/states that the agent wants to perform/achieve. Intentions are selected conducts (or *plans*) that the agent commits to (in order to advance its desires).

Since their origin [30], the essential feature associated to BDI architectures is the ability to instantiate abstract plans that can (a) react to specific situations, and (b) be invoked based on their purpose. Consequently, the BDI execution model naturally relies on a *reactive* model of computation, usually in the form of some type of *event-condition-action* (ECA) rules often referred to as *goal-plan rules*. Goal-plan rules are uninstantiated specifications of the *means* (in terms of course of actions, or plan) for achieving a certain *goal* [30]. These constructs represent essentially the procedural knowledge (*how-to*) of the agent.

In current BDI implementations, preferences between these optional conducts are specified through a static ordering assigned by the programmer, typically via the ordering of rules in the code: the higher a rule is in the script, the more priority the associated plan has. This explains why most current frameworks including Jason [3], 2APL [8], AgentScriptCC [25], etc. are genuinely *reactive*: the scripts are interpreted without the need for any additional introspection/deliberation steps. However, these frameworks also expose functions that can be modified to implement alternative mechanism for goal-plan rule selection during the deliberation cycle. The latter option has been taken by almost all works adding explicit preferences to BDI agents [34,7,27]: the selection of the most preferred alternative is taken as a *reflective* process, where preferences provide a *rationale* to be applied online during the agent's deliberation cycle. The idea of relying on an offline step is instead proposed also in [24], but they only focused on

procedural preferences ("I prefer to be doing a_i rather than doing a_j"), which have a different level of abstraction w.r.t. to declarative preferences ("I prefer being in state s_i rather than being in state s_j").

Preference Languages Several models of preferences have been presented in the decision-making and planning literature, with various levels of granularity and expressiveness (see e.g. [11]). The most straightforward *quantitative* approaches are based upon *utility theory* and related forms of decision theory. In [6] one can find some examples of integration of these types of preferences in a BDI architecture.

Although quantitative approaches bring clear computational advantages, they also suffer from the non-trivial issue of translating users' preferences into utility functions. This explains the existence of a family of *qualitative* or hybrid solutions, as LPP [1] and PDDL3 [16]. Proposals exist for integrating LPP in BDI agents [34]. Other preference models, as CP-nets (qualitative) [4] and GAI networks (quantitative) [17], have been specifically introduced for taking into account dependencies and conditions between preferences via *compact representations* [28], highly relevant in domains with a large number of features. In the present work we will focus on CP-Nets because they rely on weaker assumptions, and exhibit primarily a qualitative nature. To our knowledge, [24] was the first attempt to introduce this type of representational models in a BDI architecture, although focusing only on procedural preferences.

More in detail, conditional *ceteris paribus* preferences networks (CP-nets) are a compact representation of preferences in domains with finite *attributes of interest* [4]. An attribute of interest is an attribute in the world (e.g. *weather*) that the agent has some sort of preference over its possible values (e.g. *sunny* and *rainy*). CP-nets build upon the idea that most of the preferences people make explicit are expressed jointly with an implicit *ceteris paribus* ("all things being equal") assumption. For instance, when players say "I prefer victory over loss", they do not mean at all costs and situations, but that they prefer victory, all other things being equal. An example of conditional preference could be "If I'm losing a game, I prefer to enjoy myself".

From Agent Scripts to Logic Programs Reasoning about the effects of actions/plans in different contexts is a step necessary to decide their conditional, relative preferability. To implement a proof of concept for the proposed off-line approach, we relied here on the translation of agent scripts to ASP programs, but other choices would have also been possible. In the literature there are a few works that link BDI programs to logic programs [2], but, for the present work, we take inspiration from [10], which presents a formal method for translating a HTN planning domain to logic programs. This choice was motivated by the close connection between BDI programs and HTN planning domains, which has been explored extensively in the literature [33,23].

Answer set programming (ASP) is a knowledge representation and reasoning (KRR) paradigm, based on the *stable-model semantics* [15], oriented towards difficult (NP-hard) search problems. ASP is used successfully in a variety of applications in both academia and industry. In ASP, similarly to Prolog, the programmer models a problem in terms of rules and facts, rather than specifying an algorithm. The resulting code is given as input to a solver, which returns multiple *answer sets* or stable models that satisfy the problem.

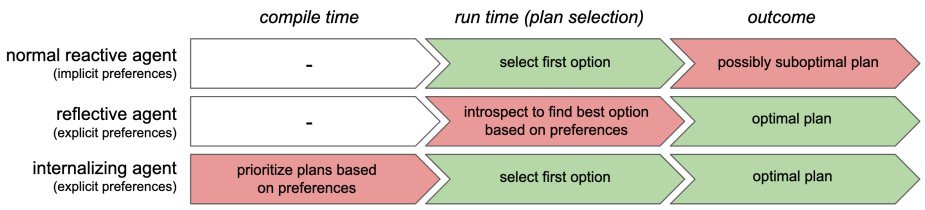

Fig. 1: Comparison of different approaches towards preferences in BDI agents.

Discrete Event Calculus Modeling and reasoning with effects of actions efficiently is still an open question in AI and logic; however, focusing on logic programming, most solutions build upon *situation calculus* [22] and *event calculus* [18,32]. For our proof of concept we will consider *discrete event calculus* (DEC) [26]. By translating the agent script into a DEC compatible ASP program and solving this program with DEC axioms we are able to evaluate the execution outcome of the translated program.

3 Method

The approach proposed here aims to integrate (embed or *internalize*) preferences into BDI scripts without modifying the BDI deliberation cycle as it is normally implemented in current BDI agent platforms. For doing so, we extend agent programs with two additional knowledge components: (1) *declarative preferences* (i.e. about states of affairs), and (2) expectations about the effects of *primitive actions*. Such enriched script is not directly used for execution (i.e. fed to the BDI interpreter). We proceed instead with an off-line method that compiles this script into a new one:

(a) by using the expectations about primitive actions, we create a set of contextualized outcomes for each goal-plan rule;
(b) for each possible context condition, goal-plan rules are ordered from best to worst, based on their outcome according to the preference specifications;
(c) the script is rewritten but this time the placement of each goal-plan rule follows its position in the ordering obtained at step (b).

Note that both preferences and primitive actions specifications are omitted from the newly generated script, which is now executable by BDI interpreters/compilers such as Jason [3] or AgentScriptCC [25]. This contrasts other approaches that extend BDI frameworks with preference checking at run-time as a rationale for plan selection (e.g. [34]). Figure 1 schematizes the differences between approaches.

3.1 Components of Extended Agent Programs

Our approach requires three components for specifying agent programs: goal-plan rules, expectations about primitive actions, and preferential structures based on CP-nets.

Goal-Plan Rules A goal-plan rule pr is a tuple $\langle e, c, p \rangle$, where: e is a *triggering event*, addressing an *invocation condition* e.g, adoption/failure of an unistantiated goal, assertion/retraction of a belief; c, the *context condition*, is a first-order formula over the agent's belief base, that, if true, makes the rule *applicable*; p, the *plan body*, consists of a finite sequence of steps $[a_1, a_2, ..., a_n]$ where each a_i is either a *goal* (i.e. an invocation attempting to trigger a goal-plan rule), or a *primitive action*. A goal-plan rule pr_i is then an *option* or a *possibility* for achieving a goal g, if the invocation condition of pr_i matches with g, and the context condition of pr_i matches the current state of the world, as perceived or encoded in the agent's beliefs.

We will refer here to a syntax close to that of AgentSpeak(L) [29], although with slightly different semantics. Unlike AgentSpeak(L), which does not primarily support declarative goals [8], `!g` will denote here an *achievement goal*. This means that `g` is a state or condition in the world that can hold or not hold (cases denoted respectively as `g` or `~g`). Positive and negative achievement goals, e.g. concerning the production and removal of a condition `g`, will be denoted respectively as `!g` and `!~g`. A primitive action named `a` will be denoted as `#a`. Then, for any condition `g`, `+!g` denotes a *goal-invocation* (possible triggering event of goal-plan rule). As an example of a script, consider:

```
+!g : c <= !a.
+!g <= !~b.
```

This code means that if the triggering event `+!g` occurs, if `c` holds, the agent commits to *achieve* `a`, otherwise (that is, `c` does not hold) the agent commits to *achieve* `~b`, or equivalently, to *escape* `b`. The backward sense of the arrow "<=" highlights the derivation due to instrumental reasoning (plans as a mean to reach the goal).

The standard AgentSpeak(L) syntax provides no unique identifier to distinguish goal-plan rules (although Jason offers some *labeling* construct). There is also no standard way to have direct access to the plan of a rule. A possible solution to identify a specific plan without explicit labeling is to refer to the invocation condition of the associated rule alongside its position, e.g. with respect to other rules with the same invocation condition. Thus, the two plans in the code excerpt above about achieving `g` via achieving `a` and escaping `b` will be respectively denoted as `!g[0]` and `!g[1]`.

Primitive Actions Primitive actions are the lowest-level actions that can be used in the procedural knowledge of an agent; they are the actual means for the agent to change the environment (or itself). As a matter of fact, BDI agents rely on goal-plan rules as an abstract task decomposition tool, mapping high-level recipes to an appropriate sequence of primitive actions to be performed in the environment.

Several approaches to specify expectations about primitive actions are available in the literature, especially in the AI planning field (e.g. *operators* for works derived from STRIPS [13], *primitive tasks* for works based on HTNs [12]), but they are not common in the BDI literature. This is because, in contrast to a common assumption in planning, BDI agents are deemed to interact with a non-deterministic environment; even more, as it is stated in [8], the effects of external (primitive) actions are "actually" determined by the environment and might be not known or incorrectly known by the agents beforehand. While this is true at run-time, it is also reasonable to consider that an agent

can have some expectations about the effects of its (primitive) actions beforehand. For instance, when an agent buys a train ticket it may encounter many problems and not receive a ticket, but it is fair to assume that the agent knows that "if I buy a train ticket, then I will have a ticket".

Because the preference compilation method proposed here occurs off-line, our work can be associated to a certain extent to the planning domain–although here agents do not create new plans but deal only with the given procedural knowledge. In STRIPS primitive actions are specified with the description of their effects, and of the conditions under which they are applicable, while in HTN primitive actions are only specified by their effects (delegating the precondition to methods). To take advantage of the complementary aspects of STRIPS and HTN approaches to primitive actions (more expressiveness and more control for the designer), we consider the hybrid solution proposed in [20]: "omitting strict action preconditions, assuming instead that actions leave the state unchanged if their preconditions are not met". This means that, although pre-conditions are part of action specification, they do not determine the applicability of the action, but they merely put conditions over the effects. A very similar approach is taken in the agent language 2APL [8] for *belief update actions*.

More formally, a primitive action a is specified as a tuple $\langle h, \Delta \rangle$, where h is the head or name of the action and Δ is the set of conditional effects of the action. Each $\delta \in \Delta$ is a combination of effects, captured as $\langle c, e \rangle$, in which c is a logical expression modeling the condition necessary for that combination to occur, and e is a list of effects each having a modifier + or − and a propositional atom t. If the expression c is true when the action occurs, then, after action completion, atoms with the + modifier are expected to hold and atoms with the − are expected to not hold. In case multiple more that one condition holds, all applicable effects are expected to happen and if there are conflicts the precedence goes to the effect described later. A simple syntax is used to represent the primitive action effects in the forms of `#h{LCE}`, where `h` is the head of action, and `LCE` is a dot separated list of condition-effects `CE` in the form of `c => e` and where `c` is the is a propositional expression representing the condition `e` is a comma (`,`) separated list of positive (initialization) or negative (termination) effects. Consider for instance the following statement:

```
#a { c1 => +p,-q. c2 => +q. }
```

The forward sense of the arrow "=>" highlights the production nature of `CE` components (action in conditions produces effects). The previous formula means that (the agent expects that) if the primitive action `a` occurs, if `c1` holds, then `p` will become true, if `c2` holds, then `q` will become true. It can also happen that both `c1` and `c2` hold which results in a contradiction between `+q` and `-q` in the post-condition. As this approach utilises an ASP solver (section 3.2) a contradiction stops the answer branch. We introduce specific axioms to raise a warning in this case.

CP-Nets for Declarative Preferences Constraining our attention on declarative goals, the preferences we target are about the presence or absence of certain conditions, here captured respectively by positive or negative literals. In this frame, the attributes of interests for the CP-net concern possible conditions that might *occur* in the world.

In behavioural terms, each attribute has two possible values: (1) achieving or maintaining the condition `g`, here denoted with the goal name `!g`, (2) avoiding or escaping the condition `g`, denoted as `!~g`. Following a syntax similar to the one used for procedural preferences in [24], we denote the preference for achieving/maintaining the condition `g` over avoiding/escaping it in condition `c` as:

```
!g > !~g : c.
```

In general, `c` might be an higher priority preferential attribute, a contextual condition or a logical `true` in the case of an unconditional preference.

3.2 Transformation to Logic Program

In order to evaluate plans in terms of their preferability, we need to infer the contextualized outcomes associated to each goal-plan rule. A possible solution for this task is to translate the initial BDI script with the added knowledge of primitive action specifications to a discrete event calculus (DEC)-based ASP program, so that each answer set of the program is a contextualized outcome for a goal-plan rule. The following section describes the translation method we followed. Other discrete simulation techniques (possibly more efficient) are indeed possible, but such difference in implementation would not functionally change the present proposal.

An agent program is a tuple $\langle S, A, P, G \rangle$, where S is a set of propositions representing all possible beliefs about the world that can be true or false at each time; A is a set of primitive actions, each formalized as $\langle h, \Delta \rangle$; P is a set of goal-plan rules, each formalized as $\langle e, c, p \rangle$; G is a set of (achievement) goals derived from the goal-plan rule heads. The following properties and relations can be identified:

1. For each goal $g \in G$, $g \in S$.
2. For each goal-plan rule $pr = \langle e, c, p \rangle \in P$, the atoms used in the expression c are propositions in S.
3. The effects of each primitive action $\alpha \in P$ is denoted as $\langle c, add, del \rangle$. The atoms of the expression c and the atoms present in add and del are all propositions in S.
4. Each step of the plan body of each goal-plan rule is either a primitive action `#a` or a sub-goal `!g`. In the former case there is a primitive action $\alpha \in A$ that $\alpha = \langle a, \Delta \rangle$ and in the latter case there is a $g \in G$.
5. All proposition $s \in S$ falls at least under one of the sets relevant for properties 1, 2, and 3.

Thus, for each *proposition* $s \in S$, a predicate $\mathit{fluent}(s)$ is added to the logic program, expressing that s is a ***fluent*** of DEC. The state of a proposition s at each time t is captured by the predicate $holdsAt(s, t)$. For each ***primitive action*** $\alpha_i = \langle a_i, \Delta_i \rangle$ a predicate $event(a_i)$. is added to the logic program expressing that a_i is an ***event*** of DEC. For each goal $g \in G$ a predicate $goal(g)$ and for each goal-plan rule $g_i = \langle g, c, p \rangle \in P$, a predicate $plan(g_i, g)$ is added to the program. Note that only primitive actions and propositions are translated into DEC predicates, whereas goal-plans and goals are only means to guide primitive actions.

Following DEC axioms, the execution of an action (event) a at time t is represented with the predicate $happens(a, t)$. In our translation we use a predicate $doAction/3$

that contains also the goal-plan rule of which this action is part of. Their relationship is captured as follows:

$$happens(A, T) \leftarrow doAction(A, P, T), event(A), time(T). \tag{1}$$

The conditional effects of primitive actions to the program are expressed by the DEC predicates $initiates/3$ and $terminates/3$. For each the primitive action $\alpha = \langle a, \Delta \rangle$, for each effect $\delta = \langle c, e \rangle \in \Delta$, for each conditional effect of initiating a proposition s with modifiers + or − under condition c, a logical rule in form of respectively rules (2) and (3) is added to the program. In the most general case, the condition expression c is a logical formula $\mathcal{C}(T)$, translated respectively by the predicates $holdsAt(c_i, T)$ and $not\ holdsAt(c_i, T)$.

$$initiates(a, s, T) \leftarrow time(T), \mathcal{C}(T). \tag{2}$$

$$terminates(a, s, T) \leftarrow time(T), \mathcal{C}(T). \tag{3}$$

To represent the adoption, completion and selection of goals or sub-goals by the agent, we introduce the predicates $adoptGoal/4$, $complGoalAt/3$, $selPlan/3$. An instantiation $adoptGoal(g, p, t', t)$ states that goal $g \in G$ is adopted by the agent at time t, as a sub-goal of plan p that started at t'; $complGoalAt(g, t, t')$ conveys that the goal g–adopted at time t–is completed at time t'; $selPlan(g_i, g, t)$ means that the plan g_i is instantiated to achieve goal g at time t.

The first step for mapping the goal-plan rules concerns the context condition. We introduce the predicate $applPlan(P, T)$, meaning that goal-plan rule P is *applicable* at time T. For each goal-plan rule $g_i = \langle g, c, p \rangle$ we add the following rule (c is again translated to $\mathcal{C}(T)$ by using predicate $holdsAt/2$):

$$applPlan(g_i, T) \leftarrow time(T), plan(g_i, G), \mathcal{C}(T). \tag{4}$$

Next, we connect triggering events to plan bodies while also taking into account the applicability of the goal-plan rules. Axiom (5) makes sure that when a goal is adopted, for each answer set exactly one of its applicable plans are selected:

$$\{selPlan(P', G, T) : applPlan(P', T), plan(P', G)\} = 1 \leftarrow adoptGoal(G, P, T', T). \tag{5}$$

Mapping the sequence of actions (steps) specified in a plan is less direct. Following the method presented in [10], we consider that the selection of the plan at a time t only triggers the first step of the plan at time $t + 1$, and from then on the completion of each step at a time t triggers the next step at time $t + 1$, save for the final step of the plan that triggers the completion of the plan at time $t + 1$. The reason behind this method is that each step of a plan can be either a sub-goal or a primitive action; if we can fairly assume a primitive action takes exactly one time-step to execute, the same can not be said for sub-goals (as, depending on their refinements, they can take several time-steps to complete).

The first step ($k = 0$) of the plan g_i associated to a goal g is encoded in rule (6) or (7) if the first step respectively is a primitive action a or adoption of a sub-goal g'.

$$doAction(a, g_i, T + 1) \leftarrow selPlan(g_i, g, T). \tag{6}$$

$$adoptGoal(g', g_i, T, T + 1) \leftarrow selPlan(g_i, g, T). \tag{7}$$

From the second step on ($k \geq 1$), the k^{th} step is encoded to happen at time $t' + 1$ if t' is the time of completion of $k - 1^{\text{th}}$ step, as shown in rule (8) if the k^{th} step is the execution of a primitive action a, rule (9) if the k^{th} step is the adoption of a sub-goal g' or rule (10) if $k - 1^{\text{th}}$ is the final step of plan g_i:

$$doAction(a, g_i, T' + 1) \leftarrow *. \tag{8}$$

$$adoptGoal(g', g_i, T, T' + 1) \leftarrow *. \tag{9}$$

$$complGoal(g, T, T' + 1) \leftarrow *. \tag{10}$$

The right side of these rules ($*$) has to be replaced with right side of rules (11) or (12) if $k - 1^{\text{th}}$ step is respectively a primitive action a' or the adoption of sub-goal g''.

$$** \leftarrow selPlan(g_i, g, T), doAction(a', g_i, T'), T' > T. \tag{11}$$

$$** \leftarrow selPlan(g_i, g, T), adoptGoal(g'', g_i, T, T''), complGoal(g'', T'', T'), T' > T'' \tag{12}$$

The following axiom is added to reflect the achievement nature of goals. When a goal $!g$ completes at time T', then g is a state in the world that holds at time T':

$$holdsAt(G, T') \leftarrow goal(G), complGoal(G, T, T') \tag{13}$$

We use the following axiom to let the ASP grounder create time-steps as it goes; i.e. $t + 1$ is a time step if t is a time-step and there is a step scheduled for t:

$$time(T + 1) \leftarrow time(T), (selPlan(..., T); doAction(..., T); adoptGoal(..., T); complGoal(..., T)). \tag{14}$$

After translating the script into an ASP program we need to find all the *traces* (i.e. outcomes of execution paths) of the script for all possible entry points (i.e. context conditions). Here, there is no assumption of a single entry point for the script and, based on internal or external events, any goal could in principle be adopted in any condition. Axiom (15) is used to force the solver to adopt exactly one goal as entry point in each answer set:

$$\{adoptGoal(G, init, 0, 1) : goal(G)\} = 1. \tag{15}$$

The answer sets of this program, denoted as R, will contain all the hypothetical paths that the agent script can start and run, including all the different refinements, i.e. all different initial states that may be at the starting point.

3.3 Plan Priority Extraction and Script Rewriting

Prioritizing plans is needed only if there is more than one plan for achieving a goal; therefore, for the rest of this section when we refer to all goals, we refer to all goals that have more than one plan associated to them.

As a first step we need to find all the conditions for which a plan g_i can be instantiated, here denoted as a multi-set $C(g_i)$. To do this, for each trace $r \in R$, for each k-th

occurrence of predicate $selPlan(g_i, g, t) \in r$, we create a set $c(g_i, r)[k]$ (for simplicity denoted as $c(g_i, r)$, assuming there is only one occurrence). Then $c(g_i, r)$ is representative of a state for which plan g_i is instantiated in the trace. Using the trace r, and assuming $selPlan(g_i, g, t) \in r$, $c(g_i, r)$ is constructed by the following elements:

1. **Motivational Context**: all the goals g' whose decomposition in r contain the plan g_i, i.e. all the goals that are adopted before but not completed at t; formally, all g' such that $(adoptGoal(g', P, T, t') \wedge complGoalAt(g', t', t'')) \in r$ with $t' \leq t \leq t''$;
2. **Propositions**: all positive (resp. negative) fluent f of the program which is in the trace r as $holdsAt(f, t)$ (resp. $not\ holdsAt(f, t)$);
3. **Achieved Goals**: all the goals g'' that are *achieved* as part a motivational context of the plan g_i prior to t; based on axiom (13), a completed goal g is present in the trace as $holdsAt(g, t)$.

An outcome of a trace r is a propositional state of the final time-step of the trace, denoted as $\Gamma(r)$, and includes all the (declarative) goals achieved in r plus the state of all the fluents at the final time-step.

Under the condition of consistency of the preferential structure and the *preferential comparison algorithm* presented in [4], there is a (possibly strict) order between outcomes, and we say an outcome $\Gamma(r)$ is preferred to outcome $\Gamma(r')$ and denote it as $\Gamma(r) \succeq \Gamma(r')$. Each $C(g_i)$ is a multi-set, meaning that for different traces $r, r' \in R$, we often have $c(g_i, r) = c(g_i, r')$ but $\Gamma(r) \neq \Gamma(r')$. This informally means that, in the same conditions, selecting a plan can have multiple different *reachable* outcomes. An outcome is called reachable for a plan g_i if *all* refinements of g_i will result in that outcome . However, as observed in [21], this approach starts from a very pessimistic view, ignoring the fact that the agent itself (not an adversary) chooses which refinements to make in the future, so instead of thinking what it might bring about in all refinements, we are interested in what is the *best* outcome that can happen under some refinement (the best outcome here indicates the optimal outcome according to the preferences specified in the CP-net).

The next step of the algorithm is *summarizing* the outcomes of plans, which means generating an optimal outcome for each plan g_i of each goal g, under each different unique condition. The optimal outcome is dependent on the conditions in which the plan was selected. More formally, an outcome $\Gamma(r)$ is optimal for the condition plan g_i under condition $c(g_i, r)$, if, for all other traces r' such that $c(g_i, r) = c(g_i, r')$, we have $\Gamma(r) \succeq \Gamma(r')$ or $\Gamma(r') \not\succeq \Gamma(r)$. This optimal outcome of plan g_i under condition of $c = c(g_i, r)$ is referred to as $\gamma(g_i, c)$.

At this point, we need to find a best-to-worst ordering between the optimal outcomes of plans of each goal g for each condition that g may be adopted. The conditions for which a plan g_i is instantiated are a subset of the conditions for which the goal g can be adopted, which means $C(g_i) \subseteq C(g)$. Similarly, the conditions for which a goal may be adopted is the union of the conditions for which all plans g_i may be instantiated.

We say that a plan $g_i \succeq g_j$ under condition $c \in C(g)$ if one of the following is true:

1. we have $c \in C(g_i)$ but $c \notin C(g_j)$
2. we have both $c \in C(g_i)$ and $c \in C(g_j)$ and also $\gamma(g_i, c) \succeq \gamma(g_j, c)$

The first rule is trivial and means that under a certain condition a plan that can be instantiated and has an outcome is preferred to one that even hypothetically does not have any outcomes. The second rule means that under similar conditions, between two plans, one that has the preferred optimistic outcome is always preferred. By running this procedure on all goals we can produce an ordering between plans.

4 Application

Suppose that a player agent has three ways to play a match: playing for fun, do whatever needed to win, or play conservative to avoid to lose. Suppose now that the player might want e.g. to enjoy its game as much as winning, but it might also prefer to gain support from observers, unless its position in the ranking (captured by propositions `first` and `last`—that can not be true at the same time) is really low. Let us start from the following procedural knowledge:

```
+!match <= !enjoy. // match[0]
        <= !win.   // match[1]
        <= !~lose. // match[2]
+!enjoy <= #funny_playing.
+!~lose <= #robust_playing.
+!win <= #opportunistic_playing.
```

And the following expectations about primitive actions:

```
#funny_playing { => +support, -win. }
#robust_playing { => +support, -enjoy. }
#opportunistic_playing { => -support, -lose }
```

We then specify our agent by the following preferences

```
!win > !~win : last.
!support > !~support : ~last.
!win > !~win : support, ~last.
!enjoy > !~enjoy : first.
```

By applying the method presented in 3.2 we obtain a logic program[1]. As an indicative excerpt, we report the goal-plan rule for `match[0]`:

```
adopts_goal(enjoy, match_0, T, T+1)
    :- selects_plan(match_0, match, T).
finished(match, match_0, T, T2+1) :-
    selects_plan(match_0, match, T),
    adopts_goal(enjoy, match_0, T, T2),
    finished(enjoy, P, T2, T3), T3 > T2.
```

By solving the program with `clingo` [14], we obtain 192 answer sets ($|R| = 192$). The only goal in the program with alternative plans is `!match` and so only the answer sets containing this goal are needed. Focusing on its three goal-plan rules we consider 36 answer sets from which we extract 6 unique outcomes for each plan. An example of condition and outcome extracted from a trace r for plan $g_i =$ `match[0]` is:

[1] Source available at https://gitlab.com/Mohajeri/as2asp

```
condition c1: ~win,~support,~enjoy,~lose,first,~last
outcome o1: ~win,support,enjoy,~lose,first,~last
```

Relating to the formalization of previous part we can say that $c(\texttt{match[0]}, r) = \texttt{c1}$ and $\Gamma(r) = \texttt{o1}$.

Each answer set is evaluated in terms of the given preferential structure, resulting in a partial ordering between different outcomes of traces. For instance, the answer sets with outcome `o2` are preferred over the answer sets with outcome `o3`. The *dominance checking* is done with the tool CRISNER [31]:

```
outcome o2: ~win,support,enjoy,~lose,first,~last
outcome o3: win,~support,~enjoy,~lose,first,~last
```

Following our formalization, if we take two answer sets (traces) r and r' such that $\Gamma(r) = \texttt{o2}$ and $\Gamma(r') = \texttt{o3}$, according to the preferential structure we infer that $\Gamma(r) \succeq \Gamma(r')$. Then, following the ranking, we can give contextualized priorities to plans, observing that e.g. if we have a condition:

```
condition c2: ~win,~support,~enjoy,~lose,first,~last
```

the plan `match[0]` is preferred over plan `match[1]`, whereas plan `match[1]` is preferred over `match[2]`. Formally this means that, under condition `c2`, we have

$$\texttt{match[0]} \succeq \texttt{match[1]} \succeq \texttt{match[2]}$$

Considering all existing plans, the initial procedural knowledge can be then prioritized. For the 3 plans associated to `+!match` there are 6 possible orderings. The following code provides an example of conditional ordering obtained via our method (including a *boolean simplification* step for the pre-conditions):

```
+!match : (last | ~enjoy) & ~first
        <= !~lose.
        <= !win.
        <= !enjoy.
+!match : (~last | lose) & (last | enjoy) & ~win
        <= !enjoy.
        <= !win.
        <= !~lose.
```

5 Discussion and Further Developments

The strong support in the decision-making literature for *compact representations* of verbalized preferences—as for instance those captured e.g. by CP-nets [4]—motivates their use in computational agents, especially in applications in which agents are deemed to reproduce human behaviour. Indeed, our more general research effort aims to capture intentional characterizations of (computational) behaviour of computational agents in data-sharing infrastructures in support of policy-making and regulation activities. Regulating data-sharing requires to reproduce to a certain extent constructs similar to those observed in human institutions (e.g. For which purpose the agent is asking access to

the resource? On which basis the infrastructure is granting access?). For *traceability* and *explainability* reasons, decisions concerning actions need to be processed by the infrastructure as much as other relevant operational aspects.

Introducing explicit preferences in BDI scripts brings three advantages: (1) It increases the *representational depth*, capturing what is the rationale behind the priority in plan selection; (2) It makes agent models more readable and *explainable*, as choices are in principle transparently derived from the preferential structure; (3) It makes the programs more *reusable*: it is plausible that agents (e.g. representatives of organizations) in a certain domain might share the same procedural knowledge even when having different preferences, as much as that agents might change their policy without changing their procedural knowledge.

The connections between desires, preferences and goals requires further clarification. In the current work we started from the AgentSpeak(L) view of desires, for which "goals are viewed as adopted desires" and while this is mostly accepted by the BDI community, it hints to a gap between goals and desires pointed out already almost twenty years ago [9]. In this work we used preferences essentially to specify desires (in the sense of *soft goals*) (e.g. "I want to enjoy the game" as "I prefer to enjoy the game more than not enjoying the game") and relative strength between desires (if not losing, "I want to gain support more than winning" as "I prefer to gain support more than winning"). The priority between plans is then selected so as to satisfy at best the desires of the agent. Note that in general the literature suggests that preferences are derived from desires [19]; for our purposes, however, we discovered that two could be seen as functionally equivalent. Further investigation is needed to see the consequences of this reduction.

The proposed method here is indeed a contribution towards enhancing BDI agent-programming languages with syntactic and semantic facilities to support explicit preferences, but, in contrast to other works, it also fulfills the aim of maintaining *reactivity*, one of the core properties of BDI agents (see [35], or [3], referring to AgentSpeak(L) agents as "reactive planning systems"). BDI agents are theoretically developed to act in dynamic environments and an offline view on preferences may seem too limiting at first, as preferences of a dynamic agent can change in a highly dynamic environment. This might explain why so few authors chose this path. However, we will put forward two reasons why this is still a relevant issue. First, agent programs, today, are static in nature; any modification at run-time relies on implicit forms of meta-programming whose general effects are difficult to be anticipated. For instance, preferences might be incomplete and/or conflicting. By adding an additional compilation step, these issues might be captured while rewriting the script, so the user can be required to take action to settle them. Second, reactivity is a valuable property to enable computationally scalable implementations (cf. modern reactive programming). Indeed, the uses we are aiming to (simulations in support of policy-making, applications running on data-sharing infrastructures) would greatly benefit of this choice. If each agent had to repeat online the full derivation from preferences to preferred plan, the computational overhead would strongly negatively affect performance. Besides this, with our method, we can still allow agents to re-adapt periodically: i.e. update their preferences based on some criteria (e.g. mimicking more successful agents), then update and reload their run-time script.

This approach is also cognitively more realistic: we, as humans, do not deliberate upon our preferences for each action we perform.

However, we also acknowledge that the exploration of all context conditions and possible outcomes is in general an intractable problem, even if we rely on optimized solvers. The proposal presented here has to be seen merely as a functional proof of concept; for actual use additional heuristics need to be added to reduce the search space to most relevant nodes, for instance exploiting weights in ASP resolution.

In general, preferences might be not only concerning desired states of affairs, but also possible states of affairs, or expectations about primitive actions, thus determining that certain contexts are more relevant than others. This part of the problem has also connections with probabilistic logic programming and reasoning with uncertainties.

In terms of priority for future developments, however, we recognize the work presented in this paper has been limited to propositional logic; clearly, extending it to consider first-order logic (FOL) descriptions of both agent scripts and preferences is an important objective for actual applicability. Additionally, while plan selection is the only component of the BDI execution model studied in this work, a similar approach could also be taken for other components of the deliberation cycle, for which other authors resorted to reflective methods, like intent selection and event selection [36].

Finally, although the logic of CP-nets is widely accepted in the literature, the presence in our research of sequential choices and different types of goals adds complexities that the CP-net syntax is not adequate to address in its default form; we acknowledge the need for a more principled extension or an exploration of other representational models for preferences.

Acknowledgments This paper results from work done within the NWO-funded project *Data Logistics for Logistics Data* (DL4LD, https://www.dl4ld.net) in the Commit2Data program (grant no: 628.001.001).

References

1. Bienvenu, M., Fritz, C., McIlraith, S.A.: Planning with Qualitative Temporal Preferences. In: Proceedings of the 10th International Conference on the Principles of Knowledge Representation and Reasoning (KR2006). pp. 134–144 (2006)
2. Blount, J., Gelfond, M., Balduccini, M.: A theory of intentions for intelligent agents. Lecture Notes in Computer Science **9345**, 134–142 (2015)
3. Bordini, R.H., Hübner, J.F., Vieira, R.: Jason and the Golden Fleece of Agent-Oriented Programming. In: Multi-Agent Programming: Languages, Platforms and Applications, pp. 3–37 (2005)
4. Boutilier, C., Brafman, R.I., Domshlak, C., Hoos, H.H., Poole, D.: CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. Journal of artificial intelligence research **21**, 135–191 (2004)
5. Bratman, M.E.: Intention, Plans, and Practical Reason, vol. 10. Harvard University Press (1987)
6. Cranefield, S., Winikoff, M., Dignum, V., Dignum, F.: No Pizza for You: Value-based Plan Selection in BDI Agents. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI2017). pp. 178–184 (2017)

7. Dasgupta, A., Ghose, A.K.: Implementing reactive BDI agents with user-given constraints and objectives. International Journal of Agent-Oriented Software Engineering **4**(2), 141 (2010)
8. Dastani, M.: 2APL: A practical agent programming language. Autonomous Agents and Multi-Agent Systems **16**(3), 214–248 (2008)
9. Dignum, F., Kinny, D., Sonenberg, L.: From desires, obligations and norms to goals. Cognitive Science Quarterly **2**, 405–427 (2002)
10. Dix, J., Kuter, U., Nau, D.: Planning in answer set programming using ordered task decomposition. In: Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science). vol. 2821, pp. 490–504 (2003)
11. Domshlak, C., Hüllermeier, E., Kaci, S., Prade, H.: Preferences in AI: An overview. Artificial Intelligence **175**(7-8), 1037–1052 (2011)
12. Erol, K., Hendler, J., Nau, D.S.: HTN planning: Complexity and expressivity. In Proceedings of the 12th AAAI Conference on Artificial Intelligence pp. 1123–1129 (1994)
13. Fikes, R.E., Nilsson, N.J.: Strips: A new approach to the application of theorem proving to problem solving. Artificial Intelligence **2**(3-4), 189–208 (1971)
14. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Clingo = ASP + control: Preliminary report. CoRR **abs/1405.3694** (2014)
15. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. Proceedings of International Logic Programming Conference and Symposium pp. 1070–1080 (1988)
16. Gerevini, A., Long, D.: Plan constraints and preferences in PDDL3. Tech. rep. (2005)
17. Gonzales, C., Perny, P.: GAI Networks for Utility Elicitation. In: Proceedings of the 9th International Conference on the Principles of Knowledge Representation and Reasoning (KR2004). pp. 224–233 (2004)
18. Kowalski, R., Sergot, M.: A logic-based calculus of events. New Generation Computing **4**(1), 67–95 (1986)
19. Lorini, E.: Logics for games, emotions and institutions. The IfCoLog Journal of Logics and their Applications **4**(9), 3075–3113 (2017)
20. Marthi, B., Russell, S., Wolfe, J.: Angelic Hierarchical Planning: Optimal and Online Algorithms (Revised). Tech. Rep. UCB/EECS-2008-150 pp. 1–22 (2008)
21. Marthi, B., Russell, S.J., Wolfe, J.: Angelic Semantics for High-Level Actions. In: Proceedings of the 17th International Conference on Automated Planning and Scheduling. pp. 232–239 (2007)
22. McCarthy, J., Hayes, P.J.: Some philosophical problems from the standpoint of artificial intelligence. In: Machine Intelligence, pp. 1–51. Edimburgh University Press (1969)
23. Meneguzzi, F., De Silva, L.: Planning in BDI agents: A survey of the integration of planning algorithms and agent reasoning. Knowledge Engineering Review **30**(1), 1–44 (2013)
24. Mohajeri Parizi, M., Sileno, G., van Engers, T.: Integrating cp-nets in reactive bdi agents. In: PRIMA 2019: Principles and Practice of Multi-Agent Systems. pp. 305–320. Springer International Publishing (2019)
25. Mohajeri Parizi, M., Sileno, G., van Engers, T., Klous, S.: Run, agent, run! architecture and benchmark of actor-based agents. proceedings of Programming based on Actors, Agents, and Decentralized Control (AGERE20), ACM (2020)
26. Mueller, E.T.: Event Calculus Reasoning Through Satisfiability. J. Log. and Comput. **14**(5), 703–730 (Oct 2004)
27. Nunes, I., Luck, M.: Softgoal-based plan selection in model-driven BDI agents. 13th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2014 **1**, 749–756 (2014)
28. Pigozzi, G., Tsoukiàs, A., Viappiani, P.: Preferences in artificial intelligence. Annals of Mathematics and Artificial Intelligence **77**(3-4), 361–401 (2016)

29. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: Agents Breaking Away. pp. 42–55 (1996)
30. Rao, A.S., Georgeff, M.P.: BDI agents: From theory to practice. In: Proceedings of the First International Conference on Multi-Agent Systems (ICMAS1995). pp. 312–319 (1995)
31. Santhanam, G.R., Basu, S., Honavar, V.: Dominance testing via model checking. In: Proceedings of the National Conference on Artificial Intelligence. vol. 1, pp. 357–362 (2010)
32. Shanahan, M.: The Event Calculus Explained pp. 409–430 (1999)
33. de Silva, L., Padgham, L., Sardina, S.: HTN-like solutions for classical planning problems: An application to BDI agent systems. Theoretical Computer Science **763**, 12–37 (2019)
34. Visser, S., Thangarajah, J., Harland, J.: Reasoning about preferences in intelligent agent systems. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI2011). pp. 426–431 (2011)
35. Wooldridge, M.J., Jennings, N.R.: Intelligent agents: Theory and practice. Knowledge Engineering Review **10**(2), 115–152 (1995)
36. Yao, Y., Logan, B.: Action-level intention selection for BDI agents. In: Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS2016). pp. 1227–1236 (2016)