Preference-Based Goal Refinement in BDI Agents

Mostafa Mohajeriparizi
University of Amsterdam
Amsterdam, The Netherlands
m.mohajeriparizi@uva.nl

Giovanni Sileno
University of Amsterdam
Amsterdam, The Netherlands
g.sileno@uva.nl

Tom van Engers
University of Amsterdam
Amsterdam, The Netherlands
t.m.vanengers@uva.nl

ABSTRACT

Computational agents based on the BDI framework typically rely on abstract plans and plan refinement to reach a degree of autonomy in dynamic environments: agents are provided with the ability to select *how-to* achieve their goals by choosing from a set of options. In this work we focus on a related, yet under-studied feature: *abstract goals*. These constructs refer to the ability of agents to adopt goals that are not fully grounded at the moment of invocation, refining them only when and where needed: the ability to select *what-to* (concretely) achieve at run-time. We present a preference-based approach to goal refinement, defining preferences based on extended *Ceteris Paribus* Networks (CP-Nets) for an AgentSpeak(L)-like agent programming language, and mapping the established CP-Nets logic and algorithms to guide the goal refinement step. As a technical contribution, we present an implementation of this method that solely uses a Prolog-like inference engine of the agent's belief-base to reason about preferences, thus minimally affecting the decision-making mechanisms hard-coded in the agent framework.

KEYWORDS

Agent-based Programming; BDI Agents; Abstract Goals; Preferences; CP-Nets; CP-Theories; Goal refinement

ACM Reference Format:
Mostafa Mohajeriparizi, Giovanni Sileno, and Tom van Engers. 2022. Preference-Based Goal Refinement in BDI Agents. In *Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), Online, May 9–13, 2022*, IFAAMAS, 9 pages.

1 INTRODUCTION

As computational agents intervene more and more in human activities, there is an increasing demand for human-oriented forms of programming, i.e. relying on concepts and abstractions mapping intuitively to what humans utilize to explain and direct their behaviour. The *belief-desire-intention* (BDI) model of agency [25], centred around a general theory of mind [5], offers one of those views, and has resulted in the proposal and development of several platforms for agent-based programming (e.g. AgentSpeak(L)/Jason [3, 24], 3APL/2APL [9], GOAL [15], IMPACT [12], JACK [16], Astra [11], LightJason [1], ASC2 [20]—a systematic review of logic-based MAS frameworks can be found in [7]).

Computational agents based on the BDI framework typically rely on abstract plans and plan refinement to reach a degree of autonomy in dynamic environments. In practice, relative autonomy in this context consists in the ability of an agent to select *how-to* achieve their goals by choosing from a set of options. BDI agent scripts typically consist of hierarchical, partial, abstract plans. This contrasts with classic forms of planning, providing agents with fully-grounded policies, designed to reach a certain specific objective.

This work focuses on a related, yet under-studied feature: *abstract goals*. These constructs refer to the ability of agents to adopt goals that are not fully grounded at the moment of invocation, refining them only when and where needed, that is, the ability to select *what-to* (concretely) achieve at run-time. Examples of abstract goals can be found typically in *activity*-level characterizations of behaviour, e.g. walking (where?), eating something (what?), meeting someone (whom?), selling (what? to whom?), etc.

The specification of abstract goals is a feature already present in some agent frameworks as those based on the AgentSpeak(L) language, albeit they rely on simplistic mechanisms for goal refinement. The present work aims to cover the goal refinement step (from abstract goals to concrete goals) as part of the agent's decision-making cycle. For doing so, we present a preference-based approach to goal refinement. We start from defining preferences based on *Ceteris Paribus* Networks (CP-Nets) [4]—more precisely, in the extended form of *Ceteris Paribus* Theories (CP-Theories) [29]—and we consider the established CP-Net logic and algorithms to guide the goal refinement of the agent. At implementation level, our target is an AgentSpeak(L)-like [24] agent programming language. Since Jason [3], AgentSpeak(L) programs are enriched with Prolog rules and facts for knowledge-level processing, occurring e.g. for testing context conditions during the plan selection phase. We present therefore an implementation of a preference-based goal refinement method that solely uses a Prolog-like inference engine of the agent's belief-base to reason about preferences, requiring only a minimal modification to the decision-making mechanisms hard-coded in the agent framework. To achieve this, a transformation method is proposed to map an extended version of CP-Nets and CP-Theories into Prolog facts and rules for the script of the AgentSpeak(L) agent, as well as a Prolog implementation of the algorithms necessary to reason with preferences.

The paper proceeds as follows: Section 2 provides a background about the concepts used in this work; Section 3 presents the method and examples for preference-based abstract goal refinement in AgentSpeak(L) agents; Section 4 describes a practical implementation of this method, and Section 5 elaborates on a discussion and conclusion over the proposed method.

2 BACKGROUND

2.1 BDI Agents

Agents specified following the BDI paradigm are characterized 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.

Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), P. Faliszewski, V. Mascardi, C. Pelachaud, M.E. Taylor (eds.), May 9–13, 2022, Online.

Intentions are selected conducts (or *plans*) that the agent commits to (in order to advance its desires).

Since their origin [25], 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 often relies on a *reactive* model of computation, usually in the form of some type of *event-condition-action* (ECA) rules often referred to as *plans*. Plans are uninstantiated specifications of the *means* (in terms of course of actions) for achieving a certain *goal* [25]. These constructs represent the procedural knowledge (*how-to*) of the agent. There are multiple proposals in the literature for programming language and architecture of BDI agents, the most commonly used being AgentSpeak(L) [24], which will serve as basis for the present proposal.

2.2 Preference Languages

Preferences play a crucial role in decision-making [23]. Several models of preferences have been presented in the literature (e.g. on decision-making, planning, etc.), with various levels of granularity and expressiveness (see e.g. [13]). The most straightforward *quantitative* approaches are based upon *utility theory* and related forms of decision theory, but, they 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. The *logical preference description* (LPD) language [6] uses ranked knowledge bases alongside preference strategies to present preference descriptions. The LPP language [2] is a first-order preference language defined in situation calculus to reason about conditional and qualitative preference. Other preference models, such as GAI networks [14], CP-Nets [4] and qualitative preference systems (QPS) [28], have been specifically introduced for taking into account dependencies and conditions between preferences. In the present work, we decided to focus on CP-Nets, and their extension CP-Theories [29], for two main reasons: they rely on weaker assumptions, and exhibit primarily a qualitative nature.

2.2.1 Ceteris Paribus networks (CP-Nets). 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. *restaurant*) that the agent has some sort of preference over its possible values (e.g. *italian* and *french*). 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 someone says "I prefer a French restaurant over an Italian one", they do not mean at all costs and situations, but that they prefer a French restaurant (over an Italian one), all other things being equal. An example of *conditional preference* is "If I'm at a French restaurant, I prefer fish over meat". CP-Theories [29] extend CP-Nets adding stronger conditional statements with the construct "*regardless of*", allowing some attributes to be released from the equality rule.

In general, CP-Nets can be associated with two tasks: (1) finding the most preferred outcome on a certain domain of variables (2) comparing two outcomes under different criteria. Both CP-Nets and CP-Theories provide efficient algorithms for these tasks [4, 29].

2.2.2 Preferences in BDI Agents. Goals are used to identify desired states or outcomes, and preferences are used to identify more (or less) desired states or outcomes. While goals are a central aspect in BDI agents, so far, none of the main BDI frameworks and languages include preferences as part of the agent specifications. Previous studies attempted to enhance BDI agents with explicit preferences. Visser et al. [26, 27] present an approach to embed preferences defined in the LPP language into BDI agents to guide plan selection. Nodes in the goal-plan tree of the agent are annotated by the designer about the effects of that plan and then this information is propagated automatically to other nodes in the tree at compile time. Then, at run-time, the agent uses the LPP logic to select the most preferred plan for a goal based on this information. Dasgupta et al. [8] proposes a look-ahead method to enhance AgentSpeak(L) agents with constraints and objectives that the agent can use for plan selection at run-time; their approach also requires annotations for plans to reason about the preferability of plans. Mohajeri et al. [19, 22] add preferences in form of CP-Nets in AgentSpeak(L)-like agents, however, their approach considers a cross-compilation step: they annotate primitive actions of agents with their expected effects, and this information is then propagated through the goal plan tree to create a conditional ordering between plans. Padgham et al. [21] add situational preferences as part of plan definitions in a BDI language. The agent can use them to quantify the value of each plan at run-time. Their method is similar to this work in the sense that it does not require any look-ahead, but it is different because they add preference valuations as part of each plan, which are then used in plan selection with the implicit preference of maximizing them—this makes the approach essentially a quantitative one.

3 METHOD

3.1 AgentSpeak(L) Agents

In terms of technical contribution, the present work targets frameworks that utilize AgentSpeak(L), one of the most commonly used BDI languages. Based on the definitions proposed by Rao et al. in [24, 25], an agent consists of a set of beliefs B called belief base, a set of plans P called plan library, a set of events E, a set of actions A, a set of intentions I, and three selection functions: S_E,S_O,S_I. When the agent receives an (internal or external) event or adopts a goal, it is added to E. The selection function S_E selects an event to process from E. Then this event is unified with the triggering events in the heads of the plans in P. The following definitions apply:

Definition 1 (Plan). *A (reactive) plan is specified by $e : C \Rightarrow H$ where e is a triggering event, C is a formula capturing context conditions, and H is a sequence of sub-goals or actions to be performed at the occurrence of the trigger event.*

Definition 2 (Relevant plan). *A plan in the form of $e : C \Rightarrow H$ is a* relevant plan *with respect to an event ϵ iff there exists a most general unifier σ such that $\epsilon\sigma = e\sigma$. Then, σ is referred to as the* relevant unifier *for ϵ.*

Definition 3 (Applicable plan). *A plan in the form of $e : C \Rightarrow H$ is an* applicable plan *with respect to an event ϵ iff there exists a relevant unifier σ for ϵ and there exists a substitution δ such that $C\sigma\delta$ is a logical consequence of belief base B. The composition $\sigma\delta$ is*

```
(P1) +!go_order(Loc,Meal) :
        restaurant(Loc) & not at(Loc) =>
        #move_to(Loc);
        !order(Meal).
(P2) +!go_order(Loc,Meal) :
        restaurant(Loc) & at(Loc) =>
        !order(Meal).
(P3) +!order(meal(S,M,W)) :
        meal(S,M,W) =>
        #ask_waiter(meal(S,M,W)).
```

Listing 1: Reactive Plans of Food-ordering Agent

referred to as the applicable unifier *for ϵ and δ is referred to as a* correct answer substitution.

As for each event there could be multiple applicable unifiers, the selection function S_O chooses one of these plans or options and applying the applicable unifier to that plan creates an instantiated plan, i.e, an intended means or the event which will be added to a new or existing intention. Then the S_I function selects an intention which will be executed. We will now consider an example agent to illustrate the plan instantiation process. The specific framework used for the examples is ASC2 [20] for practical reasons, but the approach is easily transferable to any other framework that follows the principles of AgentSpeak(L).

Example 1. Imagine an agent that, upon request, can go to a restaurant and order a three-course meal. The script for such agent is presented in Listing 1.[1] The agent has two plans for going to a restaurant and ordering a meal: the first plan (P1) is *applicable* if the agent is not at a restaurant at the moment, which means a step of moving (`#move_to` primitive action) is needed prior to ordering the meal; the second plan (P2) is applicable if the agent is already at the restaurant which means the agent will just adopt the goal of ordering the meal. There is also one plan for ordering the meal (P3) which is applicable if the agent has the belief that the meal it wants to order exists. Suppose the agent selects an event with the trigger: `!go_order(french,meal(veg,meat,white))`

For this event, both plans (P1) and (P2) are relevant with unifier σ: `{Loc/french, Meal/meal(veg,meat,white)}`

Assuming that the belief base of the agent contains the beliefs `restaurant(french)` (meaning that there exists a French restaurant) and `at(home)` (meaning that the agent is at home), then only the first plan will be an applicable plan for this event, and the applicable unifier will be the same as the relevant unifier. This entails that only plan (P1) will be instantiated as:

```
+!go_order(french,meal(veg,meat,white)) :
    restaurant(french) & not at(french) =>
    #move_to(french);
    !order(meal(veg,meat,white)).
```

Note that in case the agent had more than one applicable unifiers, meaning it had more than one option to react to this goal, then the S_O function would have been called to select one of the options.

[1] ASC2 slightly modifies the AgentSpeak(L) syntax, replacing "<-" with "=>", to further distinguish the reactive, forward nature of these rules w.r.t. the backward chaining derivation of Prolog rules ":-".

```
main(fish). main(meat). soup(veg). soup(fish).
wine(white). wine(red).
restaurant(french). restaurant(italian).
at(french).
meal(S,M,W) :- soup(S), main(M), wine(W).
```

Listing 2: Beliefs of Food-ordering Agent

3.2 Abstract Events, Abstract Goals

Partial autonomy in dynamic environments is considered a core attribute of BDI agents, and this is in fact one of the reasons that separates plan refinement in BDI agents from classical planning [10]. While the idea of choosing between distinct plans to achieve a certain goal—typically referred to as plan selection—has been investigated by the community as the core point of autonomous choice in BDI agents, there is another important type of autonomy embedded in BDI agents: *abstract events*. While the previous example only exhibited fully grounded events, BDI agents, in particular those derived from AgentSpeak(L) [24, 25] can indeed handle abstract events, referring to situations where an event contains unbounded variables and these variables can be grounded by different means such as context conditions of plans or test goals at any level in the plan refinement of the event. It can be argued that if plan selection promotes autonomy in the *how-to-do* dimension of the agent, abstract events, which include invocations of abstract goals, promote autonomy in selecting (concretely) *what-to-do* with it.

Example 2. Consider the same agent presented in Listing 1. This time we assume the agent has more information about the environment: it has beliefs about two types of soups, two types of main course, two types of wine, two restaurants, also it believes that it is standing already in one of the restaurants (the `french` one), and finally it has an inferential rule for which all possible triple of soup, main course and wine form a meal combination. Those beliefs are presented as in Listing 2.

Now assume the agent receives an abstract event `!go_order(L,M)` which basically puts no constraints over where the agent should go and what it should order, and so gives it full autonomy to choose how to proceed. When the agent receives this event, both plans P1 and P2 are considered relevant plans with unifier `{Loc/L, Meal/M}`. But considering the context conditions and the belief base, P1 will be applicable with unifier `{Loc/italian, Meal/M}`, and P2 with `{Loc/french, Meal/M}` (note that in both cases the second parameter is not grounded as it is unified to another variable). At this point, the agent's reasoning engine needs to use its plan selection function to choose one of the two plans. In the case where P1 is selected, the next step is the action `move_to(italian)`, and then the goal `!order(M)` is adopted; In the case where P2 is selected, `!order(M)` is immediately adopted. In both cases, the next event for the agent to process will be `!order(M)`, and P3 is a relevant plan for this event with unifier `{M/meal(S,M,W)}`. Taking into account the unification occurring at context conditions, this event will have in principle $2^3 = 8$ different applicable unifiers with all possible combinations for the meal, e.g: `{M/meal(veg,fish,white)}`, for which, again, the plan selection function needs to choose an option to start the actual execution. The goal-plan tree of this abstract goal can be seen in Figure 1.

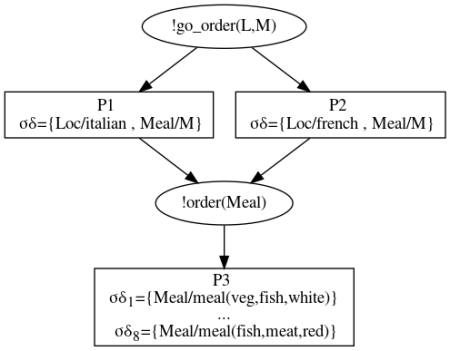

Figure 1: Goal-Plan Refinement of the Agent

In current implementations of BDI frameworks based on AgentSpeak(L), the three selection functions S_E, S_O, S_I (respectively for events, plans/options, and intentions) are typically exposed as abstract functions that the designer can override to implement any type of selection function. Although this approach promotes flexibility, the fact that part of the decision-making remains external to the agent script reduces readability, encapsulation, and transparency of the agent programs, and makes the control more opaque to the designer.

Default implementations are based on selecting the first applicable option, which is indeed a good example of simplicity. If we apply the default implementation also on this example, the first applicable unifier for `!go_order(L,M)` is {`Loc/italian, Meal/M`}, and the first applicable unifier for `!order(S,M,W)` is {`M/meal(veg,fish,white)`}.

3.3 CP-Nets and CP-Theories

In order to specify preferences, we start from the definition of CP-Nets given in [4]. Given a set of variables $X \in V$, each having a finite set of values x, conditional preference statements are in the form $u : x \succ x'$, where x, x' are assignments of a variable $X \in V$, and u is an assignment to a set of variables $U \subseteq V$ (parents of X). The interpretation of this statement is that given u, then x is preferred to x' all else equal, meaning, for all assignment s of the set of variables S, where $S = V - (U \cup \{X\})$, sux is preferred to sux', where sux and sux' are two *outcomes* (complete assignment) to all variables of V. CP-Theories are introduced in [29] to extend CP-Nets with *stronger conditional statements*. These include preferential statements in the form $u : x \succ x'[W]$, where $W \subseteq V$ which interprets that for all assignments w, w' to variables of W and assignments t to variables of $T = V - (U \cup \{X\} \cup W)$, then the outcome $tuxw$ is preferred to the outcome $tux'w'$. This means that given u and any t, then x is preferred to x' *regardless* of assignments to W.

Assuming Λ is a set of acyclic (with respect to parent-child relations) preference relations over variables of V, we say Λ is satisfied by a preference ordering $\succ$ iff $\succ$ satisfies each of the conditional preferences expressed in Λ. Considering o, o' are outcomes of V, then we say $\Lambda \models o \succ o'$ iff $o \succ o'$ holds in every preference ordering that satisfies Λ. Then o and o' can have one of the three possible relations according to Λ: either $\Lambda \models o \succ o'$; or $\Lambda \models o' \succ o$; or $\Lambda \not\models o \succ o'$ and $\Lambda \not\models o \succ o'$. The third case means there is not enough information to prove either outcome is preferred.

Based on these definitions, two distinct ways for comparing outcomes are proposed in [4]:

- Dominance queries: Asking if $\Lambda \models o \succ o'$ holds, which is referred to as o is strongly preferred to or *dominates* o'.
- Ordering queries: Asking if $\Lambda \not\models o' \succ o$ holds, which is referred to as o is weakly preferred to or consistently orderable over o'.

Although ordering queries are weaker than dominance queries, they are still sufficient in many applications, and will be used in this work. In particular, if an outcome o is present such that for all other outcomes o' we have $\Lambda \not\models o' \succ o$, then we say o is *undominated* or *most preferred* with respect to Λ.

All through this work, and for the sake of simplicity, only strict preferences $\succ$ are considered. Nevertheless, these semantics are shown to be extendable to non-strict preferences $\succeq$ and indifference $\sim$ in both CP-Nets and CP-Theories.

3.3.1 Preference Statements as Inference Rules. To transform CP-Theories to a formalism that can be used with the Prolog-like inferential systems as those used in AgentSpeak(L) agents, one should look at what needs to be decided in the process of goal refinement. An agent may have dynamically interconnected beliefs about the environment, but when it is deciding what is the most preferred approach to partially ground the variables of an event or goal in the form of e.g $!g(v_1, ..., v_n)$, only the parameters of that goal are relevant to the decision.

A conditional preference statement of the agent can be expressed in the form of an inferential rule such as:

$$G \succ G' \leftarrow C$$

where G, G' are either belief predicates in the form $g(v_1, ..., v_n)$ and $g(v'_1, ..., v'_n)$, or triggering events in the form $!g(v_1, ..., v_n)$ and $!g(v'_1, ..., v'_n)$ (or any other type of trigger, $?, +, -$). Each v_i and v'_i can be either a (partially) ground term, a named variable or an anonymous variable (underscore, "_"), and C is an arbitrary expression that *activates* the preference statement if it can be proven to be true at the time of evaluation, which can include variables that appear on the left side of the $\leftarrow$. The set of all preferences of an agent is referred to as Λ.

With this definition, for each predicate G with the form $g(v_1, ..., v_n)$, we can denote its set of variables (or features or attributes) as $V_G = \{v_1, ...v_n\}$. To express a preference statement $u : x \succ x'[W]$ in this form, on a predicate G, assuming $G_X \in V_G$ is the variable of G corresponding to X, the set $G_U \subseteq V_G$ is the set of all variables corresponding to U, the set $G_W \subseteq V_G$ is the set of all variables corresponding to W and $G_T = V_G - (G_U \cup \{G_X\} \cup G_W)$, the statement can be presented as $G \succ G' \leftarrow true$, such that G_X is written as x, G'_X is written as x', all the variables of G_U and G'_U are written as their corresponding value in u, all variables of G_W and G'_W are written as anonymous variables (underscore) and all the variables of G_T and G'_T are replaced with named variables that have the same name in both G and G'.

As an example, given a predicate with the form $g(v_1, ..., v_4)$, where the first parameter corresponds to G_X, the second parameter to G_U, the third parameter to G_W and the last parameter to G_T, a preference statement $u : x \succ x'[W]$ is expressed as:

$$g(x, u, \_, T) \succ g(x', u, \_, T) \leftarrow true$$

In order to prove that every preference statement expressed in this form is a proper CP-Nets/CP-Theories expression we need to show that the semantics of such statements are equivalent to their definition in CP-Nets/CP-Theories when comparing two outcomes. Using the previous example, given the statement of $g(x, u, \_, T) \succ g(x', u, \_, T) \leftarrow true$ and two (partially) grounded terms $g(t_1, ..., t_4)$ and $g(t'_1, ..., t'_4)$, we can infer $g(t_1, ..., t_4) \succ g(t'_1, ..., t'_4)$ iff we have $t_1 = x$, $t'_1 = x'$ and $t_2 = t'_2 = u$ and $t_4 = t'_4$ regardless of the values of t_3 and t'_3. Intuitively, this is equivalent to the definitions of CP-Nets/CP-Theories. By using induction we can see that the same can be inferred for any number or parameters that correspond to G_U, G_W, G_T or with any other rearrangement of the parameters.

3.3.2 Embedding in BDI agents. Pure CP-Theory (and by extension CP-Net) statements can be expressed in the form of $G \succ G' \leftarrow C$ where $C = true$, but, as BDI agents are designed to act in dynamic environments, simply using static CP-Theory statements is not sufficient. This is addressed by the *activation* condition C of preference statements. This condition can be any arbitrary Prolog-like expression over the belief base of the agent. A preference statement is *active* if the context condition can be proven from the belief base of the agent. Intuitively this means that the left side of the rule holds true if the right side can be proven. This can drastically increase the expressivity of the preference statements in dynamic environments.

Definition 4 (Active Preference Statement). *At any point in the life-cycle of agent with a belief base B and set of preference statements Λ, a preference statement $G \succ G' \leftarrow C$ is active iff C is a logical consequence of B.*

The next example further explores this type of preferences, and is an extended version of what is presented in the original CP-net paper [4] to facilitate comparison.

Example 3. Let us consider some preferences over the actions of our food ordering agent, starting from some preferences over the meal: (R1) for the main course, meat is preferred to fish if at an Italian restaurant; (R2) fish is preferred to meat if at a French restaurant; (R3) if the main course is meat, then a fish soup is preferred to vegetable soup; (R4) if the main course is fish, then a vegetable soup is preferred; (R5) for drinks, red wine is preferred to *any* other type of drink if vegetable soup is in the meal; likewise, (R6) white wine is preferred to *any* other drinks if fish soup is in the meal.

While multiple preferences are defined over the meal, still they do not translate directly to any of the events that the agent can handle. To fix this, we add a simple but important (meta-)preference: (R7) in the event of ordering *anything*, ordering something more preferred is also preferred to ordering something less preferred. We define at this point a few preferences about the restaurant: (R8) if the agent is already located at a restaurant, it is preferred to order at that restaurant (i.e. not to move) compared to any other restaurant, regardless of the meal; (R9) the combination of Italian restaurant with a meat main dish is preferred to any other restaurant and main dish combination if the agent is not already at another restaurant. The specification of these preferences can be seen in Listing 3. A more detailed and practical explanation of how these statements are written as Prolog rules can be found in Section 4.

We can draw as in Figure 2 the preferential relation graph associated to the predicate `meal/3` with respect to preferences in Listing 3

```
(R1) meal(S,meat,W) >> meal(S,fish,W) :- at(italian).
(R2) meal(S,fish,W) >> meal(S,meat,W) :- at(french).
(R3) meal(fish,meat,W) >> meal(veg,meat,W) :- true.
(R4) meal(veg,fish,W) >> meal(fish,fish,W) :- true.
(R5) meal(veg,M,red) >> meal(veg,M,_) :- true.
(R6) meal(fish,M,white) >> meal(fish,M,_) :- true.
(R7) !order(M1) >> !order(M2) :- M1 >> M2.
(R8) !go_order(L,_) >> !go_order(_,_) :- at(L).
(R9) !go_order(italian, meal(S,meat,W))
     >> !go_order(L,meal(S,_,W)) :-  not at(L).
```

Listing 3: Preferences of Food-ordering agent

and the beliefs in Listing 2. The graph clearly suggests that, in this context, `meal(veg,fish,red)` is the most preferred (undominated) option. Note however that this graph would have been different if the agent had the belief `at(italian)` instead of `at(french)`.

3.3.3 Generalization. The preference statements in Listing 3 have been chosen because they offer a good representation of the possible uses of the proposed method, and can be easily generalized. First of all, we observe that only R3 and R4 are pure CP-Theory preferences. The statements R1, R2 are conditioned on beliefs that are external with respect to the variables of the preference itself (in this case, the location of the agent). This type of conditional statements result in multiple preferential relations that may also be contradictory. For instance, if both conditions `at(french)` and `at(italian)` were true at any time, then the two preferences would be contradictory and the preference relations concerning `meal/3` would be unsatisfiable.

The statements R5 and R6 specify preferences that, although simple, cannot be expressed directly in standard CP-Theories, as they give conditional preference to a value of a variable (drinks) over any other value of that variable (*erga omnes* preference); this can be very useful in cases where the domain of values of a variable are unknown at design time, but the designer is aware of a few values that are always either desired or to be avoided.

As it was already observed, R7 is a meta-preference. The condition of this statement does not consult the belief-base of the agent, but rather the preferences of the agent. It works as a mapping of a preference expressed as object level to a preference expressed as action level. In general, any preference over objects is implicitly referring to a certain domain of activities, and it is as such only a more compact representation. Preferences as R7 are needed to make explicit this connection. The technical aspects of this step will be explained more in detail in Section 4.

The statement R8 is conditional over a belief that the agent may have (`at(L)`), and also connects the variable of this condition `L` (that will be grounded at run-time) to the preference relation itself, creating an interesting *parameterized* preference structure capturing in this case "*I prefer the place I am already at*".

Finally, also R9 presents a statement that cannot be expressed in CP-Theories, in which two different variables are part of the preference. This can be useful but can also easily lead to cyclic preferences, so this type of statement should be used with care.

3.4 Goal Refinement via Preferences

The integration of preferences into the BDI reasoning cycle can be implemented as a *unifier ordering* step prior the *plan selection*, that only applies when the triggering event contains unbounded

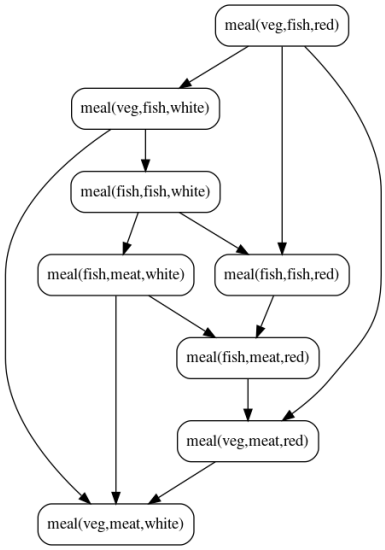

Figure 2: Preference Structure of the Agent over `meal/3`

variables. To achieve this, when the agent selects a partially unbounded event ϵ, first it needs to find all the relevant unifiers by consulting the plan library, and then find all the applicable unifiers by consulting the belief base. Afterwards, the agent can create a partial ordering between the applicable unifiers.

To extend the definition of Section 3.3 to unifiers, assuming ϵ is a partially unbound event and $(\sigma\delta), (\sigma\delta)'$ are two applicable unifiers for ϵ, we say $(\sigma\delta) \succ (\sigma\delta)'$ satisfies a preference statement $G \succ G' \leftarrow C$ if $\epsilon(\sigma\delta)$ can be unified with G and $\epsilon(\sigma\delta)'$ can be unified with G'. Given Λ, a set of acyclic preference statements in this form, we say Λ is satisfied by a preference ordering $\succ$ iff $\succ$ satisfies each of the active (i.e. C is entailed by the belief base) conditional preferences expressed in Λ. Then, $\Lambda \models (\sigma\delta) \succ (\sigma\delta)'$ iff $(\sigma\delta) \succ (\sigma\delta)'$ holds in every preference ordering that satisfies Λ. We can then say $(\sigma\delta)$ is consistently orderable or weakly preferred to $(\sigma\delta)'$ iff $\Lambda \not\models (\sigma\delta)' \succ (\sigma\delta)$. We can now define the *undominated* or *most preferred* unifier:

Definition 5 (Most preferred unifier). *A unifier $(\sigma\delta)$ is referred to as an undominated or most preferred unifier for an event ϵ iff $\sigma\delta$ is an applicable unifier for ϵ and for every other applicable unifier $(\sigma\delta)'$ we have $\Lambda \not\models (\sigma\delta)' \succ (\sigma\delta)$.*

Surprisingly, with this definition, finding the most preferred unifier is simple in a Prolog program as it matches well with how Prolog engines work. Normally in a Prolog program it is not easy to query if a fact holds with respect to every rule concerning that fact, but if a query about a fact fails, this means that all the rules about that fact have failed, i.e. that the fact cannot be proven (by refutation, a contradiction cannot be found). Thus, to find if a unifer $(\sigma\delta)$ is the most preferred one for an event ϵ, it is enough to ask if, for every other unifier $(\sigma\delta)'$, the query $(\sigma\delta)' \succ (\sigma\delta)$ fails. Intuitively, running this query for every unifier of ϵ will result in finding the most preferred unifier. More details on the implementation of this algorithm is presented in Section 4.

For simplicity, it is assumed that the agent uses the plan selection function typical in BDI frameworks, i.e. selecting the first applicable unifier for each event. This assumption means that the agent always uses an *undominated* or *most preferred* unifier to ground the variables of a (partially) abstract goal if this unifier exists, or reverts to the default behavior of selecting the first applicable unifier in case of inconsistencies with preferences that may result in situations that no unifier is the most preferred.

Example 4. Consider again the agent script given in Example 1, with the beliefs of Example 2, and with the preferences of Example 3. Assume that this agent receives an abstract event `!go_order(L,M)`. Then, as in Example 2, two applicable unifiers will be created. P1 will be applicable with the unifier `{Loc/italian, Meal/M}` and P2 with the unifier `{Loc/french, Meal/M}`. Because the agent has the belief `at(french)`, we can see that the relation:

```
!go_order(italian,M) >> !go_order(french,M)
```

cannot be proven from the preferences (R8 and R9 cannot conclude it) but the relation:

```
!go_order(french,M) >> !go_order(italian,M)
```

can be proven to be true (based on R8), so `{Loc/french, Meal/M}` is the single most preferred unifier, and P2 will be selected as this is the only plan applicable with this unifier. Next, when the sub-goal `!order(Meal)` is being considered, there are 8 applicable unifiers for plan P3, assuming the `at(french)` belief still stands, and based on the given preference rules, the ordering in Fig. 2 will apply, and then the most preferred unifier will be `{S/veg, M/fish, W/red}`. This means that the abstract goal will be refined to `!order(meal(veg,fish,red))` and consequently plan selection will instantiate the plan associated to it (P3).

Example 5. To show how partially abstract goals would be grounded with this method, consider the same agent as before, with the same set of preferences and beliefs, except that this time the agent has the belief `at(home)` instead of `at(french)`, and the agent receives an event with ordering a meal with meat as the main course:

```
!go_order(L,meal(S,meat,W)).
```

This time plan P2 will be applicable with two unifiers:

```
{Loc/italian, Meal/meal(S,meat,W)}
{Loc/french, Meal/meal(S,meat,W)}
```

because the agent has the belief `at(home)`, the statement R8 is not active for any of the unifiers and based on the the statement R9, the Italian restaurant is preferred to any other restaurant as long as there is meat main course, so again while the relation:

```
!go_order(italian,meal(S,meat,W)) >> !go_order(french,meal(S,meat,W))
```

can be proven (based on R9), but the relation:

```
!go_order(french,meal(S,meat,W)) >> !go_order(italian,meal(S,meat,W))
```

cannot be proven from the preference statements (R8 and R9 cannot conclude it), so the first unifier will be the most preferred one and then P2 is selected with it. Next, assuming the `move_to` action works correctly, the belief `at(home)` will be retracted, `at(italian)` will be added to the belief-base, and the sub-goal `!order(meal(S,meat,W))` will be adopted. At this point (see the example in [4]), based on the preference statements and agent's beliefs, the most preferred unifier will be `{S/fish, M/meat, W/white}`, meaning the goal will

be refined to `!order(meal(fish,meat,white))`, and then the plan for this goal (P3) will be instantiated by the plan selection.

An interesting point in this example is the interaction between preference statements R1 and R9. In normal CP-Nets, these two statements make the network cyclic: the preference over main dish is dependent on the location (R1), and the preference over the location depends on the main dish (R9). But in this work such preferences can be defined for two reasons: (1) *framing*: the two preferences, although being about the same variables, are defined in two different frames of choice; and (2) *context*: as the agent resides and acts in a dynamic environment, it perceives changes and modifies its beliefs, which in turn modifies the agent's preferences, e.g. while the agent has the belief `at(home)`, R1 and R2 are not active, and so they are not part of unifier selection process.

4 IMPLEMENTATION

This section presents a practical implementation of the transformation method from CP-Theory logic to Prolog logic proposed in Section 3.3.1.

Preference operator. First, we need to express a preference statement $G1 \succ G2 \leftarrow C$, where $G1, G2$ are partially grounded terms with the same functor and arity. The mapping from CP-Theory statements to this statement is presented in Section 3.3.1. This binary operator $\succ$ will be introduced both into the syntax of the agent programming language and as a binary predicate into the belief base of the agent. In the syntax, the operator $\succ$ will be denoted as `>>` making a relation such as $G1 \succ G2$ to be written as `G1 >> G2`. To write full contextually conditioned statements as $G1 \succ G2 \leftarrow C$, we can utilize the Prolog inference rules. The former preference statement can then be written as `G1 >> G2 :- C`.

Applicability. Next, as the method is implemented by utilizing the belief-base of the agent, a modification is needed to allow preference statements about different types of *goals* to be part of the belief base, as e.g. in the R8 statement from Listing 3. To do this, a supplementary predicate $applicable/2$ is introduced. When a preference statement $G1 \succ G2 \leftarrow C$ is defined where $G1$ and $G2$ are triggering events e.g. achievement goal ($!G$), test goal ($?G$), the preference statement is transformed at compile/interpretation time to:

$$applicable(t, G1) \succ applicable(t, G2) \leftarrow C$$

where t is a predefined atom describing the type of the event, e.g: the preference statement R8 in Listing 3 will be rewitten as:

```
applicable(achievement,go_order(L,_))
    >> applicable(achievement,go_order(_,_))
    :- at(L).
```

As this is a normal Prolog rule, it can be simply added to the belief base of the agent. Then, preference relations can be queried from any context, as e.g. the (meta-)preference R7 in Listing 3, in which a preference relation is used as the condition of another preference, or, more importantly, to exploit Prolog queries to find the most preferred unifier for abstract events.

Optimality. Next, the algorithm should be implemented for finding an optimal outcome, that is, the most preferred unifier(s) for a partially abstract term. The algorithms originally introduced for CP-Nets and CP-Theories generate an optimal outcome by *sweeping* through the network from top to bottom (i.e., from ancestors to descendants) setting each variable to its most preferred value given the instantiation of its parents. While these algorithms are efficient and intuitive, they are not applicable for the transformed Prolog-like rules. Unlike CP-Nets and CP-Theories, the preferential structure of an agent is not static because of the presence of extra-contextual conditions; also, with the new form of statements, the parent-child relation of the variables is not explicit anymore. For these reasons, new algorithms are needed that do not rely on the hierarchy of the variables but instead utilize the backtracking feature of Prolog.

Referring at the definition 5, given a set of preference statements as Prolog rules in a belief base B, to prove that a unifier $(\sigma\delta)$ is the most preferred unifier for a partially unbound term (or event) ϵ, it is sufficient to prove that for every other unifier $(\sigma\delta)'$ of that term, the relation $\epsilon(\sigma\delta)' \succ \epsilon(\sigma\delta)$ is not a logical consequence of B. Intuitively, with the semantics of Prolog, this means that this relation could not be concluded from any of the preference statements of B. With this, a simple algorithm that can find the most preferred (or undominated) unifier(s) for a partially unbound term is a backtracking search that goes through every possible (partial) grounding of that term to find one where there is no other (partial) grounding of that term which is more preferred to it. Such algorithm can be implemented by adding this Prolog rule to the agent's belief base:

```
most_prefered(G) :-
    copy_term(G,G2), G, forall(G2,((G2 >> G)-> fail; true)).
```

The `copy_term/2` is a standard predicate in many Prolog implementations that unifies `G2` with a copy of `G` in which all variables are replaced by new variables. When this rule is queried with a partially unbound term `G`, first a copy of `G` is created as `G2`, then the term `G` itself is called, starting a backtracking search over all possible groundings of `G`, and then `forall/2` predicate starts a nested loop over all groundings of `G2` and fails if it can find a grounding of `G2` that `(G2 >> G)`, otherwise if no such `G2` is found it returns true meaning the current grounding of `G` is the most preferred one. For the incremental derivation of Prolog, if there is more than one most preferred (undominated) grounding, asking for more answers will return them. Finally, to make this algorithm work we need to add: `T >> T :- !,fail.` specifying that a term cannot be preferred to itself. With these rules added to the belief base, given a partially unbound term, we can run queries to find the most preferred unifier for that term. If we consider the example agent, querying the belief base with the term `most_preferred(meal(S,M,W))` will give the result in one answer with the unifier `{S/veg, M/fish, W/red}`.

Embedding in the decision-making cycle. Now that the belief base can answer queries about the most preferred unifier for a term, the next step is to allow the agent's reasoning engine to ask queries about the most preferred unifier for an event. To do this, the $applicable/2$ predicate that was defined before is used again. At compile/interpretation time, for each plan of the form $e : C \Rightarrow H$ a Prolog rule is added to the belief base of the agent in the form of:

$$applicable(t, G) \leftarrow C$$

where t is an atom that represents the type of the event e and G is the term associated with e. As an example, the rule for plan P1 is:

```
applicable(achievement,go_eat(L,M)) :-
    restaurant(L) & not at(L)
```

Intuitively, at any moment in run-time, by querying this predicate, we can retrieve all possible applicable groundings of an event that can be concluded from the plan library and the belief base. For instance, by querying the term `applicable(achievement,go_eat(L,M))` on the belief base of an agent with the beliefs, plans and preferences described in Section 3, the agent obtains two answers: {`L/italian, M/M`} and {`L/french, M/M`}. Then, by using this predicate with combination of `most_preferred/2`, the agent can find the *most preferred applicable unifier* for an event. This is possible because the preference statements about events were already transformed with the `applicable` predicate. Considering our running example, by querying the term:

```
most_preferred(applicable(achievement,go_eat(L,M)))
```

only one answer {`L/french, M/M`} will be returned. At this point, we need to embed the goal refinement step into the agent reasoning cycle. After the event selection step, if the selected event ϵ contains free variables, then the most preferred unifier(s) should be found for this event by querying the belief base of the agent with the aforementioned method, and the resulting answer(s) are sent to the plan selection function.

Complexity. The algorithmic complexity of this approach is comparable to the original algorithms of CP-Nets. Considering a CP-Net with n number of preference statements, the complexity of comparing two outcomes is $o(n)$ [4, Theorem 5], then, if there are m outcomes the complexity of finding the ordering between all outcomes is $o(n \times m^2)$ [4, Theorem 6]. In this work, to compare two outcomes (groundings), every preference statement should be tested which give the same complexity of $o(n)$, also, the core of the method is the predicate `most_preferred/1`, and this rule has two nested backtracking loops over the possible groundings of the input term. In the worst case scenario, each two groundings of a term should be queried with all the preference statements associated to that rule. Then, for a term T, if the there are m possible groundings at any time, and there are n preference statements over T, then, in the worst case scenario, the time complexity of finding the most preferred unifier will be $o(n \times m^2)$ which is the same polynomial complexity of CP-Nets.

5 DISCUSSION AND CONCLUSION

This paper contributes to recent efforts to integrate preferences into BDI agents. Despite the 'D' in the acronym, desires play a limited role in contemporary BDI agent platforms, as they are generally conflated to goals (procedural or declarative). This paper showed that by interacting adequately with the belief base and plan library of the agent, abstract goals can be refined, taking into considerations the agent's preferences. Stated differently, preferences act here as background desires modifying/impacting goals, playing the role in turn of contingent desires. (Note that in general the literature suggests that preferences are derived from desires [18]; for our purposes, however, we discovered that the two can be seen as filling the same functional niche.)

Although this work illustrated the use of preferences focusing on a single agent and on goal refinement, preference statements can be relevant in other contexts too. For instance, MAS frameworks allow agents to communicate and transmit their beliefs to each other. Leveraging the present proposal, because preference statements are implemented as beliefs, agents can directly communicate their preferences to other agents. This can be very useful e.g. in social simulation or social learning contexts, where the agents may need to decide to act (or not to act) depending on both their own and other agents' preferences. Another interesting use-case for this approach could be the implementation of normative agents, utilizing preference both to capture personal and societal norms (see the use of CP-nets for deontic logic in [17]).

Preference statements introduced in this work are in a form processable in Prolog logic programs. We have shown that a subset of this form can be used to express pure CP-Theory preferences. However, this new form can also be used to express contextually conditioned and parameterized preferences, resulting in much more flexibility than pure CP-Theories. Consider for instance the statement R8 in Example 3: "*I prefer the place I am already at*"; that depends completely on the state of the agent in the environment and, if the environment is unknown and unpredictable, so will be the preference statement. Also, unlike CP-Theories that are fully qualitative, with the proposed form quantitative preferences can be expressed by using arbitrary arithmetic equations in the context condition of preferences; e.g: consider a statement "*I prefer a cheaper restaurant to a more expensive one*" that can easily be expressed in this form with an arithmetic comparison as the condition of the statement. Finally, one of the main requirement behind this work is accessible usability. The transformation method from CP-Theory preference statements to Prolog-like programs has been conceived to enable its use almost directly with AgentSpeak(L)-like frameworks [1, 3, 11, 20]. Indeed, no extra reasoning component is introduced, all the preference reasoning and algorithms required for goal refinement is done through beliefs and inferential rules. Furthermore, unlike many of the works that embed preferences into BDI frameworks, e.g. [8, 19, 22, 26, 27], this approach does not require any extra annotation of the agent's script with information about effects of plans or actions and thus makes it more accessible for the designer.

A concrete Prolog implementation of ordering queries of CP-Nets/CP-Theories was presented, and a working proof of concept for this approach is publicly available[2]. Analyzing its complexity, we showed that the proposed algorithm run in polynomial time. Such worst-case scenario could be however reduced by optimizing the relative positions of preferential rules and groundings in the belief base, for instance by exploiting statistical information concerning their applicability or relevance for the decision-making cycle.

ACKNOWLEDGMENTS

The work as presented in this paper has been done as part of the Dutch Research project *Data Logistics for Logistics Data* (DL4LD), supported by the Dutch Organisation for Scientific Research (NWO), the Dutch Institute for Advanced Logistics TKI Dinalog (http://www.dinalog.nl/) and the Dutch Commit-to-Data initiative (http://www.dutchdigitaldelta.nl/big-data/over-commit2data) (grant no: 628.009.001).

[2] https://github.com/uva-cci/aamas2022-preferences-poc

REFERENCES

[1] Malte Aschermann, Sophie Dennisen, Philipp Kraus, and Jörg P. Müller. 2018. LightJason, a Highly Scalable and Concurrent Agent Framework: Overview and Application. In *Proceedings of the 17th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS2018)*. 1794–1796.

[2] Meghyn Bienvenu, Christian Fritz, and Sheila A. McIlraith. 2006. Planning with Qualitative Temporal Preferences. In *Proceedings of the 10th International Conference on the Principles of Knowledge Representation and Reasoning (KR2006)*. 134–144.

[3] Rafael H. Bordini, Jomi F. Hübner, and Renata Vieira. 2005. Jason and the Golden Fleece of Agent-Oriented Programming. In *Multi-Agent Programming: Languages, Platforms and Applications*. 3–37.

[4] Craig Boutilier, Ronen I Brafman, Carmel Domshlak, Holger H Hoos, and David Poole. 2004. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of artificial intelligence research* 21 (2004), 135–191.

[5] Michael E. Bratman. 1987. *Intention, Plans, and Practical Reason*. Vol. 10. Harvard University Press.

[6] G Brewka. 2004. A Rank Based Description Language for Qualitative Preferences. *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI'04)* May (2004), 303–307.

[7] Roberta Calegari, Giovanni Ciatto, Viviana Mascardi, and Andrea Omicini. 2021. Logic-Based Technologies for Multi-Agent Systems: Summary of a Systematic Literature Review. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems* (Virtual Event, United Kingdom) *(AAMAS '21)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 1721–1723.

[8] Aniruddha Dasgupta and Aditya K. Ghose. 2010. Implementing reactive BDI agents with user-given constraints and objectives. *International Journal of Agent-Oriented Software Engineering* 4, 2 (2010), 141.

[9] Mehdi Dastani. 2008. 2APL: A practical agent programming language. *Autonomous Agents and Multi-Agent Systems* 16, 3 (2008), 214–248.

[10] Lavindra de Silva and Lin Padgham. 2004. A Comparison of BDI Based Real-Time Reasoning and HTN Based Planning. November (2004), 1167–1173. https://doi.org/10.1007/978-3-540-30549-1

[11] Akshat Dhaon and Rem W. Collier. 2014. Multiple Inheritance in AgentSpeak(L)-Style Programming Languages. In *Proceedings of the 4th International Workshop on Programming Based on Actors Agents & Decentralized Control*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/2687357.2687362

[12] Jürgen Dix and Yingqian Zhang. 2005. *Impact: A Multi-Agent Framework with Declarative Semantics*. Vol. 15. 69–94. https://doi.org/10.1007/0-387-26350-0_3

[13] Carmel Domshlak, Eyke Hüllermeier, Souhila Kaci, and Henri Prade. 2011. Preferences in AI: An overview. *Artificial Intelligence* 175, 7-8 (2011), 1037–1052.

[14] C. Gonzales and P. Perny. 2004. GAI Networks for Utility Elicitation. In *Proceedings of the 9th International Conference on the Principles of Knowledge Representation and Reasoning (KR2004)*. 224–233.

[15] Koen V. Hindriks. 2009. Programming Rational Agents in GOAL. In *Multi-agent programming: Languages, platforms and applications*. Chapter 4, 119–157.

[16] Nick Howden, Ralph Rönnquist, Andrew Hodgson, and Andrew Lucas. 2001. JACK Intelligent Agents - Summary of an Agent Infrastructure. (01 2001).

[17] Andrea Loreggia, Emiliano Lorini, and Giovanni Sartor. 2020. A Ceteris Paribus Deontic Logic. In *Proceedings of the 35th Italian Conference on Computational Logic - CILC 2020, Rende, Italy, October 13-15, 2020 (CEUR Workshop Proceedings, Vol. 2710)*, Francesco Calimeri, Simona Perri, and Ester Zumpano (Eds.). CEUR-WS.org, 248–262. http://ceur-ws.org/Vol-2710/paper16.pdf

[18] Emiliano Lorini. 2017. Logics for games, emotions and institutions. *The IfCoLog Journal of Logics and their Applications* 4, 9 (2017), 3075–3113.

[19] Mostafa Mohajeri Parizi, Giovanni Sileno, and Tom van Engers. 2019. Integrating CP-Nets in Reactive BDI Agents. In *PRIMA 2019: Principles and Practice of Multi-Agent Systems*. Springer International Publishing, 305–320.

[20] Mostafa Mohajeri Parizi, Giovanni Sileno, Tom van Engers, and Sander Klous. 2020. Run, Agent, Run! Architecture and Benchmark of Actor-based Agents. proceedings of Programming based on Actors, Agents, and Decentralized Control (AGERE20), ACM.

[21] Lin Padgham and Dhirendra Singh. 2013. Situational preferences for BDI plans. *12th International Conference on Autonomous Agents and Multiagent Systems* (2013), 1013–1020. http://dl.acm.org/citation.cfm?id=2485080

[22] Mostafa Mohajeri Parizi, Giovanni Sileno, and Tom van Engers. 2021. Declarative Preferences in Reactive BDI Agents. In *PRIMA 2020: Principles and Practice of Multi-Agent Systems*, Takahiro Uchiya, Quan Bai, and Iván Marsá Maestre (Eds.). Springer International Publishing, Cham, 215–230.

[23] Gabriella Pigozzi, Alexis Tsoukiàs, and Paolo Viappiani. 2016. Preferences in artificial intelligence. *Annals of Mathematics and Artificial Intelligence* 77, 3-4 (2016), 361–401.

[24] Anand S. Rao. 1996. AgentSpeak(L): BDI agents speak out in a logical computable language. In *Agents Breaking Away*. 42–55.

[25] Anand S. Rao and Michael P. Georgeff. 1995. BDI agents: From theory to practice.. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS1995)*. 312–319.

[26] Simeon Visser, John Thangarajah, and James Harland. 2011. Reasoning about preferences in intelligent agent systems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI2011)*. 426–431.

[27] Simeon Visser, John Thangarajah, James Harland, and Frank Dignum. 2016. Preference-based reasoning in BDI agent systems. *Autonomous Agents and Multi-Agent Systems* 30, 2 (2016), 291–330.

[28] Wietske Visser, Reyhan Aydoğan, Koen V. Hindriks, and Catholijn M. Jonker. 2012. A framework for qualitative multi-criteria preferences. *ICAART 2012 - Proceedings of the 4th International Conference on Agents and Artificial Intelligence* 1 (2012), 243–248. https://doi.org/10.5220/0003718302430248

[29] Nic Wilson. 2004. Extending CP-Nets with Stronger Conditional Preference Statements. In *Proc. 19th National Conf. on Artificial Intell. (AAAI'04)*, Vol. 4. 735–741.