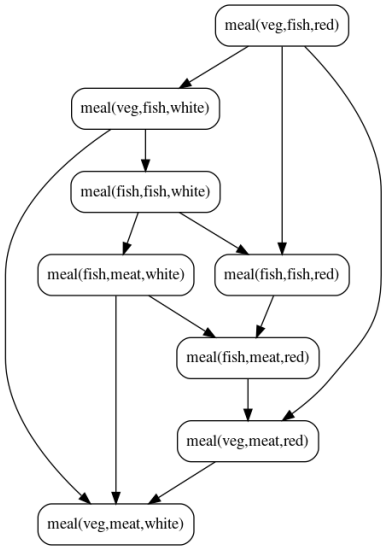

Figure 2: Preference Structure of the Agent over `meal/3`

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.