People have developed a number of different languages for encoding preferences over a set of alternatives (e.g., cardinal, ordinal, and combinatorial domains). Each of these domains may have different properties. For instance, a cardinal domain may not be able to express all preferences in an ordinal domain. Similarly, an ordinal domain may not be able to express all preferences in a combinatorial domain. Therefore, a language may be a good way to express a preference domain over a set of alternatives.

## Combinatorial Domains

In the context of combinatorial domains, it is often desirable to express preferences over bundles of goods. For example, a voter may prefer one set of goods to another, even if the individual goods in the two sets are identical. To express these preferences, we may use a language that allows us to express preferences over bundles of goods.

## Explicit Representation

Preferences in a combinatorial domain can be expressed explicitly as a set of weighted propositional formulas. By convention, table entries with entries \( \subseteq G \) may be omitted.

### Weighted Propositional Formulas

- \( G \) is a finite set of propositional letters.
- \( N \) is a set of numerical weights.
- \( X \) is a finite set of alternatives.
- \( X \rightarrow \mathbb{R} \) is a cardinal domain, and \( \mathbb{R} \) is a domain of numerical weights.

For all models \( M \), \( (\neg A \land B) \cup C \) should be the same as \( A \land (B \lor C) \). That is, here each \( \mathbb{R} \) is a binary relation.

## Properties of Preference Representation Languages

- Expressivity
- Succinctness
- Complexity

## Reference Representation Languages

- \( \mathcal{L}_1 \) is a set of explicit representations.
- \( \mathcal{L}_2 \) is a set of succinct representations.
- \( \mathcal{L}_3 \) is a set of expressive representations.

### Explicit Form

- \( \mathcal{L}_4 \) is a set of combinatorial domains.
- \( \mathcal{L}_5 \) is a set of propositional languages.

## Complexity

- \( \mathcal{L}_6 \) is a set of computational complexity classes.
- \( \mathcal{L}_7 \) is a set of polynomial time complexity classes.

## Acknowledgments

This work was supported by the European Commission under the 7th Framework Programme (FP7/2007-2013) under Grant Agreement 247875 (PASTA).


Positive cubes are unique utility functions. For any choice of \( \alpha \), we have seen that positive cubes are fully expressive. Hence, we can express all \( \alpha \)-related utility functions, except \( \top \). Clearly, \( \top \)-related utility functions, can express all \( \alpha \)-related utility functions, except \( \top \). We have seen that positive cubes are fully expressive. Hence, we can express all \( \alpha \)-related utility functions, except \( \top \). Clearly, \( \top \)-related utility functions, can express all \( \alpha \)-related utility functions, except \( \top \). We have seen that positive cubes are fully expressive. Hence, we can express all \( \alpha \)-related utility functions, except \( \top \). Clearly, \( \top \)-related utility functions, can express all \( \alpha \)-related utility functions, except \( \top \). We have seen that positive cubes are fully expressive. Hence, we can express all \( \alpha \)-related utility functions, except \( \top \). Clearly, \( \top \)-related utility functions, can express all \( \alpha \)-related utility functions, except \( \top \). We have seen that positive cubes are fully expressive. Hence, we can express all \( \alpha \)-related utility functions, except \( \top \). Clearly, \( \top \)-related utility functions, can express all \( \alpha \)-related utility functions, except \( \top \). We have seen that positive cubes are fully expressive. Hence, we can express all \( \alpha \)-related utility functions, except \( \top \). Clearly, \( \top \)-related utility functions, can express all \( \alpha \)-related utility functions, except \( \top \). We have seen that positive cubes are fully expressive. Hence, we can express all \( \alpha \)-related utility functions, except \( \top \). Clearly, \( \top \)-related utility functions, can express all \( \alpha \)-related utility functions, except \( \top \). We have seen that positive cubes are fully expressive. Hence, we can express all \( \alpha \)-related utility functions, except \( \top \). Clearly, \( \top \)-related utility functions, can express all \( \alpha \)-related utility functions, except \( \top \). We have seen that positive cubes are fully expressive. Hence, we can express all \( \alpha \)-related utility functions, except \( \top \). Clearly, \( \top \)-related utility functions, can express all \( \alpha \)-related utility functions, except \( \top \). We have seen that positive cubes are fully expressive. Hence, we can express all \( \alpha \)-related utility functions, except \( \top \). Clearly, \( \top \)-related utility functions, can express all \( \alpha \)-related utility functions, except \( \top \). We have seen that positive cubes are fully expressive. Hence, we can express all \( \alpha \)-related utility functions, except \( \top \). Clearly, \( \top \)-related utility functions, can express all \( \alpha \)-related utility functions, except \( \top \). We have seen that positive cubes are fully expressive. Hence, we can express all \( \alpha \)-related utility functions, except \( \top \). Clearly, \( \top \)-related utility functions, can express all \( \alpha \)-related utility functions, except \( \top \). We have seen that positive cubes are fully expressive. Hence, we can express all \( \alpha \)-related utility functions, except \( \top \). Clearly, \( \top \)-related utility functions, can express all \( \alpha \)-related utility functions, except \( \top \). We have seen that positive cubes are fully expressive. Hence, we can express all \( \alpha \)-related utility functions, except \( \top \). Clearly, \( \top \)-related utility functions, can express all \( \alpha \)-related utility functions, except \( \top \). We have seen that positive cubes are fully expressive. Hence, we can express all \( \alpha \)-related utility functions, except \( \top \). Clearly, \( \top \)-related utility functions, can express all \( \alpha \)-related utility functions, except \( \top \). We have seen that positive cubes are fully expressive. Hence, we can express all \( \alpha \)-related utility functions, except \( \top \). Clearly, \( \top \)-related utility functions, can express all \( \alpha \)-related utility functions, except \( \top \). We have seen that positive cubes are fully expressive. Hence, we can express all \( \alpha \)-related utility functions, except \( \top \). Clearly, \( \top \)-related utility functions, can express all \( \alpha \)-related utility functions, except \( \top \). We have seen that positive cubes are fully expressive. Hence, we can express all \( \alpha \)-related utility functions, except \( \top \). Clearly, \( \top \)-related utility functions, can express all \( \alpha \)-related utility functions, except \( \top \). We have seen that positive cubes are fully expressive. Hence, we can express all \( \alpha \)-related utility functions, except \( \top \). Clearly, \( \top \)-related utility functions, can express all \( \alpha \)-related utility functions, except \( \top \). We have seen that positive cubes are fully expressive. Hence, we can express all \( \alpha \)-related utility functions, except \( \top \). Clearly, \( \top \)-related utility functions, can express all \( \alpha \)-related utility functions, except \( \top \).
The paradox of multiple elections can be illustrated using an example with binary issues. Suppose there are 10 binary issues to be decided upon by 13 voters. Each voter is asked to vote for three issues, resulting in a total of 39 votes. The problem is to find a winning combination of issues that is acceptable to the majority of voters.

Possible Solutions

Solution (1): Explicit Vote for Combinations
In this solution, voters are explicitly asked to vote for combinations of issues. This is a straightforward approach but can lead to a large number of possible combinations, especially in domains with many variables.

Solution (2): Vote for Top Combinations only
Voters are asked to vote for the top three combinations among the possible ones. This reduces the complexity of the voting process but may result in a lack of representation for smaller combinations.

Solution (3): Vote for Selected Combinations only
Voters are asked to vote for a small number of combinations selected by a predefined rule. This can help reduce the complexity of the voting process but may not capture the preferences of all voters.

Solution (4): Combinatorial Vote
This approach involves voting on combinations using a compact representation language and applying a voting rule to the succinctly encoded ballots received. It addresses the communication problems of solutions (1) and (2).

Problem: The problem of voting in combinatorial domains involves choosing a winner from a large set of possible combinations. This is particularly challenging when the set of combinations is large or the set of voters is small. One goal is to find a winning combination that is acceptable to the majority of voters.

Discussion: The goal is to find a winner that is acceptable to the majority of voters. This can be achieved through a variety of methods, such as finding a novel way of aggregating the votes or using a compact representation language to reduce the complexity of the voting process. However, finding algorithms to compute the winners is also complex and may require additional constraints to ensure fair and representative outcomes.
and some voting rule to give voters (Boutilier et al., 2004), which relaxes the:
and the generalised CP-nets rule. Single goals Vote separately on each issue, but do so (preferences over)
Vote separately on each issue. However, don't just promote Condorcet loser positive formulas results in a winning combination that is a to any ballot.
Sequential voting Just think what happens during the election for the language is a candidate that loses against any other of the winning of
Idea: For example, Brams et al. (2007) suggest choosing the combination to the combinations the voters voted for (ballots).
Discussion: For a discussion of compact preference representation languages
for combinatorial domains and references to the literature, see:

✓ was still possible after the penultimate election. Lacy and Niou (2000) show that sequential voting can avoid this:
Example:
Solution (5b): Vote Separately and Aggregate
Solution (5a): Vote Separately Anyway

Instance:
Voter preferences in Among-Winners

L
f
(L, f)

\text{Among-Winners is used?}

Winner Verification under Plurality

Solution (50b): Vote Separately and Aggregate
Solution (50a): Vote Separately Anyway

where formulation under plurality