More than the sum of its parts: compact preference representation over combinatorial domains

Uckelman, J.D.

Citation for published version (APA):
Uckelman, J. D. (2009). More than the sum of its parts: compact preference representation over combinatorial domains Amsterdam: Institute for Logic, Language and Computation

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 3

Expressivity

3.1 Introduction

An important feature of any preference representation language is the range of preferences which can be represented in it. This available range is known as the *expressivity* of the language, and this chapter is devoted to determining the expressivity of numerous goalbase languages.

From the point of view of the theorist, we want to know how expressive goalbase languages are, both for their own sake and because these expressivity results are necessary for proving succinctness results in Chapter 4. As a user of goalbase languages, knowing the expressivity of goalbase languages can help us choose the language which is best suited for our application. Can our language of choice represent all functions belonging to a given class of utility functions which interests us? Not all languages are equally expressive and not all applications require full expressivity. Excess expressivity is often undesirable, because highly expressive languages tend to be computationally more demanding to reason about.

We are interested in *correspondence results* between languages and classes of functions. For instance, a very simple result which we present shows that the language we obtain by restricting formulas to *literals* and by using summation to aggregate weights can express all *modular* utility functions, and only those.

An interesting property closely related to expressivity is *uniqueness* of representation. A language has the uniqueness property with respect to a given class of utility functions if it has no more than one way of representing any function from that class. This is an interesting property because it suggests that the language in question is parsimonious in its expressivity. Syntactically rich languages often lack the uniqueness property. Non-uniqueness may be considered wasteful or useful, depending on the intended application—but from a theoretical point of view uniqueness is a property which greatly simplifies proofs (especially those in Chapter 4).
This chapter is divided into two main parts, the first covering the expressivity of languages using the sum aggregator, the second covering those using the max aggregator. Within each section, we present some useful goalbase transformations, followed by the expressivity results themselves. Most of the expressivity results are correspondences between goalbase languages and classes of utility functions. Our expressivity results for sum languages are summarized in Section 3.4.4 and Figure 3.1, and in Section 3.5.4 and Figure 3.2 for max languages.

3.2 Preliminaries

We note here some properties of utility functions to which we make frequent reference:

**Definition 3.2.1** (Properties of Utility Functions). Suppose that $u$ is a utility function. Then:

- $u$ is *normalized* iff $u(\emptyset) = 0$.
- $u$ is *nonnegative* iff $u(X) \geq 0$ for all $X$.
- $u$ is *monotone* iff $u(X) \geq u(Y)$ for all $X \supseteq Y$.
- $u$ is *submodular* iff $u(X \cup Y) \leq u(X) + u(Y) - u(X \cap Y)$ for all $X, Y \subseteq \mathcal{P}S$.
- $u$ is *supermodular* iff $u(X \cup Y) \geq u(X) + u(Y) - u(X \cap Y)$ for all $X, Y \subseteq \mathcal{P}S$.
- $u$ is *modular* iff $u$ is both sub- and supermodular.
- $u$ is *subadditive* iff $u(X \cup Y) \leq u(X) + u(Y)$ for all $X \cap Y = \emptyset, X, Y \subseteq \mathcal{P}S$.
- $u$ is *superadditive* iff $u(X \cup Y) \geq u(X) + u(Y)$ for all $X \cap Y = \emptyset, X, Y \subseteq \mathcal{P}S$.
- $u$ is *additive* iff $u$ is both sub- and superadditive.
- $u$ is a *unit-demand* valuation iff $u(X) = \max_{a \in X} u(\{a\})$ and $u$ is normalized.
- $u$ is a *simple unit-demand* valuation iff $u(X) = 1$ for all $X \neq \emptyset$ and $u$ is normalized.

For further examples of utility function properties, see [Nisan, 2006] and [Lehmann, Lehmann, and Nisan, 2006a]. Note that modularity with normalization is equivalent to additivity, and that (super-) submodularity together with normalization implies (super-) subadditivity.

Additionally we define the property $k$-additivity:
3.2. Preliminaries

Definition 3.2.2. Let \( \mathcal{PS}(k) \) be the set of all subsets of \( \mathcal{PS} \) with at most \( k \) elements. A utility function \( u \) is \( k \)-additive if there exists a mapping \( m: \mathcal{PS}(k) \to \mathbb{R} \) such that

\[
  u(X) = \sum_{Y \subseteq X, Y \in \mathcal{PS}(k)} m(Y)
\]

for each set \( X \subseteq \mathcal{PS} \).

The \( k \)-additive functions play an important role in fuzzy measure theory [Grabisch, 1997] as well as in combinatorial auctions [Conitzer, Sandholm, and Santi, 2005] and distributed multiagent resource allocation [Chevaleyre, Endriss, Estivie, and Maudet, 2008a]. For example, if the variables in \( \mathcal{PS} \) are used to model whether an agent owns certain resources, then \( k \)-additive utility functions naturally model situations where synergies among different resources are restricted to bundles of at most \( k \) elements.

The \( k \)-additivity property is a generalization of another property of utility functions, namely modularity. The 1-additive utility functions are exactly the modular ones: For a utility function to be modular, the value it assigns to each good must be independent of the bundle it appears in, and 1-additivity enforces just that. Modular utility functions are interesting precisely because they are very simple and have limited expressive power. As a result, they are frequently used in applications, e.g., in work on modeling negotiation between autonomous software agents [Rosenschein and Zlotkin, 1994].

Supermodularity (and its counterpart, submodularity) are widely used concepts in the economics literature [Moulin, 1988]. The traditional assumption in economics is that goods have decreasing marginal utility—I will be willing to pay less for the \((n+1)\)th ice cream cone than the \(n\)th—which is to say that buyers’ utility functions are submodular. Similarly, nonnegativity, normalization, and monotonicity are nearly universally assumed by economists. Unit-demand valuations are often encountered when dealing with items where only a single one is necessary to fulfil its purpose. (For example, the typical person has unit-demand valuations for items like stoves and mobile phones.)

Goalbases may sometimes contain formulas which fail to contribute value to the goalbase in any state. We call these formulas superfluous.

Definition 3.2.3 (Superfluity). A weighted formula \((\varphi, w) \in G\) is superfluous under aggregator \(F\) if \(G \equiv_F G \setminus \{(\varphi, w)\}\).

It is easy to see that for sum languages, the only superfluous formulas are those equivalent to \(\bot\) or having zero weight; for every other formula, there would be at least one state which would change in value if it were removed. For max languages, the picture is more complex: While contradictions are still superfluous, it is possible for satisfiable formulas with nonzero weight to be superfluous and for formulas with zero weight not to be. We take up the issue of superfluity for max languages in detail in Section 3.5.1.
Chapter 3. Expressivity

Some goalbase languages have a unique way of expressing a given utility function, while others allow for several alternative representations. Here we make this notion of uniqueness precise:

**Definition 3.2.4 (Unique Representations).** A utility function $u$ is represented in a language $\mathcal{L}$ if there exists a goalbase $G \in \mathcal{L}$ such that $u = u_G$. A utility function $u$ is uniquely represented (modulo formula equivalence) in a language $\mathcal{L}$ if, given a set of formulas $\Phi$ containing one representative formula for each formula equivalence class in $\mathcal{L}$ (except $\bot$), there is a unique goalbase $G$ such that $\text{For}(G) \subseteq \Phi$, $u_G = u$, and $G$ contains no superfluous formulas. A language $\mathcal{L}$ is said to have unique representations if every $u$ represented in $\mathcal{L}$ is uniquely represented.

Any language which has unique representations can be thought of as minimal in the sense that any further restriction on permissible weighted formulas will lead to a reduction in expressivity. Effectively this means that the set of representatives of formula equivalence classes forms a minimal basis for the vector space in which the goalbases live. We discuss uniqueness for sum languages in Section 3.4.2, and for max languages in Theorem 4.5.3.

3.3 Related Work

In broad terms, the expressivity of languages has occupied mathematicians, and logicians in particular, for quite some time. The question of whether various figures can be constructed using only a compass and an unmarked straightedge was a particular favorite of the ancient Greeks, and is fundamentally a question of language expressivity. In this vein, Gauss’ celebrated proof that a heptadecagon is constructable can be thought of as a positive expressivity result, while Wantzel’s proof that arbitrary angles cannot be trisected shows that additional expressivity (e.g., substituting a marked ruler for an unmarked straightedge) is needed in order to admit such constructions. Expressivity has been a perennial issue in logic from early in the 20th century to the present day. “What can I say in this language?” is a natural question for logicians, and its pursuit has been fruitful. We can point to results concerning the expressive power of quantifier logics (e.g., that there is no first-order formula corresponding to finiteness) and the substantial literature on correspondences between formulas in modal logic and properties of classes of Kripke frames (Sahlqvist’s Theorem being a particularly nice example of this [Chagrov and Zakharyaschev, 1997, Section 10.3]).

We now move closer to the topic at hand. The expressivity of some ordinal preference representation languages has been studied by Coste-Marquis et al. [2004, Theorem 1]. In particular, they examine the ability of eight logic-based languages to represent classes of preorders over finite sets of objects. All of the languages considered consist of sets of propositional formulas which are annotated with
additional information. This additional information may be a weight, a penalty, a priority, a distance, or a context, depending on the language. Boutilier et al. [2004] discuss CP-nets, a graph-based language for representing ordinal conditional preferences. Only consistent preference orders (and not even all of those) are representable using acyclic CP-nets, though more (including some intransitive ones) are representable if dependency cycles are allowed. Often CP-nets will produce only a preorder, with which many linear orders are compatible. For more about CP-nets, see Section 2.3.1.

Ieong and Shoham [2005, Proposition 1] prove that MC-nets are fully expressive, which is equivalent to one part of our Corollary 3.4.9. For a discussion of MC-nets, see Sections 2.3.9 and 4.3.

Nisan [2006] collects numerous results on the expressivity of the OR/XOR family of languages for representing utility functions. (These languages are discussed in detail in Section 6.3.1.) For example, the XOR language corresponds to the monotone utility functions, while the OR language represents all superadditive utility functions. Several succinctness results given there also entail that particular classes of utility functions are representable in certain OR/XOR languages.

Lehmann et al. [2006a] define a syntactic hierarchy of OR/XOR languages which they aim to characterize. The languages defined are OS (OR of singletons), XS (XOR of singletons), OXS (OR of XS), and XOS (XOR of OS); these are shown to have expressivity such that

\[
\text{OXS} \subset \text{GS} \subset \text{SM} \subset \text{XOS} \subset \text{CF},
\]

where GS is the class of gross substitutes valuations (valuations where the demand for an item does not decrease when the prices of other items increase), SM is the class of submodular valuations, and CF is the class of complement-free valuations \((u(X) + u(Y) \geq u(X \cup Y))\). Though they do not further characterize these languages, Lehmann et al. do show that there is a large gap between GS and SM by demonstrating that each \(m\)-item subclass of GS has measure zero in \((2^m - 1)\)-dimensional Euclidean space, while the corresponding \(m\)-item subclass of SM has positive measure—which implies that XOS is a much more expressive language than OXS.

### 3.4 Expressivity of Sum Languages

Throughout this section, when the aggregator function is omitted from the notation, it is indented to be \(\Sigma\).

#### 3.4.1 Goalbase Equivalences

Recall (from Definition 2.2.8) that two goalbases \(G\) and \(G'\) are equivalent \((G \equiv G')\) iff they generate the same utility function (i.e., \(u_G = u_{G'}\)). The following
Theorem 3.4.2. Given any utility function \( L \) which we now set out to prove.

Proof. Given any utility function \( L \) we must have unique representations, namely \( L \) has unique representations. Neither \( L(\text{cubes}, \mathbb{R}, \Sigma) \) nor \( L(\text{clauses}, \mathbb{R}, \Sigma) \), for instance, have them: The two goalbases \( \{(\top, 3), (p, 2)\} \) and \( \{(p \land q, 5), (p \land q, 5), (\neg p \land q, 3), (\neg p \land q, 3)\} \) define the same utility function over \( PS = \{p, q\} \) in \( L(\text{cubes}, \mathbb{R}, \Sigma) \), while the two goalbases \( \{(\neg p, 1)\} \) and \( \{(p \lor \neg p, 1), (p, 1)\} \) yield the same utility function in \( L(\text{clauses}, \mathbb{R}, \Sigma) \). However, two sum languages (and all of their sublanguages) do have unique representations, namely \( L(\text{pcubes}, \mathbb{R}, \Sigma) \) and \( L(\text{pclauses} + \top, \mathbb{R}, \Sigma) \), which we now set out to prove.

Theorem 3.4.2. \( L(\text{pcubes}, \mathbb{R}, \Sigma) \) has unique representations.

Proof. Given any utility function \( u \), the weight of each positive cube is uniquely determined: We must have \( w_\top = u(\emptyset) \), because \( \top \) is the only positive cube satisfied by the model \( \emptyset \), and furthermore \( w_\land X = u(X) - \sum_{Y \subseteq X} w_\land Y \) for any nonempty set \( X \).

The recursive definition of the weights given in the proof of Theorem 3.4.2 can be turned into a direct rule for computing weights by using the so-called Möbius inversion [Rota, 1964; Grabisch, 1997]:

\[
w_\land X = \sum_{Y \subseteq X} (-1)^{|X \setminus Y|} \cdot u(Y).
\]
3.4. Expressivity of Sum Languages

As an example, consider the utility function

\[
\begin{align*}
    u(\emptyset) &= 0 & u(\{c\}) &= 2 \\
    u(\{a\}) &= 1 & u(\{a, c\}) &= 6 \\
    u(\{b\}) &= 0 & u(\{b, c\}) &= 0 \\
    u(\{a, b\}) &= 0 & u(\{a, b, c\}) &= 7
\end{align*}
\]

over the three items \(a, b, c\). We now compute the representation of \(u\) in \(L(pcubes, \mathbb{R}, \Sigma)\) using the Möbius inversion. The pattern formed by equation (3.8) is best seen by calculating the weight for \(a \land b \land c\):

\[
w_{a \land b \land c} = -u(\emptyset) + u(\{a\}) + u(\{b\}) + u(\{c\}) - u(\{a, b\}) - u(\{a, c\}) - u(\{b, c\}) + u(\{a, b, c\}) = -0 + 1 + 0 + 2 - 0 - 6 + 0 + 7 = 4
\]

Because \(|\{a, b, c\}| = 3\) is odd, the utility of odd-sized subsets of \(\{a, b, c\}\) is added while the utility of even-sized subsets is subtracted. For the weights of even-length formulas, the opposite happens:

\[
w_{a \land b} = +u(\emptyset) - u(\{a\}) - u(\{b\}) + u(\{a, b\}) = +0 - 1 - 0 + 0 = -1
\]

For the remaining weights, we have

\[
\begin{align*}
w_{a \land c} &= +u(\emptyset) - u(\{a\}) - u(\{c\}) + u(\{a, c\}) = +0 - 1 - 2 + 6 = 3 \\
w_{b \land c} &= +u(\emptyset) - u(\{b\}) - u(\{c\}) + u(\{b, c\}) = +0 - 0 - 2 + 0 = -2 \\
w_a &= -u(\emptyset) + u(\{a\}) = -0 + 1 = 1 \\
w_b &= -u(\emptyset) + u(\{b\}) = -0 + 0 = 0 \\
w_c &= -u(\emptyset) + u(\{c\}) = -0 + 2 = 2 \\
w_{A \emptyset} &= +u(\emptyset) = +0 = 0
\end{align*}
\]

which gives us the goalbase

\[\{(a, 1), (c, 2), (a \land b, -1), (a \land c, 3), (b \land c, -2), (a \land b \land c, 4)\}\].

Note that we omit \(\top\) and \(b\), since \(w_{A \emptyset}\) and \(w_b\) turned out to be zero.

As can be seen from our example, the Möbius inversion is not an efficient method of calculating weights, due not only to the fact that we have \(2^{|PS|}\) weights to calculate, but also because for each weight we must sum a number of terms exponential in the length of the formula being weighted.
Theorem 3.4.3. \( L(p\text{clauses}, \mathbb{R}, \Sigma) \) has unique representations.

Proof. Let \( u \) be any utility function represented by positive clauses with weights \( u_X \) (with nonempty sets \( X \subseteq \mathcal{P}\mathcal{S} \)). Then for any \( Y \subseteq \mathcal{P}\mathcal{S} \), \( u(Y) \) must be equal to the sum of the weights \( u_X \) for which \( X \) and \( Y \) have a nonempty intersection:

\[
    u(Y) = \sum_{X \cap Y \neq \emptyset} u_X
    = \sum_{\emptyset \subset X \subseteq \mathcal{P}S} u_X - \sum_{\emptyset \subset X \subseteq \mathcal{P}S \setminus Y} u_X
    = u(\mathcal{P}S) - \sum_{\emptyset \subset X \subseteq \mathcal{P}S \setminus Y} u_X
\]

This shows that each weight \( u_X \) is uniquely determined: For singletons \( X = \{p\} \), by setting \( Y = \mathcal{P}S \setminus \{p\} \) in the above equation, we obtain \( w_p = u(\mathcal{P}S) - u(\mathcal{P}S \setminus \{p\}) \).

For general sets \( X \), using \( Y = \mathcal{P}S \setminus X \), we then obtain

\[
    w_X = u(\mathcal{P}S) - u(\mathcal{P}S \setminus X) - \sum_{\emptyset \subset X' \subset X} w_{X'},
\]

which completes the proof. \( \square \)

By unraveling the recursive definition of the weights given in the proof above, we can also provide a direct rule for computing weights in \( L(p\text{clauses}, \mathbb{R}, \Sigma) \), similar to the Möbius inversion:

\[
    w_X = \sum_{Y \subseteq X} (-1)^{|X \setminus Y| + 1} \cdot u(\mathcal{P}S \setminus Y) \tag{3.9}
\]

Furthermore, we have the following corollary due to the fact that no positive clause is a tautology:

Corollary 3.4.4. \( L(p\text{clauses} + \top, \mathbb{R}, \Sigma) \) has unique representations.

Finally, we note that having unique representations is a property which is preserved in all sublanguages of any language having the property; hence, it follows that many other languages not explicitly mentioned in this section, e.g., \( L(\text{atoms}, \mathbb{R}^+, \Sigma) \), also have unique representations.

While we can use the Möbius inversion for demonstrating that \( L(p\text{cubes}, \mathbb{R}, \Sigma) \) has unique representations, and a similar construct for \( L(p\text{clauses}, \mathbb{R}, \Sigma) \), it is also possible to show this by application of a more general method. The problem of determining whether a utility function \( u \) has a unique representation in a given language \( L \) amounts to examining the system of linear equations which describes \( u \) in \( L \). A language \( L \) has \( |L/\equiv| = m \) distinct nonequivalent formulas, and over a set of atoms \( \mathcal{P}\mathcal{S} \) there are \( 2^{|\mathcal{P}\mathcal{S}|} = n \) states. Each state \( i \in 2^{\mathcal{P}\mathcal{S}} \) defines a constraint

\[
    a_{i1}w_1 + \ldots + a_{im}w_m = b_i
\]
where \( a_{ij} \in \{0, 1\} \), depending on whether formula \( j \) is true in state \( i \); and \( b_i = u(X_i) \), where \( X_i \) is the set of true atoms in state \( i \). Taken together as matrices \( Ax = b \), we have:

\[
\begin{bmatrix}
a_{11} & \cdots & a_{1n} \\
a_{21} & \cdots & a_{2n} \\
\vdots & \ddots & \vdots \\
a_{m1} & \cdots & a_{mn}
\end{bmatrix}
\begin{bmatrix}
w_1 \\
w_2 \\
\vdots \\
w_n
\end{bmatrix}
= 
\begin{bmatrix}
b_1 \\
b_2 \\
\vdots \\
b_m
\end{bmatrix}
\]

That is, the \( w_i \) are the weights, the \( b_i \) are the values of the utility function, and the \( a_{ij} \) mark which formulas are true in which states. If \( n = m \), i.e., if the number of formulas in the language over \( \mathcal{P} \mathcal{S} \) equals the number of states over \( \mathcal{P} \mathcal{S} \), then the matrix \( A \) will be square. This makes available to us a well-known fact from linear algebra, viz. that the determinant of the square matrix \( A \) is nonzero only when the system has a single, unique solution [Anton, 1994, Theorem 2.3.6].

Using this, we give an alternative proof of Theorem 3.4.3:

**Proof.** Suppose that \( G \) contains only positive clauses and generates \( u_n \), where \( n = |\mathcal{P} \mathcal{S}| \). We can write one constraint for each state except \( \emptyset \), so we have \( 2^n - 1 \) constraints. (The constraint for \( \emptyset \) can be omitted, since all positive clauses are false in that case.) We have the clause \( \bigvee X \) for each nonempty \( X \subseteq \mathcal{P} \mathcal{S} \), so also \( 2^n - 1 \) distinct nonequivalent clauses, and hence \( 2^n - 1 \) variables for weights. The coefficient matrix formed by the states and formulas will be square (\( 2^n - 1 \) rows and columns), so the strategy described above—proving that this matrix has nonzero determinant—is applicable.

Enumerate the positive clauses such that the index \( j \) codes for the positive clause \( \varphi_j = \bigvee \{p_k \mid j \& 2^k \neq 0\} \), where ‘\&’ stands for bitwise conjunction. E.g., \( \varphi_7 = p_0 \lor p_1 \lor p_2 \), because \( 7 = 2^0 + 2^1 + 2^2 \). Then, let \( a_{ij} = 1 \) if \( i \& j \neq 0 \), and \( a_{ij} = 0 \) otherwise. This sets \( a_{ij} = 1 \) iff clause \( j \) is true in state \( i \). In other words, each row of \( A_n \) is a state, and the ones in a row mark the positive clauses which are true in that state.

Now observe that \( A_1 = [1] \) and \( A_{n+1} \) is the block matrix

\[
A_{n+1} = 
\begin{bmatrix}
0 & & & & \\
A_n & \cdots & A_n & & \\
0 & & & & \\
0 & 1 & 1 & \cdots & 1 \\
A_n & & & & 1 \\
& & & & 1
\end{bmatrix}
\]

where \( 1 \) is a matrix of the appropriate size, with every element a 1. The additional rows in \( A_{n+1} \) (over \( A_n \)) are for states in which \( p_n \), the new variable, is true. The middle row is the state where only \( p_n \) is true, and from there down \( p_n \) is true in
every state. With respect to the other variables, the states in the bottom half
repeat the states in the top half. The additional columns in $A_{n+1}$ are for positive
clauses which contain $p_n$. The middle column is for $p_n$, the degenerate positive
clause formed by that variable alone, and the columns thereafter repeat the first
$2^{n-1} - 1$ columns with $p_n$ as an additional disjunct. Therefore, the upper left and
upper right blocks repeat $A_n$ since no state there makes $p_n$ true; the lower left
block repeats $A_n$ since no clause there contains $p_n$; and the lower right block is all
ones because every state and clause there contains $p_n$.

Clearly, $\det(A_1) = 1$. Suppose that $\det(A_n) \neq 0$. To show that $\det(A_{n+1}) \neq 0$,
we will twice use the following fact about determinants of block matrices:

**Fact 3.4.5.** For the block matrix $\begin{bmatrix} A & B \\ C & D \end{bmatrix}$,

$$\det\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \det(A) \det(D - CA^{-1}B)$$

where $A$ is $m \times m$, $B$ is $m \times n$, $C$ is $n \times m$ and $D$ is $n \times n$.

Slice $A_{n+1}$ into blocks like so,

$$A = \begin{bmatrix} A_n & 0 & \vdots & A_n \\ 0 & \cdots & 0 & 1 & 1 & \cdots & 1 \\ \vdots & 1 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ A_n & : & 1 & : & : & : & 1 \end{bmatrix} = B$$

$$C = \begin{bmatrix} A_n \\ 0 \vdots 0 \\ 1 \\ 1 \\ : \\ 1 \end{bmatrix} = D$$

and note that $A = A_n = A_n^{-1}$ (because the $A_i$ are symmetric about their main
diagonal). Further,

$$CA^{-1} = \begin{bmatrix} 0 & \cdots & 0 \\ A_n & \vdots & 0 \end{bmatrix} A^{-1} = \begin{bmatrix} 0 & \cdots & 0 \\ 1 \end{bmatrix}$$

$$CA^{-1}B = \begin{bmatrix} 0 & \cdots & 0 \\ 1 & \vdots & A_n \end{bmatrix} \begin{bmatrix} 0 & \cdots & 0 \\ : & : & : & : & : \end{bmatrix} = \begin{bmatrix} 0 & \cdots & 0 \\ : & : & : & : & : \end{bmatrix}$$

$$D - CA^{-1}B = 1 - \begin{bmatrix} 0 & \cdots & 0 \\ : & : & : & : & : \end{bmatrix} = \begin{bmatrix} 1 & \cdots & 1 \\ : & : & : & : & : \end{bmatrix}$$

where $I$ and $1$ are an identity matrix and a matrix of ones, respectively, of the
appropriate sizes.
3.4. Expressivity of Sum Languages

Next, slice $D - CA^{-1}B$ into blocks like so,

\[
A' = \begin{bmatrix}
1 & 1 & \ldots & 1 \\
1 & & & \\
& 1 - A_n & \\
& & 1
\end{bmatrix} = B'
\]

\[
C' = \begin{bmatrix}
\vdots \\
1 - A_n \\
1
\end{bmatrix} = D'
\]

and then by Fact 3.4.5, we have that

\[
det(D - CA^{-1}B) = det(A') det(D' - C'A'^{-1}B')
\]

\[
= det(1 - A_n) det\left(1 - A_n - \begin{bmatrix} 1 \\
\vdots \\
1 \end{bmatrix} \begin{bmatrix} 1 & \ldots & 1 \end{bmatrix}\right)
\]

\[
= det(1 - A_n - 1)
\]

\[
= det(-A_n)
\]

\[
= -det(A_n).
\]

Applying Fact 3.4.5 a second time, we have that

\[
det(A_{n+1}) = det(A_n) det(D - CA^{-1}B) = -det(A_n)^2 = -1 \neq 0,
\]

which completes the induction.

Having shown that $det(A_n)$ is nonzero, it follows that the system has exactly one solution. Therefore $G$ is the unique positive-clause generator of $u_n$. \qed

The next proof demonstrates the generality of the method used in the previous proof. Here we use block matrices to show that $L(p\text{cubes} - \top, R, \Sigma)$ has unique representations. (This differs from Theorem 3.4.2 in that we omit $\top$ from the language, in order that this proof may parallel the previous one. If $\top$ were to be included, a similar proof could be carried out by adding to the $A_n$ matrices a column for the formula $\top$ and a row for the state where all atoms are false, in order to keep these matrices square.)

Proof. Suppose that $G$ contains only positive cubes and generates $u_n$. Proceed as above. Let the coefficient matrix $A_n$ be such that

\[
a_{ij} = \begin{cases} 
1 & \text{if } i \mid j = i \\
0 & \text{otherwise}
\end{cases}
\]

where ‘$\mid$’ is bitwise disjunction. So $a_{ij} = 1$ iff the ‘on’ bits in $j$ are a subset of the ‘on’ bits in $i$, which can be thought of as encoding the state description for constraint $i$, where each bit corresponds to one of the $n$ atoms; $j$ codes for the cube $\varphi_j = \bigwedge\{p_k \mid j \& 2^k = 1\}$. 
Now observe that $A_1 = [1]$ and $A_{n+1}$ is the block matrix

$$A_{n+1} = \begin{bmatrix}
0 & 0 \\
A_n & 0 \\
0 & 1 \\
A_n & A_n \\
1 & 1
\end{bmatrix}$$

where 0 is a matrix of the appropriate size, with every element set to 0. Slice $A_{n+1}$ into blocks like so,

$$A = \begin{bmatrix}
A_n & 0 \\
0 & 0 \\
0 & 1 \\
A_n & A_n \\
1 & 1
\end{bmatrix} = B$$

$$C = \begin{bmatrix}
A_n & 0 \\
0 & 0 \\
0 & 1 \\
A_n & A_n \\
1 & 1
\end{bmatrix} = D$$

noting that

$$CA^{-1}B = \begin{bmatrix}
0 & \cdots & 0 \\
\vdots & & \vdots \\
0 & & 0
\end{bmatrix} = 0$$

because $B$ is a zero matrix, and so $\det(D - CA^{-1}B) = \det(D - 0) = \det(D)$.

Next, slice $D$ into blocks like so:

$$A' = \begin{bmatrix}
1 & 0 & \cdots & 0 \\
1 & & & \\
\vdots & \ddots & \ddots & \vdots \\
1 & & & \ddots \\
A_n & & & \\
1 & & & \ddots \\
1 & & & \ddots
\end{bmatrix} = B'$$

$$C' = \begin{bmatrix}
A_n \\
1 \\
\vdots \\
1
\end{bmatrix} = D'$$

Applying Fact 3.4.5 to $D$, we have

$$\det(D) = \det(A') \det(D' - C'A'^{-1}B') = \det [1] \det \left( A_n - \begin{bmatrix}
1 \\
\vdots \\
1
\end{bmatrix} \begin{bmatrix}
1 & 0 & \cdots & 0
\end{bmatrix}\right) = \det(A_n - 0) = \det(A_n).$$
Applying Fact 3.4.5 a second time, we have that
\[
\det(A_{n+1}) = \det(A_n) \det(D - CA^{-1}B) = \det(A_n) \det(D) = \det(A_n)^2 = 1 \neq 0,
\]
which completes the induction.

Having shown that \(\det(A_n)\) is nonzero, it follows that the system has exactly one solution. Therefore \(G\) is the unique positive-cube generator of \(u_n\). \(\square\)

This method can be used to investigate, for any language where the formulas and states give a square matrix, whether that language has the uniqueness property. Though calculating determinants appears much more involved, it is in fact easier to carry out than proving uniqueness by giving a formula for computing weights. Calculating the coefficient matrix is entirely mechanical and the number of different ways that it may be decomposed into blocks is finite and small; once armed with the general insight of how to proceed, it is not difficult to compute the needed determinants and resolve whether a language has unique representations. (Note that had \(\det(A_n) = 0\), that would have been sufficient to prove that the language in question lacked unique representations.) In contrast, finding a formula for weights is a matter of trial and error; while the result may be more pleasing, the process is less so.

### 3.4.3 Correspondences

We now address the following question: What class of utility functions can we model using a given language? As much as possible we will strive for exact characterization results that establish the correspondence between a natural goalbase language and a commonly used class of utility functions.

For the next proof, and indeed much of this chapter, we will make frequent use of the fact that whenever \(\Phi \subseteq \Phi'\) and \(W \subseteq W'\), then \(U(\Phi, W, F) \subseteq U(\Phi', W', F)\).

**Theorem 3.4.6.** Each of the following classes:

- \(U(k\text{-}pcubes, \mathbb{R}, \Sigma)\)
- \(U(k\text{-}cubes, \mathbb{R}, \Sigma)\)
- \(U(k\text{-}pclauses + \top, \mathbb{R}, \Sigma)\)
- \(U(k\text{-}pforms, \mathbb{R}, \Sigma)\)
- \(U(k\text{-}forms, \mathbb{R}, \Sigma)\)

is equal to the class of all \(k\)-additive utility functions.

**Proof.** Inspection of the definition of \(k\)-additivity shows that it is simply a notational variant of the language based on positive cubes of length \(\leq k\): A \(k\)-additive function can be represented by a mapping \(m: \mathcal{PS}_k \rightarrow \mathbb{R}\), and we can define a bijection \(f\) from such mappings \(m\) onto goalbases \(G \in L(k\text{-}pcubes, \mathbb{R}, \Sigma)\):

\[
f: m \mapsto \{(p_1 \land \ldots \land p_k, \alpha) \mid m(\{p_1, \ldots, p_k\}) = \alpha\}.
\]
Clearly \( m = u_f(m) \). That is, \( \mathcal{U}(k\text{-}pcubes, \mathbb{R}, \Sigma) \) is the class of all \( k \)-additive utility functions.

Next, we show that all six languages are expressively equivalent. By language inclusion, we have the following:

\[
\begin{align*}
\mathcal{U}(k\text{-}cubes, \mathbb{R}, \Sigma) & \subseteq \mathcal{U}(k\text{-}forms, \mathbb{R}, \Sigma) \supseteq \mathcal{U}(k\text{-}clauses, \mathbb{R}, \Sigma) \\
\mathcal{U}(k\text{-}pcubes, \mathbb{R}, \Sigma) & \subseteq \mathcal{U}(k\text{-}pforms, \mathbb{R}, \Sigma) \supseteq \mathcal{U}(k\text{-}pclauses + \top, \mathbb{R}, \Sigma)
\end{align*}
\]

Taken together, equivalences (3.1) and (3.3) from Fact 3.4.1 can be used to transform any goalbase in \( L(k\text{-}forms, \mathbb{R}, \Sigma) \) into an equivalent goalbase in \( L(k\text{-}pcubes, \mathbb{R}, \Sigma) \). (Use (3.3) to eliminate all disjunctions, then (3.1) to eliminate all negations. While these equivalences add more formulas, they never add longer formulas.) Thus, \( \mathcal{U}(k\text{-}pcubes, \mathbb{R}, \Sigma) = \mathcal{U}(k\text{-}forms, \mathbb{R}, \Sigma) \), which collapses the leftmost four classes.

Equivalences (3.2) and (3.4) from Fact 3.4.1 can be used to transform any goalbase in \( L(k\text{-}forms, \mathbb{R}, \Sigma) \) into an equivalent goalbase in \( L(k\text{-}pclauses, \mathbb{R}, \Sigma) \). (Use (3.4) to eliminate all conjunctions, then (3.2) to eliminate all negations.) Thus, \( \mathcal{U}(k\text{-}pclauses, \mathbb{R}, \Sigma) = \mathcal{U}(k\text{-}forms, \mathbb{R}, \Sigma) \), which collapses the rightmost four classes.

This completes the proof, as all six classes have been collapsed together, and one of them has been shown to equal to the class of \( k \)-additive functions.

The next lemma clarifies the effect that the ability to express tautologies in a language has on the class of utility functions that can be defined. Roughly speaking, any utility function \( u \) expressible in a language based on strictly positive formulas must be normalized, i.e., will satisfy \( u(\emptyset) = 0 \).

For the next proof, we make use of translations of utility functions. The (affine) translation function \( t_c \) shifts a utility function \( u \) such that \( t_c(u(X)) = u(X) + c \) for all \( X \subseteq PS \). A property \( P \) is invariant under translation if, for all utility functions \( u, c \in \mathbb{R} \), and \( X \subseteq PS \), \( u \) has property \( P \) iff \( t_c(u) \) has property \( P \).

**Lemma 3.4.7.** Fix \( \Phi \) as a strictly positive set of formulas and \( P \) a property of utility functions which is invariant under translation. Then \( \mathcal{U}(\Phi, W, \Sigma) \) is the class of normalized utility functions with property \( P \) iff \( \mathcal{U}(\Phi \cup \{\top\}, W, \Sigma) \) is the class of utility functions with property \( P \).

**Proof.** (\( \Rightarrow \)) Suppose that \( \mathcal{U}(\Phi, W, \Sigma) \) is the class of normalized utility functions with property \( P \). First, we show that every \( u_G \in \mathcal{U}(\Phi \cup \{\top\}, W, \Sigma) \) has property \( P \): Fix \( u_G \in \mathcal{U}(\Phi \cup \{\top\}, W, \Sigma) \). \( G \setminus \{(\top, w) \mid w \in W\} \in L(\Phi, W, \Sigma) \) and so \( u_G\setminus\{(\top, w)\mid w\in W\} \) has property \( P \) by hypothesis. \( u_G = t_w(u_G\setminus\{(\top, w)\mid w\in W\}) \), so by invariance \( u_G \) has property \( P \).

Next, we show that every \( u \) with property \( P \) is in \( \mathcal{U}(\Phi \cup \{\top\}, W, \Sigma) \): Fix \( u \) with property \( P \). The translation \( t_{u(\emptyset)}(u) \) is normalized and has property \( P \) by
3.4. Expressivity of Sum Languages

invariance, so \( t_{u(\emptyset)}(u) \in U(\Phi, W, \Sigma) \) by hypothesis. Let \( G \) represent \( t_{u(\emptyset)}(u) \) in \( L(\Phi, W, \Sigma) \). Then \( G \cup \{(\top, u(\emptyset))\} \in L(\Phi \cup \{\top\}, W, \Sigma) \) and \( u_{G \cup \{(\top, u(\emptyset))\}} = u \).

(\( \Leftarrow \)) Suppose that \( U(\Phi \cup \{\top\}, W, \Sigma) \) is the class of utility functions with property \( P \). First, we show that every \( u \in U(\Phi, W, \Sigma) \) is normalized and has property \( P \): Fix \( u \in U(\Phi, W, \Sigma) \). Normalization follows due to \( \Phi \) being strictly positive. \( G \cup \{(\top, w)\} \in L(\Phi \cup \{\top\}, W, \Sigma) \) for any \( w \in W \), and by hypothesis \( u_{G \cup \{(\top, w)\}} \) has property \( P \). \( u_{G \cup \{(\top, w)\}} = t_w(u_G) \), so by invariance has property \( P \).

Next, we show that every normalized \( u \) with property \( P \) is in \( U(\Phi \cup \{\top\}, W, \Sigma) \): Fix \( u \) normalized and with property \( P \). For any \( w \in W \), \( t_w(u) \) has property \( P \) by invariance and so is in \( U(\Phi \cup \{\top\}, W, \Sigma) \). Let \( G \) represent \( t_w(u) \) in \( L(\Phi \cup \{\top\}, W, \Sigma) \). Since \( \Phi \) is strictly positive, \( (\top, w) \in G \). Then \( u_{G \setminus \{(\top, w)\}} = t_w^{-1}(t_w(u)) = u \), and so \( u \in U(\Phi \cup \{\top\}, W, \Sigma) \).

Finally, it is easy to see that we have the following as a corollary:

**Corollary 3.4.8.** Every utility function in \( U(\Phi, W, \Sigma) \) is normalized if \( \Phi \) is strictly positive or \( W = \{0\} \).

Next we explore the class of \( k \)-additive utility functions for specific values of \( k \). It is a well-known fact that any utility function is \( k \)-additive for some \( k \in \mathbb{N} \) (certainly for \( k = |PS| \)). (This is why we refer to general functions as \( \omega \)-additive. See Chapter 2, note 1.) Our next result is therefore an immediate corollary of Theorem 3.4.6.

**Corollary 3.4.9.** Each of the following classes:

- \( U(pcubes, \mathbb{R}, \Sigma) \)
- \( U(cubes, \mathbb{R}, \Sigma) \)
- \( U(pclauses + \top, \mathbb{R}, \Sigma) \)
- \( U(clauses, \mathbb{R}, \Sigma) \)
- \( U(pforms, \mathbb{R}, \Sigma) \)
- \( U/forms, \mathbb{R}, \Sigma) \)

is equal to the class of all utility functions.

**Corollary 3.4.10.** \( U(literals, \mathbb{R}, \Sigma) \) and \( U(\text{atoms} + \top, \mathbb{R}, \Sigma) \) are the class of all modular utility functions, and \( U(\text{atoms}, \mathbb{R}, \Sigma) \) is the class of all normalized modular utility functions.

**Proof.** The set of 1-pcubes is equal to the set of atoms together with \( \top \). Therefore, by Theorem 3.4.6, \( U(\text{atoms} + \top, \mathbb{R}, \Sigma) \) is the class of all modular functions. The class of 1-cubes is equal to the class of literals together with \( \top \). But by equivalence (3.7) of Fact 3.4.1, literals alone have the same expressive power as literals together with \( \top \). Hence, again by Theorem 3.4.6, \( U(literals, \mathbb{R}, \Sigma) \) is the class of all modular functions. The fact that \( U(\text{atoms}, \mathbb{R}, \Sigma) \) is the class of all normalized modular utility functions follows from Lemma 3.4.7. \( \Box \)
Chapter 3. Expressivity

For the remainder of this section we consider languages where the set of weights is restricted to the positive reals. Clearly, the utility functions that can be so expressed will be nonnegative, i.e., \( u(X) \geq 0 \) for all \( X \subseteq \mathcal{P} \mathcal{S} \). The question is whether we can express all nonnegative utility functions in this manner.

**Theorem 3.4.11.** \( \mathcal{U}(\text{cubes}, \mathbb{R}^+, \Sigma) \) and \( \mathcal{U}(\text{forms}, \mathbb{R}^+, \Sigma) \) are the class of all nonnegative utility functions.

**Proof.** Clearly every \( u \in \mathcal{U}(\text{forms}, \mathbb{R}^+, \Sigma) \), and by inclusion, also every \( u \in \mathcal{U}(\text{cubes}, \mathbb{R}^+, \Sigma) \), is nonnegative. For the converse, suppose that \( u \) is nonnegative. Then define
\[
G = \left\{ (\bigwedge M \cup \neg \bar{M}, u(M)) \mid M \subseteq \mathcal{P} \mathcal{S} \text{ and } u(M) \neq 0 \right\}
\]
and observe that \( u_G = u \) and that \( G \) contains only positively-weighted cubes. \( \square \)

That is, general formulas as well as cubes are fully expressive over nonnegative utility functions when weights are required to be positive. As we shall see next, the same is not true for clauses.

**Theorem 3.4.12.** \( \mathcal{U}(\text{clauses}, \mathbb{R}^+, \Sigma) \) is a proper subset of all nonnegative utility functions.

**Proof.** By definition, \( \mathcal{L}(\text{clauses}, \mathbb{R}^+, \Sigma) \subset \mathcal{L}(\text{forms}, \mathbb{R}^+, \Sigma) \), so by Theorem 3.4.11, \( \mathcal{U}(\text{clauses}, \mathbb{R}^+, \Sigma) \) contains only nonnegative utility functions. This utility function over \( \mathcal{P} \mathcal{S} = \{p, q\} \) demonstrates that the inclusion is strict:
\[
u(X) = \begin{cases} 1 & X = \{p, q\} \\ 0 & \text{otherwise.} \end{cases}
\]
The following five constraints must be satisfied if a \( G \in \mathcal{L}(\text{clauses}, \mathbb{R}^+, \Sigma) \) exists which represents \( u \):
\[
\begin{align*}
w_p + w_q + w_{p \vee q} + w_{p \vee \neg q} + w_{p \vee \neg p} &= 1 \\ w_p + w_{\neg q} + w_{p \vee q} + w_{p \vee \neg q} + w_{p \vee \neg p} &= 0 \\ w_{\neg p} + w_q + w_{p \vee q} + w_{p \vee \neg q} + w_{p \vee \neg p} &= 0 \\ w_{\neg p} + w_{\neg q} + w_{p \vee q} + w_{p \vee \neg q} + w_{p \vee \neg p} &= 0 \\ w_{\varphi} &\geq 0 \text{ for all clauses } \varphi
\end{align*}
\]
Together, constraints (3.11), (3.12), (3.13), and (3.14) force \( w_{\varphi} = 0 \) for every clause \( \varphi \), contradicting (3.10). \( \square \)

\( \mathcal{L}(\text{clauses}, \mathbb{R}^+, \Sigma) \) seems not to characterize a natural class of functions. For (strictly) positive formulas with positive weights, on the other hand, we do obtain nice correspondences. Recall that a utility function \( u \) is monotone if, for all \( X, Y \subseteq \mathcal{P} \mathcal{S}, u(X) \leq u(Y) \) whenever \( X \subseteq Y \).
Theorem 3.4.13. $\mathcal{U}(\text{spforms, } \mathbb{R}^+, \Sigma)$ is the class of all normalized monotone utility functions.

Proof. No strictly positive formula is a tautology, so every $u \in \mathcal{U}(\text{spforms, } \mathbb{R}^+, \Sigma)$ is normalized, and because all weights and formulas are positive, it is also monotone.

For the converse: Let $u$ be an arbitrary normalized monotone utility function. We construct a $G \in \mathcal{L}(\text{spforms, } \mathbb{R}^+, \Sigma)$ for which $u_G = u$ as follows. Define a sequence of utility functions $u_1, \ldots, u_n$ such that

$$u_k(X) = \max\{u(X') \mid X' \subseteq X \text{ and } |X'| \leq k\}.$$ 

In this way, $u_1 = \max_{a \in X} u(\{a\})$ and $u_n = u$. Additionally, we define $u_0(X) = 0$ for all $X$, for convenience. Observe that we can use the these $u_i$ to decompose $u$ such that $u = \sum_{k=1}^{n} (u_k - u_{k-1})$, and so if we can construct a goalbase for each $u_k - u_{k-1}$, then the union of those goalbases will be a goalbase for $u$. Hereafter, we will abbreviate $u_k - u_{k-1}$ to $u_k^*$. To construct $G_k$, a goalbase for $u_k^*$, let $X_0 = \emptyset$ and $\langle X_1, \ldots, X_{n_k} \rangle$ be the set of size-$k$ subsets of $\mathcal{P} \mathcal{S}$, ordered so that $u_k^*(X_i) \leq u_k^*(X_j)$ for $i < j$. Then let

$$G_k = \left\{ \left( \bigvee_{j=i}^{n_k} \bigwedge X_j, u_k^*(X_i) - u_k^*(X_{i-1}) \right) \mid 1 \leq i \leq \binom{n}{k} \right\}$$

from which it can easily, though tediously, be checked that $u_{G_k} = u_k^*$. (For example, if $\mathcal{P} \mathcal{S} = \{a, b, c\}$ and $u(a) \leq u(b) \leq u(c)$, then $G_1 = \{(a \lor b \lor c, u(a)), (b \lor c, u(b) - u(a)), (c, u(c) - u(b))\}$. View items $a$, $b$, and $c$ as substitutes, but with $b$ conferring a bonus over $a$, and $c$ a further bonus over $b$. This is the structure which can be seen in $G_1$. Higher-order $G_i$s capture this same idea, but for sets of items larger than singletons.)

Finally, let $G = \bigcup_{k=1}^{n} G_k$. Now $u_G = u$, since for each $k$, $u_{G_k} = u_k^*$ and $\sum_{k=1}^{n} u_k^* = u_n = u$. Finally, observe that every formula in $G$ is strictly positive; and all the weights $u_k^*(X_i) - u_k^*(X_{i-1})$ are nonnegative by virtue of the ordering declared over the $X_i$.

Note that in the preceding theorem, we could also add nonnegative as a property, because normalization and monotonicity together imply nonnegativity. An application of Lemma 3.4.7 yields the following corollary:

Corollary 3.4.14. $\mathcal{U}(\text{pforms, } \mathbb{R}^+, \Sigma)$ is the class of all nonnegative monotone utility functions.

Supermodularity seems not to correspond directly to a natural goalbase language, but we can characterize a large subclass.

Theorem 3.4.15. $\mathcal{U}(\text{pcubes, } \mathbb{R}^+, \Sigma)$ is the class of all nonnegative utility functions satisfying the constraint $\sum_{Y \subseteq X} (-1)^{|Y|} \cdot u(Y) \geq 0$ for all $X \subseteq \mathcal{P} \mathcal{S}$. 

3.4. Expressivity of Sum Languages
Proof. That $U(p\text{cubes}, \mathbb{R}^+, \Sigma)$ is the class of all nonnegative utility functions satisfying $\sum_{Y \subseteq X} (-1)^{|Y|} \cdot u(Y) \geq 0$ immediately follows from the fact that the weight of any positive cube is determined by the Möbius inversion as stated in equation (3.8).

Note that the property corresponding to $U(p\text{cubes}, \mathbb{R}^+, \Sigma)$ implies nonnegativity, monotonicity, and supermodularity. Nonnegativity and monotonicity follow from Corollary 3.4.14. For supermodularity, suppose that $G \in \mathcal{L}(p\text{cubes}, \mathbb{R}^+, \Sigma)$. Then

$$u_G(X \cup Y) = \sum_{Z \subseteq X \cup Y} w_Z \cdot \min(1, |Z|)$$

which is equivalent to the supermodularity condition.

The utility function $u: X \mapsto \max(1, |X|)$ shows that there are supermodular utility functions which are not in $U(p\text{cubes}, \mathbb{R}^+, \Sigma)$. As can easily be checked, if $\mathcal{P}S = \{p, q, r\}$, then expressing $u$ in terms of positive cubes requires the use of a negative weight: $w_{p \lor q \lor r} = -1$. The previous theorem holds also for $U(s\text{cubes}, \mathbb{R}^+, \Sigma)$ if “nonnegative” is replaced with “normalized”.

**Theorem 3.4.16.** $U(p\text{clauses}, \mathbb{R}^+, \Sigma)$ is the class of all nonnegative utility functions satisfying the constraint $\sum_{Y \subseteq X} (-1)^{|Y|+1} \cdot u(\mathcal{P}S \setminus Y) \geq 0$ for all $X \subseteq \mathcal{P}S$.

**Proof.** Follows from the fact that weights of positive clauses are determined by equation (3.9).

Note that the property corresponding to $U(p\text{clauses}, \mathbb{R}^+, \Sigma)$ implies monotonicity, normalization, and submodularity. Normalization and monotonicity follow from Theorem 3.4.13. To show submodularity, let $G \in \mathcal{L}(p\text{clauses}, \mathbb{R}^+, \Sigma)$ and let $X, Y \subseteq \mathcal{P}S$. For positive clauses $\varphi$, $X \cup Y \models \varphi$ together with $X \not\models \varphi$ implies $Y \models \varphi$. Furthermore, $X \not\models \varphi$ implies $X \cap Y \not\models \varphi$. Therefore:

$$\{(\varphi, w) \in G \mid X \cup Y \models \varphi \text{ and } X \not\models \varphi\} \subseteq \{(\varphi, w) \in G \mid Y \models \varphi \text{ and } X \cap Y \not\models \varphi\}$$

As all the weights $w$ are positive, we immediately obtain the required inequality characterizing submodularity, namely $u_G(X \cup Y) - u_G(X) \leq u_G(Y) - u_G(X \cap Y)$.

An example which confirms that not all submodular utility functions belong to $U(p\text{clauses}, \mathbb{R}^+, \Sigma)$ is the function $u: X \mapsto \min(2, |X|)$ for $\mathcal{P}S = \{p, q, r\}$. On the one hand, $u$ is submodular; on the other we must have $w_{p \lor q \lor r} = -1$ if we
are to express \( u \) using positive clauses. The previous theorem holds also for \( \mathcal{U}(\text{pclauses} + \top, \mathbb{R}^+, \Sigma) \) if “normalized” is replaced with “nonnegative”.

The functions characterized by Theorem 3.4.15 are also known as belief functions, while those characterized by Theorem 3.4.16 are known as plausibility functions (when the functions are restricted to the interval \([0, 1] \)) [Dempster, 1967; Shafer, 1976].

### 3.4.4 Summary

Our correspondence results for sum languages are summarized in Figure 3.1. In the figure, each node represents one language we examined, and an arrow from one node to another indicates that the tail language is included in the head language. Within each node, the expressivity of the language is given, according to the key below:

- \( 1 \): 1-additive (modular)
- \( m \): monotone
- \( k \): \( k \)-additive
- \( \star \): plausibility function
- \( \omega \): \( \omega \)-additive (general)
- \( \dagger \): belief function
- \( n \): normalized
- \( \subset \): proper subset of
- \( + \): nonnegative
- \( \subseteq \): subset of

Where \( \subset \) (or \( \subseteq \)) is indicated, the language represents a (proper) subset of the class of utility functions with the given properties. In all other cases, the language represents exactly the class of utility functions with the given properties. The \( x \)-axis (increasing to the right) is the cubes axis, along which allowable cubes grow from length 1 up to \( \omega \); the \( y \)-axis (increasing into the page) is the clauses axis, also running from 1 to \( \omega \). The \( z \)-axis (decreasing upward) is the positivity axis and has three steps: strictly positive, positive, and general. Each language in the lower graph is a sublanguage of the corresponding language with general weights in the upper graph, but we have omitted these arrows for clarity.

We have not analyzed the interplay of bounding the length of formulas and restricting weights to positive reals in detail. By Theorem 3.4.6, any language restricting the length of formulas to at most \( k \) atoms can only generate \( k \)-additive utility functions. The opposite direction is less clear. While inspection of the proofs of Theorems 3.4.15 and 3.4.16 show that these results extend to the \( k \)-additive case in the expected manner, this is not so for Theorems 3.4.11 and 3.4.13. For instance, we do not know whether \( \mathcal{U}(k\text{-cubes}, \mathbb{R}^+, \Sigma) \) is the class of all nonnegative \( k \)-additive functions or only a subclass thereof.

### 3.5 Expressivity of Max Languages

In many ways, sum is the most obvious aggregator to consider. However, there are certain classes of utility functions—in particular, single-minded and unit-demand
Chapter 3. Expressivity

Figure 3.1: Summary of expressivity results for sum languages.
utility functions—which are not easily captured by sum languages. We turn now to max languages, both to exhibit languages which capture other classes of utility function and to serve as a study of how varying the aggregator affects the expressivity of goalbase languages.

We begin by establishing some simple results regarding equivalences between languages, which allow us to narrow down the range of languages to be considered in the remainder of the section. We then characterize the expressivity of the most important (distinct) languages.

When using a max language, only the weight of the most important goal satisfied by a given alternative matters. An example of a simple max language is \( L(\text{atoms}, \mathbb{R}, \text{max}) \), which allows us to assign a value to any atomic proposition and where the utility of a state is equal to the value of the most valuable proposition which is true in that state. The utility functions in \( U(\text{atoms}, \mathbb{R}, \text{max}) \) are known as unit-demand valuations (of which the aforementioned simple unit-demand valuation is a special case) in the literature on combinatorial auctions [Nisan, 2006]. We will return to prove that \( U(\text{atoms}, \mathbb{R}, \text{max}) \) does indeed have this property in Theorem 3.5.10.

Throughout this section, when the aggregator function is omitted from the notation, it is intended to be max.

### 3.5.1 Superfluous Goals

For max languages, some weighted goals in a goalbase may never contribute to the utility of an alternative. Here we introduce terminology for speaking about this kind of situation and state some simple facts about potentially superfluous goals.

**Definition 3.5.1 (Properties of Weighted Goals).**

- \((\varphi, w_\varphi) \in G\) is *dominated* if there exists a \((\psi, w_\psi) \in G\) such that \(\varphi \models \psi\) and \(w_\varphi < w_\psi\).
- \((\varphi, w_\varphi) \in G\) is *active* in a state \(M\) when \(M \models \varphi\) and for all \(\psi\), if \(M \models \psi\) then \(w_\psi \leq w_\varphi\).

Recall from Definition 3.2.3 that \((\varphi, w) \in G\) is superfluous if \(G \equiv_F G \setminus \{(\varphi, w)\}\). Note that if there are no models for which \((\varphi, w)\) is active, then the superfluity condition is fulfilled vacuously, so \((\varphi, w)\) is superfluous in that case.

**Fact 3.5.2.** Fix \((\varphi, w_\varphi) \in G\), and the aggregator as max. Then:

1. If \((\varphi, w_\varphi)\) is dominated, then \((\varphi, w_\varphi)\) is never active.
2. If \((\varphi, w_\varphi)\) is never active, then \((\varphi, w_\varphi)\) is superfluous.
3. If \((\varphi, w_\varphi)\) is superfluous, for every state \(M\) where \((\varphi, w_\varphi)\) is active, the set \(\{ (\psi, w_\psi) \mid M \models \psi \text{ and } w_\psi \text{ is maximal} \}\) is not a singleton.
Chapter 3. Expressivity

Note that superfluous does not imply never active: For example, in the goalbase

\{(a, 1), (b, 1), (a \land b, 1)\},

the weighted formula \((a \land b, 1)\) is superfluous, but nonetheless active in the state \(\{a, b\}\).

Since superfluous formulas are useless, they may be removed without harm. We can remove superfluous formulas from any \(G \in \mathcal{L}(\text{forms}, \mathbb{R}, \text{max})\) as follows:

\[
\text{for all } (\varphi, w_\varphi) \in G \text{ do} \\
\quad \text{for all } (\psi, w_\psi) \in G \text{ do} \\
\quad\quad \text{if } w_\varphi \leq w_\psi \text{ then} \\
\quad\quad\quad \text{if } \models \varphi \rightarrow \psi \text{ then} \\
\quad\quad\quad\quad G := G \setminus \{(\varphi, w_\varphi)\} \\
\quad\quad\quad \text{end if} \\
\quad\quad \text{end if} \\
\quad \text{end for} \\
\text{end for}
\]

Two aspects of this algorithm are noteworthy: First, we remove superfluous formulas one at a time. It would not be correct to remove all superfluous formulas in one step, as superfluity is defined relative to a particular goalbase and removal of a formula changes the goalbase. It is possible that some formulas which were superfluous will stop being so if another superfluous formula is removed. For example, all formulas in \(\{(a, 1), (\neg\neg a, 1)\}\) are superfluous under max; however, removing both produces the empty goalbase, which is not equivalent to the original.

Second, this algorithm is not efficient, as it requires a quadratic number of calls to an UNSAT oracle; we would like to do better than a coNP algorithm. In the next section, we will see that the only max languages which are (expressively) interesting are based on cubes. This fact greatly reduces the complexity of deciding whether a formula is superfluous, since deciding whether one cube implies another is polynomial: Given two satisfiable cubes \(\wedge X\) and \(\wedge Y\), \(\wedge X \rightarrow \wedge Y\) is a tautology iff the positive literals in \(Y\) are a subset of the positive literals in \(X\) and the negative literals in \(Y\) are a subset of the negative literals in \(X\).

By way of comparison, it is easy to see that these concepts of domination and activity are not useful when considering sum languages. The formula \((a \land b, 1)\) is never active in the goalbase \(\{(a, 2), (a \land b, 1)\}\), since \((a, 2)\) dominates it \((a \land b\) implies \(a\), and \(2 > 1\)), but nonetheless it is not superfluous—we cannot remove it and retain an equivalent goalbase under the sum aggregator.

**Fact 3.5.3.** Under the max aggregator, if \(G\) contains no superfluous formulas then every \((\varphi, w) \in G\) has a state in which it is uniquely active.
3.5. Expressivity of Max Languages

3.5.2 Goalbase Equivalences

We are interested in comparing languages generated by different types of goalbases, in particular those generated from the following restrictions on formulas: literals, cubes, clauses, and general formulas using both conjunction and disjunction; as well as positive formulas and those including negation. Regarding weights, we want to consider positive and general weights. This gives rise to \(4 \cdot 2 \cdot 2 = 16\) different languages. Here we establish several equivalences amongst languages and thereby show that we actually only need to consider a subset of all the languages that can be defined in this manner. Furthermore, if we are interested only in monotone utility functions, then we can further reduce the range of languages to consider.

We first show that disjunction is not an expressively helpful connective for max languages:

**Theorem 3.5.4.** \(G \cup \{(\varphi_1 \lor \ldots \lor \varphi_n, w)\} \equiv_{\text{max}} G \cup \{\varphi_i, w \mid 1 \leq i \leq n\} \).

*Proof.* Fix a state \(X\). If \(w_{\varphi_1 \lor \ldots \lor \varphi_n}\) is not the maximum \(w_{\psi}\) such that \(X \models \psi\), then some other \((\psi, w_{\psi})\) \(\in G\). Since \(w_{\varphi_i} = w_{\varphi_1 \lor \ldots \lor \varphi_n}\), then \(w_{\psi} > w_{\varphi_i}\) for all \(i\). In this case, \(G\) alone determines the value of the left and right goalbases.

If \(w_{\varphi_1 \lor \ldots \lor \varphi_n}\) is the maximum \(w_{\psi}\) such that \(X \models \psi\), then some \(\varphi_i\) are such that \(X \models \varphi_i\). For each such \(\varphi_i\), we have \(w_{\varphi_i} = w_{\varphi_1 \lor \ldots \lor \varphi_n}\) in the goalbase on the right, and so both the left and right have the same maximum.

This tells us that with max as our aggregator, disjunction as main connectives do not contribute to a language’s expressivity. In fact, as any formula has an equivalent representation in disjunctive normal form, this tells us that disjunction can never increase the expressive power of a max language. In particular, from Theorem 3.5.4 we get the following equivalences between languages:

**Corollary 3.5.5.** Fix \(W \subseteq \mathbb{R}\). Then:

1. \(U(p\text{clauses}, W, \text{max}) = U(\text{atoms}, W, \text{max})\),
2. \(U(\text{clauses}, W, \text{max}) = U(\text{literals}, W, \text{max})\),
3. \(U(p\text{forms}, W, \text{max}) = U(p\text{cubes}, W, \text{max})\),
4. \(U(\text{forms}, W, \text{max}) = U(\text{cubes}, W, \text{max})\).

*Proof.* For \(U(\text{forms}, W, \text{max}) = U(\text{cubes}, W, \text{max})\): Suppose that \((\varphi, w) \in G\). Without loss of generality, assume that \(\varphi = \psi_1 \lor \ldots \lor \psi_n\) is in DNF. By Theorem 3.5.4, we may replace \((\psi_1 \lor \ldots \lor \psi_n, w)\) by \((\psi_1, w), \ldots, (\psi_n, w)\) and preserve goalbase equivalence. Repeating this for each original formula in \(G\) converts \(G\) to the language \(\mathcal{L}(\text{cubes}, W, \text{max})\), since each \(\psi_i\) is a cube and the weights were left unchanged.
We use the same argument for each of the other cases, noting that the same transformation reduces a positive formula (in DNF) to a set of positive cubes, a clause to a set of literals, and a positive clause to a set of atoms.

The same is not true for sum languages. E.g., under summation clauses are more expressive than literals: For $W = \mathbb{R}$, clauses can express all utility functions, while literals can express only modular functions (see Corollary 3.4.10). For each of the equivalences in Corollary 3.5.5, there is a set of weights $W$ which violates it under summation.

Recall that a utility function $u$ is called monotone if $M \subseteq M'$ implies $u(M) \leq u(M')$. Monotonicity is a reasonable assumption for many applications, in particular if propositional variables are interpreted as goods. Next we show that negation is not a helpful operation in case we are only interested in modeling monotone functions.

**Theorem 3.5.6.** Fix $G$. Let $X^+$ be a set of positive literals, and $X^-$ a set of negative literals, such that no atom appears in both. If $u_{G \cup \{(\bigwedge X^+ \cup X^-, w)\}, \max}$ is monotone, then

$$G \cup \{(\bigwedge X^+ \cup X^-, w)\} \equiv_{\max} G \cup \{(\bigwedge X^+, w)\}.$$ 

**Proof.** There are two cases to consider: states which are supersets of $X^+$, and states which are not.

- Write $u$ for $u_{G \cup \{(\bigwedge X^+ \cup X^-, w)\}, \max}$. In states $M \supseteq X^+$, we have that $M \models \bigwedge X^+$. It must be the case that $u(M) \geq w$ because $u(X^+) \geq w$ and $u$ is monotone. Therefore, substituting $(\bigwedge X^+, w)$ for $(\bigwedge X^+ \cup X^-, w)$ cannot change the value of state $M$, since the value in $M$ is already at least $w$.

- In states $M \nsubseteq X^+$, we have that both $\bigwedge X^+ \cup X^-$ and $\bigwedge X^+$ are false, and so cannot be active. Thus substituting $(\bigwedge X^+, w)$ for $(\bigwedge X^+ \cup X^-, w)$ cannot change the value at $M$, as inactive formulas do not affect the value of a utility function.

Therefore, in all states $M$ we have that

$$u_{G \cup \{(\bigwedge X^+ \cup X^-, w)\}, \max}(M) = u_{G \cup \{(\bigwedge X^+, w)\}, \max}(M).$$

The following result shows that we can further reduce the range of languages to consider if we limit ourselves to monotone utility functions. It follows immediately from Theorem 3.5.6 and Corollary 3.5.5. (Note that $\bigwedge \emptyset = \top$.)

**Corollary 3.5.7.** Let MONO be the class of monotone utility functions. Fix $W \subseteq \mathbb{R}$. Then:

1. $\mathcal{U}(\text{clauses}, W, \max) \cap \text{MONO} = \mathcal{U}(\text{atoms} + \top, W, \max) \cap \text{MONO}$.

2. $\mathcal{U}(\text{forms}, W, \max) \cap \text{MONO} = \mathcal{U}(\text{pcubes}, W, \max) \cap \text{MONO}$.
3.5. Expressivity of Max Languages

3.5.3 Correspondences

Corollary 3.5.5 tells us that the interesting languages, expressivity-wise, are those based on cubes, positive cubes, literals, and atoms. We prove that cubes are expressively complete for the full range of utility functions and that positive cubes correspond to the class of monotone functions:

**Theorem 3.5.8.** \( U(\text{cubes}, \mathbb{R}, \text{max}) \) is the class of all utility functions.

**Proof.** Given a utility function \( u \), define

\[
G = \left\{ \left( \bigwedge X \cup \neg \bar{X}, u(X) \right) \mid X \subseteq \mathcal{P} \mathcal{S} \right\}
\]

Since the formulas are the states, and as such are mutually exclusive, exactly one weight will be active in each state, and so \( u(X) = u_G(X) \).

**Theorem 3.5.9.** \( U(\text{pcubes}, \mathbb{R}, \text{max}) \) is the class of monotone utility functions.

**Proof.** (\( \Rightarrow \)) Suppose that \( G \in U(\text{pcubes}, \mathbb{R}, \text{max}) \) but \( u_G \) is not monotone. So there are states \( M \subset M' \) such that \( u_G(M') < u_G(M) \). Then there is a \( (\varphi, u_G(M)) \in G \) which is active in \( M \) such that \( M \models \varphi \) but \( M' \not\models \varphi \). Since \( M' \supset M \), then there is some \( a \in M' \setminus M \) for which \( \varphi \models \neg a \). Therefore, \( \varphi \) is not a positive formula, which contradicts the hypothesis that \( \varphi \) is a pcube.

(\( \Leftarrow \)) If \( u \) is monotone, then let \( G = \{ (\bigwedge X, u(X)) \mid X \subseteq \mathcal{P} \mathcal{S} \} \). Note that for \( Y \subseteq X, u(Y) = w_{\bigwedge Y} \leq w_{\bigwedge X} = u(X) \) follows directly from the monotonicity of \( u \). In state \( X \), \( u_G(X) = \max \{ w_{\bigwedge Y} \mid Y \subseteq X \} = w_{\bigwedge X} \). Hence \( u_G(X) = u(X) \).

The class of unit-demand utility functions has no simple corresponding sum language, but does have a corresponding max language:

**Theorem 3.5.10.** \( U(\text{atoms}, \mathbb{R}, \text{max}) \) is the class of unit-demand utility functions.

**Proof.** Suppose that \( u \) is a unit-demand valuation, which by definition means that \( u(X) = \max_{a \in X} u(\{a\}) \). Construct a \( G \in U(\text{atoms}, \mathbb{R}, \text{max}) \) such that \( G = \{ (a, w) \mid a \in \mathcal{P} \mathcal{S}, u(\{a\}) = w \} \). Then

\[
u(X) = \max_{a \in X} u(\{a\}) = \max_{(a,w) \in G} w = u_{G,\text{max}}(X).
\]

Conversely, suppose that \( G \in U(\text{atoms}, \mathbb{R}, \text{max}) \), and note that the same series of equivalences holds.

We are not aware of a property of utility functions referred to in the literature which would characterize \( U(\text{literals}, \mathbb{R}, \text{max}) \). The desired property is a generalization of the unit-demand valuation that also allows us to specify a value for \textit{not} receiving a particular item.
By restricting the set of weights $W$ we can capture classes of utility functions with a particular range. $\mathcal{U}(\text{pcubes}, \mathbb{R}^+, \text{max})$, for instance, is the class of nonnegative monotone functions. This class is known to be equal to $\mathcal{U}(\text{pforms}, \mathbb{R}^+, \Sigma)$ (see Corollary 3.4.14).\footnote{To be precise, $\mathcal{U}(\text{pcubes}, \mathbb{R}^+, \text{max}) = \mathcal{U}(\text{pforms}, \mathbb{R}^+, \Sigma)$ over total functions only. Because the max languages can also express partially defined functions returning $-\infty$ for some states while sum languages cannot, if we expand our consideration to partially-defined utility functions, then $\mathcal{U}(\text{pcubes}, \mathbb{R}^+, \text{max}) \supset \mathcal{U}(\text{pforms}, \mathbb{R}^+, \Sigma)$.} This is a case where a syntactically simple language is more expressive with max than with sum.

On the other hand, some very simple classes of utility functions are hard to capture in structurally simple languages using max aggregation. For instance, recall that a utility function $u$ is called modular iff $u(M \cup M') = u(M) + u(M') - u(M \cap M')$ for all $M, M' \in 2^{PS}$. Modular functions are nicely captured by $\mathcal{U}(\text{literals}, \mathbb{R}, \Sigma)$ (see Corollary 3.4.10). However, there is no natural restriction to formulas that would allow us to characterize the modular functions under max aggregation. On the contrary, among the max languages considered here, only $\mathcal{L}(\text{cubes}, \mathbb{R}, \text{max})$ can express all modular functions, and this language is so powerful that it can actually express all utility functions.

In particular, $\mathcal{L}(k\text{-pcubes}, \mathbb{R}, \text{max})$ misses some modular utility functions when $k < |PS|$: Suppose that $u$ is modular and nonnegative, and at least $k + 1$ singleton states $\{p_1\}, \ldots, \{p_{k+1}\}$ have nonzero value. Then there is no $G \in \mathcal{L}(k\text{-pcubes}, \mathbb{R}, \text{max})$ such that $u(\{p_1, \ldots, p_{k+1}\}) = u_G(\{p_1, \ldots, p_{k+1}\})$, because $u_G(\{p_1, \ldots, p_{k+1}\})$ is the weight of some $k$-cube (because max is our aggregator) and by assumption if $X \subset \{p_1, \ldots, p_{k+1}\}$ then $u(X) < u(\{p_1, \ldots, p_{k+1}\})$. The same modular utility functions are missing from $\mathcal{L}(k\text{-cubes}, \mathbb{R}, \text{max})$, due to the fact that the addition of negation to the language is not helpful for representing monotone utility functions (see Theorem 3.5.6). As a result, we have the following:

**Theorem 3.5.11.** For all $k > j$, $\mathcal{U}(k\text{-pcubes}, W, \text{max}) \supset \mathcal{U}(j\text{-pcubes}, W, \text{max})$ and $\mathcal{U}(k\text{-cubes}, W, \text{max}) \supset \mathcal{U}(j\text{-cubes}, W, \text{max})$.

3.5.4 Summary

Our correspondence results for max languages are summarized in Figure 3.2. In the figure, each node represents one language we examined, and an arrow from one node to another indicates that the tail language is included in the head language. Within each node, the expressivity of the language is given, according to the key below:

| $\omega$ | $\omega$-additive (general) |
| +       | nonnegative |
| ‡       | general unit-demand |

Where $\subset$ is indicated, the language represents a proper subset of the class of utility functions with the given properties. In all other cases, the language...
3.5. Expressivity of Max Languages

Figure 3.2: Summary of expressivity results for max languages.
represents exactly the class of utility functions with the given properties. The $x$-axis (increasing to the right) is the cubes axis, along which allowable cubes grow from length 1 up to $\omega$; the $y$-axis (decreasing into the page) is the positivity axis and has two steps: positive and general. Each language in the lower graph is a sublanguage of the corresponding language with general weights in the upper graph, but we have omitted these arrows for clarity. Note that several languages do not appear in the figure due to being expressively equivalent to some language which does appear there. For max languages, this is due to the fact expressed in Corollary 3.5.5, namely that disjunction does not contribute expressivity to max languages.

3.6 Odds and Ends

In this section, we present two results which fit nowhere else, both of which bear on how weights affect expressivity. The first is the observation that the weights can be limited to the range of the representable utility functions if every state formula is in the language:

**Theorem 3.6.1.** If $\Phi$ contains all state formulas and $F \in \{\Sigma, \text{max}\}$, then $U(\Phi, W, F)$ contains every utility function $u$ such that $\text{ran } u \subseteq W$.

*Proof.* Let $G = \{ (\bigwedge (M \cup \neg M), u(M)) \mid M \subseteq \mathcal{PS} \}$. Clearly $u_{G,F} = u$ and $G \in \mathcal{L}(\Phi, W, \Sigma)$, since each formula in $G$ is a state formula, and only weights in $W$ are used.

Next, we examine the relationship between integer-valued utility functions and noninteger weights:

**Theorem 3.6.2.** If $u_{G,\Sigma} : 2^{\mathcal{PS}} \rightarrow \mathbb{Z}$ and $G$ is minimal and contains any noninteger weight, then $G$ contains at least three noninteger weights.

*Proof.* If $G$ contains exactly one noninteger weight, $w_1$ from $(\varphi_1, w_1)$, then if $M \models \varphi_1$, $u_{G,\Sigma}(M) \notin \mathbb{Z}$, so if there are any, there must be at least two noninteger weights. If $(\varphi_2, w_2) \in G$ and $w_2 \notin \mathbb{Z}$, then by minimality of $G$ we know that $\varphi_1$ and $\varphi_2$ are not equivalent, and so there must be a state $M$ such that either $M \models \varphi_1 \land \neg \varphi_2$ or $M \models \neg \varphi_1 \land \varphi_2$. Hence if $w_1$ and $w_2$ are the sole noninteger weights, $u_{G,\Sigma}(M) \notin \mathbb{Z}$. Therefore, if $G$ contains any noninteger weights, it must contain at least three of them.

While we would like to continue the proof as an induction to show that no finite number of noninteger weights suffices, and hence no minimal representation of an integer-valued utility function will use noninteger weights, we cannot do so, for the following reason: By minimality, we know that the formulas in $G$ are pairwise nonequivalent, and hence this gives us the needed model where exactly
one formula with a noninteger weight is true. But pairwise nonequivalence does not guarantee that for three formulas we can find a model which makes one of the formulas true and the other two false, and hence we do not find the needed model.

We expect that minimality does in fact entail that no noninteger weights are needed to represent integer-valued utility functions. For example, consider the goalbase \{((\top, \frac{1}{2})), ((a, \frac{1}{2})), ((\neg a, \frac{1}{2}))\}, which represents \(u(X) = 1\) using the minimal number of noninteger weights, but is itself clearly not minimal.

## 3.7 Conclusion

In this chapter we have characterized the expressivity of nearly all natural sum and max languages. The sum languages correspond to a wide variety of classes of utility functions, ranging from full (e.g., \(L(\text{forms}, \mathbb{R}, \Sigma)\)) to rather limited (e.g., \(L(\text{atoms}, \mathbb{R}^+, \Sigma)\)) expressivity, with a great many alternatives between (e.g., \(L(\text{k-cubes}, \mathbb{R}, \Sigma)\)). The max languages are bipolar in their expressivity—there are fully-expressive languages (e.g., \(L(\text{cubes}, \mathbb{R}, \max)\)), extremely circumscribed ones (\(L(\text{atoms} + \top, \mathbb{R}^+, \max)\)), and nothing in the middle. (The \(k\)-languages, which occupy the middle for sum, appear to have hardly more expressivity than atoms or literals for max.) There are sum or max languages corresponding to many common classes of utility functions, so it is likely that an appropriately expressive one may be found for any desired application. Additionally, we have demonstrated that some languages, such as \(L(\text{pcubes}, \mathbb{R}, \Sigma)\), \(L(\text{pclauses}, \mathbb{R}, \Sigma)\), and their sublanguages, have unique representations for all utility functions which are representable in them.