Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Free University of Bolzano
Faculty of Computer Science

Answer Set Programming for Multiagent Resource Allocation

Belopeta Mito

Master Thesis in
Computational Logic

Dissertação apresentada na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa para obtenção do grau de Mestre em Lógica Computacional

supervision:
Prof. João Alexandre Leite
Prof. José Júlio Alferes

Lisboa 2008

Abstract

Multiagent resource allocation is a growing area of research at the interface of Economics and Computer Science. It encapsulates many different aspects from resource allocation raging from economical, game theoretical, to purely computational. This work concerns the computational aspect of resource allocation, particularly, the complex combinatorial problems. Besides the extensive theoretical work and besides the raising number of practical applications, many fundamental problems in multiagent resource allocation still lack proper solutions. Here, it shown how a fully declarative and highly expressive logic programming paradigm, Answer Set Programming, can be used for providing general and flexible solutions of these problems in a compact and declarative manner.

Table of content

1. Introduction

Resource allocation is the problem of finding a suitable distribution of resources among several entities. Depending on the type of resources and the entities to which the resources are being distributed, such problem can take various forms relevant to different application domains. Since some of these domains are closely related to Economics, resource allocation has been extensively studied under this scientific discipline. In Economics the focus is on the qualitative aspect of resource allocation (what is a suitable distribution of resources?), while, the computational aspect (how to find a good distribution of resources?) has been addressed in Computer Science. However, both scientific disciplines lack completeness in their studies. A comprehensive approach has to address both the computational and the qualitative aspects of resource allocation. It turns out that the Multi Agent System (MAS) paradigm provides the necessities for the augmentation of such approach. In fact, many problems which appear in MAS can be considered, with certain abstraction, as research allocation problems in which both the qualitative and the computational aspects are relevant.

MAS is a collection of autonomous software entities '*agents*' which act on their behalf in order to achieve the desired performances of the overall systems. In MAS, resources are being distributed among agents, thus the area is referred to as Multi Agent Resource Allocation (MARA). Generally, MARA studies the problem of finding a distribution of resources to agents which satisfies certain quality criteria that depends, in one way or the other, on the agent's preferences (an indication of the agent's satisfaction for a particular allocation). The main issues addressed by MARA are: the representation of agent's preferences, the definition of quality criteria and the procedures for finding a particular distribution of resources (protocols). There has been extensive work done on all these issues with both theoretical and practical results [1]. In essence, several methods have been developed for compact representation of agent's preferences. Furthermore, quality criteria have been defined for the requirements of various applications of MARA and notions like optimal, Pareto-optimal and envy-free distributions have been established. Finally, a number of protocols have been developed for both centralized (a single agent computes the final distribution of resources) and distributed (the final distribution of resources is a result of sequence of steps in each of which some agents agree on a resource exchange) allocation of resources. This includes studies of the agents' behavior under these protocols as well as results on their computational and communicational complexity.

Despite the extensive theoretical work done in MARA which has contributed for better understanding of resource allocation in general, and despite the growing number of applications relevant to MARA (examples are scheduling, strategic planning, industrial procurement, logistics, airport traffic management, public transport, efficient and fair exploration on earth observation satellites, and network routing [1]), the computational aspect of MARA problems have not drawn much the attention of researches in the past years with practical results are still lacking. The reason for this is that finding an allocation of resources which satisfies certain quality criteria is in not at all easy. In fact, as long there are no strict restrictions on the number of resources, most of the problems in MARA are of NP-complete (or higher) complexity and often include additional constraints on the final allocation. Thus, no matter which distribution protocol is used,

some of the agents will need to be equipped with an algorithm, not only powerful enough for solving the problem, but also very flexible in accommodating additional constraints, different quality criteria, and different preference representations.

Consider an example from the e-commerce domain, the well known winner determination problem in combinatorial auctions, defined as follows: a special agent the auctioneer announces a set of resources that it wants to sell and receives multiple bids (from other agents) for subset of those resources. Obviously the goal of the auctioneer in such scenario is to accept the set of mutually exclusive bids (each resource appears at most in one bid) which provides the biggest possible revenue. Since this is a centralized scenario, it is up to the auctioneer to decide to whom to sell the resources and thus to compute the best possible distribution. This amounts to searching all possible combinations of mutual exclusive bids, and the number of such combination is exponential in respect to the number of bids. Moreover, real world auctions are dynamic and often include various constraints. For example, when selling large amounts of resources is common to limit the maximum number of resources allocated to a single agent due to shipping convenience. Auctions can also take different forms in terms of the valid bids, or the treatment of resources (auctioning bottles of wine is much different than auctioning electricity). Accordingly, a perfect auctioneer must deal with complex problems which can take different forms.

Imagine that instead of finding algorithms for solving such problems, the problems can be solved automatically. All that is needed is a specification of what the problem is, rather than how the problem is solved. Something which for the example above look as follows:

- the set of bids are…,
- each bid can be either accepted or not accepted,
- two bids containing the same item can not be accepted,
- the sum of the values of the accepted bids needs to be maximized.

This is known as declarative problem solving or declarative programming, and it has been of special interest to the AI community, in which, there is a constant tendency towards higher abstraction and clear semantics of programs. There are several approaches to declarative programming. Logic Programming is perhaps the most popular and most widely studied because it produces intuitive programs. Accordingly, many Logic Programming paradigms have been developed for both weakly declarative (beside the problem specification, the program also contains control information) and strongly declarative programming (the program contains only the problem specification). The two most significant examples of the latter are logic programming under the well-founded semantics [37] and logic programming under the stable model semantics [29], also referred to as Answer Set Programming (ASP) [33, 34, 35].While the well-founded semantics is not expressive enough for specifying most of the problems in MARA, ASP is a good candidate for declarative and flexible solution to such problems.

ASP is a kind of logic programming oriented to difficult (NP - Hard) search problems. It is “pure” logic in the sense that it has clearly defined semantics, the stable model semantics for normal logic programs [29] (programs which do not contain explicit

negation, for programs with explicit negation, i.e. extended logic programs, the semantics are referred to as answer set semantics, and thus the name ASP). The syntax of ASP is similar to Prolog, however ASP is fully declarative and thus much different. In general, when a problem is to be solved using ASP, it is encoded with a logic program whose answer sets correspond to the solutions of the problem. Because of the full declaratives of ASP, the programs are typically compact, intuitive and easy to come up with. An answer set solver is used for computing the answer sets of a program. Common answer set solvers are Smodels and Clasp for extended logic programs [31,43], and DLV [32] for extended logic programs with disjunction in the head of the rules, all of which include optimization statements (rules for minimizing or maximizing certain value which is depended, in terms of weights, on the literals present in the answer set) for modeling optimization problems in ASP. This is very important since there are many such problems in MARA.

In fact, ASP has been already successfully applied to solving the winner determination problem [39]. The main advantage of this approach, as opposed to other approaches, is that it is fully declarative and it leads to intuitive and flexible (in terms of generalizations and additional constrain) solutions. In fact, the ASP formalization of the winner determination problem directly resembles the declarative specification above. Regarding flexibility, generalizing from single-unit auctions (there is a single unit of each resource, the resource itself) to multi-unit auctions (there are many units of each resource, and agents bids for unit of resources) does not require detailed knowledge of the original program or its structure, while constrains (for example two resources must be allocated to the same agent or set of resources should not be sold under specific price) in most of the cases amount to adding few rules in the program. Also the ASP approach is rather efficient, although not comparable to the specialized algorithms.

Taking this into account, in this dissertation it is investigated whether ASP can be used for solving other problems in MARA, preserving the advantage in declaratives and flexibility. This is motivated by the high expressivity of ASP and its knowledge representational roots. Because of the former, ASP should be able to capture most of the problems in MARA, and because of the latter, ASP seems adequate for compact representation of agents' preferences in most of the methods studied in MARA. Finally, the reasonably efficient implementations of answer set solvers promise efficient solutions, which are likely to improve in the future (with improvements in answer set solving).

The concern is mostly on problems in centralized resource allocation which is more natural for ASP. Distributed protocols are commonly applied to problems in which finding the optimal allocation is infeasible and small improvements on the initial distribution are considered a success. Exceptions are problems in distributed protocols which relay on some complex agent behavior, and which are partly addressed in this dissertation.

1.1 Structure

The dissertation is structured as follows. The following chapter (chapter 2) contains an overview of MARA where its main components are discussed: resources, preference representations, allocations procedures (e.g. negotiations, auctions), collective utility functions (the social welfare) and complexity of some decision problems in MARA. All of the resource allocation problems that are solved later on are covered in this overview. Chapter 3 is an overview of ASP, its syntax and semantics, and the answer set solvers relevant to this dissertation: DLV, Clasp and Smodels. Chapter 4 is the main chapter of the dissertation. It starts with the motives for applying ASP in MARA, followed by the ASP solutions of several MARA problems. The experimental results for the solutions are presented in Chapter 5. Finally the conclusions are given in chapter 6.

1.2 Contributions

There are three main contribution of this dissertation:

- It provides sound and complete solutions for several problems in MARA.
- It provides examples of how problems of different complexity classes can be declaratively solved in ASP, more specifically, from NP-Complete up to Σ^P_2 problems.
- It provides benchmarks for testing answer set solvers.

Regarding centralized allocation of discrete, single unit, indivisible, non sharable, and static resources, solutions are presented for the following problems: Welfare Improvement, Welfare Optimization (both the decision and the optimization version), Pareto-optimality (both the decision and the optimization version), Envy Freeness (both the decision and the optimization version), Pareto-optimality and Envy Freeness combined (both the decision and the optimization version) and Winner determination in combinatorial auctions (for the XOR and the OR language). Regarding the distributed allocation, a solution is presented for the problem of determining the next proposal under the Zeuthen strategy in the simple Monotonic Concession protocol.

In general, all common notions of social welfare are taken into account, as well as different preference representations, including: the bundle form, the k-additive form, weighted goals, and the different orderings of prioritized goals.

Beside the above, the dissertation also provides an overview of both Multiagent Resource Allocation and Answer Set Programming.

2. Multiagent Resource Allocation – Overview

Resource allocation is an area studied by both Computer Science and Economics. While in computer science the concern is mostly of procedural nature, (how to find a particular distribution of resources) in economics the focus is on qualitative issues (what is a good distribution of resources). Thus, in order to comprehensively address the problem of allocating resources an interdisciplinary approach is needed. MARA is such an approach based on the Multi Agent System (MAS) framework. The following is the definition of MARA as given in [1]:

Multi Agent Resource Allocation is the process of distributing a number of items amongst a number of agents.

The definition should be further specified in terms of the following questions: what are the items to be distributed? How are the items being distributed? And why are the items being distributed (what are the objectives of the distribution)? The answers to these questions are not straightforward and are the essence of this chapter.

In MARA the term *resource* is used to refer to the items that are being distributed to agents, although this is not literally true. For example in task oriented domains tasks (not resources) are distributed to agents (the actual resources). The term *allocation* is used for referring to a particular distribution of resources and the term *bundle* is used for referring to a collection of resources.

This chapter focuses on the topics in MARA relevant to this dissertation and is based on the MARA overview in [1]. It is structured as follows. Firstly the different categorizations of resources which can appear in MARA problems are discussed followed by an explanation of the different structures and representations of agents' preferences. The different properties of a potential solution are discussed next, followed by an explanation of the allocation procedures that can be used for allocating resources. Finally, before the conclusion, the complexity of some problems in MARA is stated.

2.1 Resources

Resources play a central role in any resource allocation problem. Depending on the nature of the resources or the way they are treated during allocation there are several categorizations of resources.

The first categorization divides resources to **continuous** (e.g. liquid, gas, energy…) or **discrete** (e.g. fruit, table, chair…). As in other domains the meaning of continuous is infinitely divisibleand the meaning of discrete is not infinitely divisable. A common approach is to treat a continuous resource as discrete by dividing it to a number of small indivisible units, thus to some extant the methods used for discrete MARA can also be applied in the continuous case.

Alternatively, resources can be treated as **indivisible** or **divisable**. Unlike the previous characterization which is a property of the resources themselves, the distinction between indivisible and divisibleis done when the resources are being allocated. For example, a bottle of wine can appear as both divisibleand indivisible resource. If a supplier allocates bottles of wine to super markets, a bottle of wine is indivisible since the supplier can not allocate a half bottle to one market and another half to other. On the other hand, if wine is

to be served (allocated) to people in a restaurant, a bottle is divisible since someone can be given a glass of wine.

Resources can also be **sharable** or **not sharable**. Sharable resources are the ones that can be allocated to more than one agent at once. Such resource can be an internet server which can be used by more than one agent simultaneously. On the other hand a non sharable resource (e.g. chair) can be only allocated to a single agent at a time.

Other categorization is the one of **static** and **non static** resources. Non static resources are either *consumable* i.e. the resource is used up by the actions performed by an agent (example fuel), or *perishable*, i.e. the value of the resource reduces over time (example food). If the resource does not change its properties over time then it is considered static. Although not all resources are static in MARA they are often assumed to be. The rational behind this is that even when resources are consumable or perishable they can be assumed to be static for the time needed for finding an allocation.

Resources can also be divided to **single-unit** and **multi-unit** resources. In the multi-unit case many items of a specific type are treated as one resource. For example a multi-unit resource wine can represent many bottles of wine that are to be allocated to agents. In the single-unit case every unit is a distinct resource with a unique name. It is obvious that a multi-unit resource can be transformed to many single-unit resources by giving each unit a unique name, i.e. consider it as a separate resource. Accordingly, the distinction between **single-unit** and **multi-unit** is only a matter of representation.

Task oriented domains with certain abstraction can also be considered as a resource allocation problems where tasks are resources on which the agent give negative utility. However, tasks have typically a set of constrains associated with them (e.g. one task should be done before other) which is not the case in a common resource allocation problem.

The primary concern in this dissertation is the allocation of **discrete**, **indivisible**, **non sharable** and **static** resources. Note that tasks can also be considered as such.

2.2 Preferences

Agents' preferences represent the relative or the absolute agent's satisfaction with a particular allocation and are necessary for defining the quality aspects of a solution to a research allocation problem. They can be either specified over alternative allocations or over alternative bundles. This does not make a difference in most of the formalizations that follow. However, in most parts of the dissertation, it is assumed that preferences are defined over alternative bundles. This is sufficient for problems that are free of allocative externalities, i.e. agent's preferences only depend on the resources allocated to it and not on the resources allocated to the other agents. In general, agents' preferences are defined in terms of structure and representation.

2.2.1 Preference Structure

A structure represents the preferences of an agent over a possible set of alternatives. There are four preference structures.

The **cardinal** preference structure consists of a utility function $u: X \rightarrow Val$ where X represent the set of alternative bundles, and Val is a set of numeric values or a totally ordered scale of qualitative values (e.g. very bad, bad, average, good….). The former is called *quantitative* while the latter alternatives *qualitative* preference structure.

The **ordinal** preference structure consist of binary relations between alternatives $x \preceq y$ denoting that the bundle x is at least as preferred as the bundle y. $x \prec y$ denotes strict preference. The ordinal preferences are not as expressive as the cardinal preferences in the sense that they cannot express preference intensity. On the other hand, each cardinal preference induces an ordinal preference.

A **binary** preference structure simply partitions the set of alternatives into two sets: one representing good and one representing bad alternatives. This kind of structure can be considered as special case of cardinal or ordinal structures.

A **fuzzy** preference structure is a fuzzy relation over the set of alternatives X, i.e. a function $f: X \times X \rightarrow [0,1]$ defining the degree to which one alternative is preferred over the other. Since such preferences have not been extensively studied and are rarely used in practice, they are not further considered in the dissertation.

2.2.2 Representing quantitative preference

Preferences need to be properly represented when used in practice. The problem is that the number of alternatives is usually exponential in respect to the number of overall resources and thus an *explicit* representation (listing all alternatives with their associated utility values for cardinal preference, or the full $\preceq$ relation for ordinal preferences) will be exponentially large. For this reason, MARA is also concerned with languages for more succinct preference representation. In what follows some of these languages are explained, first for the quantitative and then for the ordinal preferences. The common bidding languages used in combinatorial auctions are explained at the end of the section.

The **bundle form** is the most basic approach for representing quantitative preferences. It simply enumerates all bundles for which the agent has a non-zero utility. Thus, a bundle representation is a set of pairs $\langle R,u(R)\rangle$ with R being the bundles for which $u(R) \neq 0$. For example, if the set of resources is $\{a, b, c\}$ and the preference of an agent are $\{\langle\{a\},2\rangle,\langle\{b\},1\rangle,\langle\{a,b\},3\rangle,\langle\{a,b,c\},3\rangle\}$, aside for the explicit utilities, a 0 utility is assigned to the bundles $\{c\},\{b,c\}$,and $\{a,c\}$. It is obvious that the bundle form is fully expressive in terms of representing any possible utility function.

Another approach is the ***k*-additive form** [2] which exploits regularities in a function in order to build efficiently computable succinct representation. Given $k \in \mathrm{N}$, a utility function u is k-additive if and only if for every set of resources T of size less or equal to k there exist a coefficient α_T such that:

$$u(R) = \sum_{T \subseteq R} \alpha_T$$

If a utility function is represented in terms of such coefficient then it is given in the k-additive form.

Intuitively the coefficient α_T represents the additional gain of utility when having all the resources in T together aside of the utilities associated to any of its proper subsets. Accordingly, if an agent has a utility 2 for the bundle $\{a\}$, 2 for the bundle $\{b\}$, and 5 for the bundle $\{a, b\}$, the synergetic value of having the resources a and b together is 1 (the utility of the whole set, minus the utility of its proper subsets), and thus $\alpha_{\{a, b\}}=1$, $\alpha_{\{a\}}=2$ and $\alpha_{\{a\}}=2$.

The k-additive preference representation is fully expressive only when k is equal to the overall number of resources. However in many application domains the utility functions can be assumed k-additive only with a relatively small value for k.

Straight line programs (**SLPs**) [3] can also be used for representing agents' preferences. They are acyclic graphs with two types of vertexes: inputs, which have in-degree 0, and gates with in-degree 2. The gates with out-degree 0 are the outputs. In addition to the graph structure SLPs are defined with a binary operation associated to each gate vertex. A topological labeling of the vertexes assigns a values $\lambda(x_i) = i$, to each input vertex $x_0 \ldots x_i$, and a value $\lambda(v) > \max\{\lambda(w_1), \lambda(w_2)\}$ to each vertex v given its in-edges $\langle w_1, v\rangle$ and $\langle w_2, v\rangle$.

An SLP C, consisting of the inputs $\langle x_1, x_2 \ldots x_m\rangle$ and p gates, t of which $\langle s_0, s_1 \ldots s_{t-1}\rangle$ output gates, computes the result by executing a program consisting of exactly $m+p$ lines each computing a single bit value $res(i)$ for each vertex i. Given an input vector $\langle \alpha_1, \alpha_2 \ldots \alpha_m\rangle$, $res(i) = \alpha_i$ for the lines computing the res for the inputs vertexes, i.e. $1 \leq i \leq m$, and $res(i) = res(j_1) \theta res(j_2)$ for $m + 1 \leq i \leq m + p$, where θ is the binary operation associated to the ith vertex with the corresponding input vertexes j_1 and j_2. The output of the SLP C for a input vector $\underline{\alpha}$ is then: $val(C, \underline{\alpha}) = \sum_{i=0}^{t-1} res(s_i) * 2^i$. Accordingly, given a mapping from an alternative S to its characteristic m-bit input vectors $\underline{\alpha}(S)$, the utility of S, $u(S) = val(C, \underline{\alpha}(S))$.

A SLP provide an alternative for representing a utility function of the form u: $2^R \rightarrow$ N (where R is the set of resources, and N is the set of natural numbers, with possible extensions to integers and reals), and like the bundle form, it's fully expressive. Note that, for some functions the SLPs representation is exponentially smaller than the one given in the bundle form. Consider a utility function which returns 1 for odd number of resources, i.e. $|S|$ is odd, and 0 otherwise. The number of lines needed by the bundle representation is equivalent to the number of bundles with odd number of resources 2^{m-1}(where m is the number of resources) which is exponential. The same can be linearly represented by an SLP consisting of $2m$-1ines, where there are m inputs (each input corresponds to a resource and it has value 1 if the resource belongs to S, and 0 otherwise) and m-1 gates (one of which is an output) each computing XOR. For example, the SLP in figure 2.1 can be used for computing the function when the number of overall resources is 4:

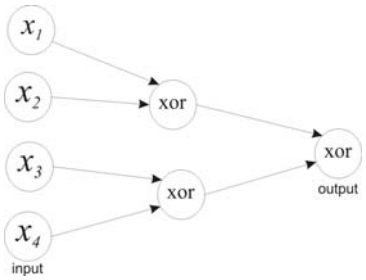

Figure 2.1. SLP example when overall number of resources is 4:

Similarly, when the overall number of resources is 5, the SLP of figure 2.2 can be used.

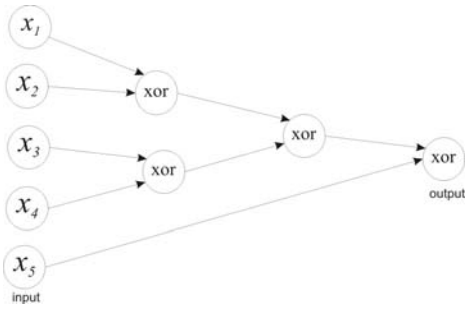

Figure 2.2. SLP example when overall number of resources is 5:

An alternative approach for compact representation of preferences is the explicit use of logic languages [4]. In a **logic-based** preference representation each resource *r* correspond to a propositional formula p_r that is true if the agent owns that resource and false otherwise. Here every bundle corresponds to a model. Agents then express their preferences in terms of propositional formulas (goals) that they want to be satisfied. The simplest such representation is to have a single propositional formula representing the goal *G*. The agent's utility is 1 if the received bundle *R* satisfies the goal *G* ($R \models G$), and 0 otherwise ($R \not\models G$). A possible refinement is to have a set of goals $GB = \{G_1,\ldots,G_n\}$ and then count the number of satisfied goals in *R*. Further refinement is to associate a weight α_i to each goal (i.e. $GB = \{\langle G_1,\alpha_1\rangle, \ldots, \langle G_n,\alpha_n\rangle\}$). The weight represents the importance of a goal and is used to associate penalties to the bundles that do not satisfy that goal.

$$p_{GB}(R) \;=\; \sum\{\alpha_i \mid R \not\models G_i\}$$

The utility of the bundle is the reverse of its penalty:

$$u_{GB}(R) = -p_{GB}(R)$$

Such representation is called **weighted goals**. Similar to the k-additive representation, this representation can express the synergetic value of having two resources together, just instead of coefficients it uses weights. Thus the example used above for the k-additive representation consists of the following database of weighted goals $GB = \{\langle has(a),-2\rangle, \langle has(b),-2\rangle, \langle has(a) \wedge has(b),-1\rangle\}$ (where *has(r)* is true if the resource *r* is allocated to the agent whose utilities are represented). The final utilities in this case are -3 for the bundle $\{a\}$, -3 for the bundle $\{b\}$, and 0 for the bundle $\{a, b\}$. This is equivalent to the utilities in the k-additive case mentioned before, only instead of from 0 to 5 the value ranges from 0 to -5. It is enough to use only conjunction in the weighted goals to express any utility function, having in mind that the k-additive representation is fully expressive (in the

general case) and conjunction is sufficient for simulating it with weighted goals. Moreover, this representation can express more complex goals (and relationships between resources), for example the goal "*has(a)* → *has(b)*" can be used for expressing the statement: the agent wants resource *b* whenever the resource *a* is allocated to it.

2.2.3 Representing of ordinal preference

When representing ordinal preferences, logic based languages play a central role. In essence, a very similar representation to the one of weighted goals is used for representing ordinal preferences. The only difference is that instead of weights this representation uses priority relation on goals and thus it is called **prioritized goals**. In the case when the priority relation is complete, which is a usual assumption, the priorities on the formulas can be represented with a function *r* from integers to integers. In such case, a goal base is a set of goals with an associated function $GB = \langle \{G_1,...,G_n\}, r \rangle$ where if $r(i) = j$, j is the rank of the goal G_i. By convention a smaller rank means a higher priority. There are several methods how to further extend this representation in a preference relation $\preceq$ over alternatives. The following are the most common:

- *best-out ordering*: An allocation *R* is best out preferred over an allocation *R'* if the goal(s) with the smallest rank (highest priority) among the unsatisfied goals in *R* has greater rank compared to the corresponding such(s) goal in *R*'. Formally, $R \preceq^{bo}_{GB} R'$ iff $min\{r(i) \mid R \not\models G_i\} \leq min\{ r(i) \mid R' \not\models G_i\}$ such
- *descrimin ordering*: An allocation *R* is descrimin preferred over an allocation *R'* if the goal(s) with the smallest rank among the goals satisfied by *R* and not satisfied by *R*' has a greater rank compared to the corresponding such goal(s) in *R*'. Formally, let $d(R, R') = \min\{r(i) \mid R \not\models G_i \wedge R' \models G_i\}$. $R \preceq^{dis}_{GB} R'$ iff $d(R, R') < d(R', R)$ or $\{G_i \mid R \models G_i\} = \{ G_i \mid R' \models G_i\}$.
- *leximin ordering*: An allocation *R* is leximin preferred over an allocation *R'* if it satisfies more goals of certain rank *i*, and an equal number of goals for all ranks which are smaller than *i*. Formally, let $d_k(R) = |\{ G_i \mid R \models G_i \wedge r(i) = k\}|$. $R \prec^{lex}_{GB} R'$ iff there exist a *k* such that $d_k(R) < d_k(R')$ and $\forall j < k$, $d_j(R) = d_j(R')$. $R \preceq^{lex}_{GB} R'$ iff $R \prec^{lex}_{GB} R'$ or $\forall j$, $d_j(R) = d_j(R')$.

Note that $\preceq^{bo}_{GB}$ and $\preceq^{lex}_{GB}$ are total orderings while $\preceq^{dis}_{GB}$ is a partial ordering. Also $R \prec^{bo}_{GB} R'$ entails $R \prec^{dis}_{GB} R'$ entails $R \prec^{lex}_{GB} R'$.

Consider a simple example with three resources *a*, *b*, *c* and the following goal base GB = $\{\langle has(a),1\rangle, \langle has(b),2\rangle, \langle has(c),2\rangle, \langle has(a) \wedge has(b),3\rangle\}$. The following are the full preference relations for the best-out, descrimin and leximin orderings repetitively:

- $\{a,b,c\} \prec^{bo}_{GB} \{a,b\} \preceq^{bo}_{GB} \{a,c\} \preceq^{bo}_{GB} \{a\} \prec^{bo}_{GB} \{b,c\} \prec^{bo}_{GB} \{b\} \preceq^{bo}_{GB} \{c\} \prec^{bo}_{GB} \{\}$;
- $\{a,b,c\} \prec^{dis}_{GB} \{b,c\}, \{a,c\}, \{a,b\}, \{a\}, \{b\}, \{c\}, \{\}$,
  $\{a,b\} \prec^{dis}_{GB} \{a,c\}, \{b,c\}, \{a\}, \{b\}, \{\}$
  $\{a,c\} \prec^{dis}_{GB} \{b,c\}, \{b\}, \{c\}, \{a\}, \{\}$
  $\{a\} \prec^{dis}_{GB} \{b,c\}, \{c\}, \{b\}, \{\}$,
  $\{b,c\} \prec^{dis}_{GB} \{c\}, \{b\}, \{\}$
  $\{b\} \prec^{dis}_{GB} \{\}$,

$\{c\} \prec^{dis}_{GB} \{\}$,

- $\{a,b,c\} \prec^{lex}_{GB} \{a,b\} \prec^{lex}_{GB} \{a,c\} \prec^{lex}_{GB} \{a\} \prec^{lex}_{GB} \{b,c\} \prec^{lex}_{GB} \{b\} \preceq^{lex}_{GB} \{c\} \preceq^{lex}_{GB} \{\}$;

It is obvious that the leximin is the most detailed relation, while the descrimin leaves incomparable bundles such as $\{b\}$ and $\{c\}$.

Another logical approach for representing ordinal preferences is the so called **ceteris paribus preferences**. This language represents preferences by statements of the form "given all other things being equal, I prefer these alternatives over other ones". Formally a ceteris paribus statement C: $G > G'$ [V], where C, G and G' are propositional formulas and V is a set of propositional variables, means "when C is true, all irrelevant things being equal, I prefer $G \wedge \neg G'$ to $G' \wedge \neg G$. The preference relation is then the transitive closure of the preference relations induced from such statements. A special case of the ceteris paribus preferences is the one of binary CP-nets obtained by imposing the following three restrictions:

- Goals G' and G are literals speaking about the same propositional variable.
- The variables mentioned in the context C of a preference statement about variable p must belong to a fixed set, called the parents of p.
- For each variable p and each possible assignment π of the parents of p, there is one and only one preference statement C: $p > \neg p$ or C: $\neg p > p$ such that $\pi \models$ C.

There are various extensions of the binary CP-nets with greater expressive power. Such examples are TCP-nets [5] (with dominance relation between variables) and UCP-nets [6] (based on general additive independence).

Unfortunately all the presented languages for representing preferences (excluding the bidding languages which are a separate class and are reviewed next) suffer from deficiencies that limit their use. In essence, it is difficult to extract numerical preferences from agents for the quantitative preferences representation, the prioritized goals lack expressivity, while the ceteris paribus preferences have high complexity and leave many incomparable pairs.

2.2.3 Bidding languages

Bids are special type of numeric preferences that are used in the auction allocation procedure. They are typically represented as a combination of atomic bids in the form $\langle R,p \rangle$, where p is the amount an agent is prepared to pay for a bundle R. There are several languages for expressing bids. The most common are the **OR** and the **XOR** language.

The OR language is probably the most widely used. Here the value of a bundle is the maximum amount that can be obtained when summing disjoint bids for subsets of that bundle. Thus "$\langle\{a\},2\rangle$ OR $\langle\{b\},2\rangle$ OR $\langle\{c\},1\rangle$ OR $\langle\{a, b\},1\rangle$" means that the agent is prepared to pay 2 for a alone, 5 for a and b and 6 for the full set. It is obvious that the OR language is not fully expressive since it can not express sub-additive utility functions. For example, in the previous case it can not express that the agent is only prepared to pay 4 for the full set.

In the XOR language[7] the atomic bids are assumed to be mutually exclusive. The value of a bundle in this case is simply the highest value offered for any of its subsets. Unlike the OR language, the XOR is fully expressive. However it is in general far less compact when compared to the OR language.

Since the OR language is considered to be a simple and natural way of expressing bids there have been several attempts to increase its expressivity. Particular examples are the XOR-of-OR and the OR-of-XOR bidding language which are both mixtures of the OR and XOR languages [8]. For example, in the XOR-of-OR language a statement like "$\langle\{a\},2\rangle$ OR $\langle\{b\},2\rangle$ XOR $\langle\{c, a\},3\rangle$ OR $\langle\{c, b\},3\rangle$ XOR $\langle\{c\},1\rangle$ " will have the meaning that the agent can pay 2 for $\{a\}$, 2 for $\{b\}$, 1 for $\{c\}$, 4 for $\{a, b\}$, 3 for $\{ c, a \}$, 3 for $\{c, b\}$, 6 for the full set, and 0 for any other bundle. Both the XOR-of-OR and the OR-of-XOR languages are fully expressive since they are a super-class of the XOR language.

An interesting alternative is also the OR* language [9] which uses dummy (fake) resources in order to simulate XOR bids in the OR language. The previous example (for the OR language) can be modified in a way that it is stated that the agent is prepared to pay only 4 for the full set. This is done by adding a fake resource f in the bundles $\{c\}$ and $\{a, b\}$ thus $\langle\{c, f\},1\rangle$ and $\langle\{a, b, f\},5\rangle$, and adding additional pair $\langle\{a, b, c\},4\rangle$.

2.3 Qualitative issues

In general the quality of a solution to a resource allocation problem is defined by a metric that depends, in one way or another, on the preferences of the agent. The aggregation of the individual preferences in such metric is often modeled using the notion of *social welfare* as studied in Welfare Economics and Social Choice Theory. Depending on the system's goals, different social welfare functions can be used for measuring the quality of an allocation [10,11]. Some of these functions are presented next for both the cardinal and the ordinal preference representations, together with examples of applications where they can be applied. Before, the notation used throughout this section is stated.

$A = \{1,\ldots,n\}$ is a set of n agents. Depending on whether ordinal or cardinal preferences are used, each agent i is equipped either with a preference relation $\preceq_i$ or a utility function u_i. An allocation of resources P is a mapping from agents to bundles, and thus $P(i)$ denotes the bundle held by the agent i. If not stated differently, it is irrelevant whether preferences are defined over alternative bundles or alternative allocations. Thus $P \preceq_i Q$ can be considered as an abbreviation for $P(i) \preceq_i Q(i)$ and $u_i(P)$ for $u_i(P(i))$.

The most fundamental quality criterion for a solution is **Pareto-optimality**. This concept is defined as follows:

An allocation P is *Pareto dominated* by another allocation Q if and only if:

- $P \preceq_i Q$ for all agents $i \in A$, and
- $P \prec_i Q$ for at least on agent $i \in A$

An allocation is *Pareto-optimal* if and only if it is not Pareto dominated by any other allocation.

The concept of Pareto-optimality is purely ordinal and it induces a partial order of Pareto-dominance over the possible allocations.

Envy freeness is also a purely ordinal concept. An allocation is envy-free when every agent is at least as satisfied with the bundle allocated to it as with the bundles allocated to other agents. More formally, an allocation P is envy-free if and only if $P(j) \preceq_i P(i)$ holds for all agents i and j. Envy freeness can only be defined when the preferences are defined over alternative bundles (rather than over alternative allocations).

If all resources are to be allocated, then an envy-free allocation does not always exist: for example consider an allocation of a single resource desired by all agents, an envy free allocation will not exist. Even when not all resources are required to be allocated, the existence of a solution that is both envy-free and Pareto-optimal is not guaranteed. In such cases a reasonable approach is to look either for a Pareto-optimal solution with the least number of envious agents, or for a Pareto-optimal solution which minimizes the average degree of envy (the distance to the most envied competitor) of all envious agents. The latter requires the agents to have quantitative preferences, i.e. utility functions.

When the agents have their preferences expressed with a utility function every allocation P gives rise to the utility vector $\langle u_1(P), \ldots, u_n(P) \rangle$. A **collective utility function (CUF)** is a mapping from such a vector to a numeric value. Since every allocation determines a utility vector, CUF can also be considered as a function from allocations to numeric values which represent the social welfare of the corresponding allocation. Thus, given a CUF sw an allocation Q is socially preferred to allocation P if and only if $sw(P) < sw(Q)$. There are several CUFs which are useful for different applications of MARA.

The most common CUF is the **utilitarian social welfare** defined as the sum of the individual utilities of the agents. Formally:

$$sw_u(P) \quad = \quad \sum_{i \in \mathcal{A}} u_i(P)$$

This type of CUF is useful for the overall and average profit in different e-commerce applications.

If ensuring fairness is a priority, then the proper CUF is **egalitarian social welfare** defined by the utility of the agent that is currently worst off. Formally:

$$sw_e(P) \quad = \quad \min\{u_i(P) \mid i \in \mathcal{A}\}$$

There are several MARA applications where fair division is a requirement. For example a system that needs to satisfy the minimum needs of a large number of customers.

The opposite of the above CUF is the **elitist social welfare** defined by the utility of the agent that is currently best off. Formally:

$$sw_{el}(P) \quad = \quad \max\{u_i(P) \mid i \in \mathcal{A}\}$$

These can be used in systems where only one agent is required to achieve its goals.

The **Nash product** is defined as the product of the individual utilities. Formally:

$$sw_N(P) \quad = \quad \prod_{i \in \mathcal{A}} u_i(P)$$

This CUF is a compromise between the utilitarian and egalitarian social welfare in the sense that it favors overall utility and reduces inequalities between agents. Observe also that it only leads to a meaningful metric if all of the agent's utilities are positive.

The egalitarian and elitist social welfare are specific cases of a more general CUF, the **k-rank dictator**. This CUF evaluates social welfare to be equal to the k-th smallest utility assigned to an agent. Thus, elitist social welfare is the n-rank dictator, where n is the number of agents, and the egalitarian social welfare is the 1-rank dictator. A special case is the *median-rank dictator* where $k = \lceil n/2 \rceil$. This is used in applications in which of particular importance is the performance of an agent that does better than half of the agents, and worse than the other half. In general, it has similar effect as the utilitarian social welfare but it is not affected by extreme utilities. For example, if utilitarian social welfare is used to evaluate the performance of students in a class, the overall evaluation will be effected by the exceptionally bad or the exceptionally good students. The median-rank dictator does not suffer from such deficiencies and thus can be considered more appropriate for this case.

The k-rank dictator can be further used for defining the **leximin** social welfare ordering. In this ordering allocations are ordered by first comparing the utilities of the least satisfied agents (1-rank), in case these coincide, comparing the utilities of the second least satisfied agents (2-rank), in case these coincide also, then the third (3-rank) and so on. Stated differently, the leximin ordering is a lexicographical ordering over the ordered utility vectors derived from the allocations. As well as the egalitarian social welfare, it favors fair division of resources. An allocation is *leximin optimal* if and only if it is not leximin-preferred by any other allocation.

Most of the above presented methods require the agents' preferences to be inter-comparable, which is not always the case. Thus, it is necessary to normalize the preferences prior to using them as an input for the CUF. A way to do this is to start with the initial allocation P_0, then for every other allocation P that Pareto-dominates P_0 use the utility gains $u_i(P)$ - $u_i(P_0)$ rather than the utilities $u_i(P)$ as an input to the CUF or the leximin ordering.

An alternative is to evaluate an agent's utility gains relative to the best possible gains. Formally, let Adm be the set of all admissible allocations, the maximum individual utility $u_i^{\max}$ of an agent i is defined as follows:

$$u_i^{\max} = \max\{u_i(P) \mid P \in Adm\}$$

The normalized individual utility $u_i'(P)$ for an allocation P is defined as follows:

$$u_i'(P) = u_i(P) / u_i^{\max}$$

Observe that the maximum normalized utility is 1 for all agents.

The leximin optimal solution in respect to normalized utilities is also known as *Kalai-Smorodinsky* solution [12]

2.4 Allocation Procedures

An allocation procedure is a procedure for finding a suitable distribution of resources to agents. In general, when studying allocation procedures there are three issues which are of particular concern:

- *protocols*- the actual procedure (the sequence of steps that comprise the procedure and the rules of communication)
- *strategies*- the behavior of agents for best exploring a particular protocol (provides a feedback to the previous issue, for example a protocol should prevent agents from gaining bigger revenue by cheating)
- *algorithms* - solving the computational problems of the first two.

This section is concerned with the first two of these issues, while the last one will be addressed throughout the dissertation.

In general there two types of allocation **protocols**, *centralized* and *distributed*. In the centralized protocols a single entity is used for computing a suitable allocation. In the distributed case the final allocation is a result of a sequence of steps in which some agents agree on exchanging resources. It is fundamental to choose the convenient protocol for a particular problem since the two approaches offer different benefits and carry different drawbacks. The centralized protocols require simple communication and have fast and reliable algorithms for their computational aspects [7,9]. On the other hand, the distributed protocols demand a lot of communication as well as computation if optimal allocations are to be found. Thus, the latter approach is more appropriate in cases where finding the optimal allocation is computationally infeasible and small improvements of the original one are considered success.

The most common examples of the two approaches are discussed next: the auction protocols for the centralized case; and negotiation protocols for the distributed case.

2.4.1 Auctions

In general, a centralized protocol consist of the following basic steps involving two roles: a special agent enhanced with a mechanism for computing allocations, and a set of agents to which resources will be allocated (possibly including the special agent):

- *Announcement:* The special agent announces (informs the other agents) the set of resources to be allocated
- *Reporting Preferences:* The other agents report their preferences to the special agent.
- *Computing:* The special agent computes a particular allocation (according to the algorithm it uses for finding an allocation)
- *Allocation:* The special agent assigns the resources accordingly (and receives a payoff if any).

A particular variant of this protocol with its roots in economics are **auctions** [13,14,15,16,17]. In auctions the special agent is called auctioneer, and preferences are

expressed in terms of bids (section 1.3). Depending on the privacy of bids, there are public (*open-cry*) and private (*sealed bid*) auctions. In public auctions the bids are given one at the time in either acceding (English auction) or descending (Dutch auction) order. In both cases a computational step is not required since the former finishes when no more bids are received (a possible time limit) with the resources being allocated to the last bidder, while the latter finishes when one bid is accepted with the resources being allocate to the owner of that bid. On the other hand, sealed bid auctions are much more interesting in computational sense. The bidding in these auctions is done in one round in which all interested agents report their bids. The auctioneer uses this information for finding the combination of mutually acceptable bids that lead to the highest revenue. This amounts to solving the winner determination problem, a well known NP-hard problem.

Note that agents are not restricted to bidding their true preferences, rather they make a bid that they believe to best serve their interest (since agents are by definition rational). This motivated researches to come up with an auction protocol in which the best interest of the agents is to bid truthfully. Such example is the Vickery auction which is a single round sealed bid auction in which the resources are allocated to the winner for the prize (the auctioneer revenue) of the second highest bid for the same resources. This ensures that the agents do not have an incentive to lie about their preferences.

2.4.2 Negotiations

The most common method for distributed allocation of resources is **negotiations**. Such protocols proceed in sequence of negotiation steps each of which ends in a contract among agents for an exchange of resources.

A rational agent will only agree on a contract from which it benefits. Thus, in order to increase the number of multi-beneficial deals, negotiations typically incorporate side payments (*pay-off*). A value linear to the value of the preferences which an agent can pay in order to compensate for the others agent loss of utility in the negotiations. This is very important if optimal allocations are to be found. For example, consider two-agent A and B and single resource for which the agents have different preference values, A has 2 and B has 5. The optimal allocation in this case is to allocate the resource to the agent B (the one which has greater preference), thus if the resource is initially owned by agent A the agents should reach an agreement for an exchange of the resource. This can not be achieved by a simple exchange since B does not have anything to offer to A, and A is better off with the resource than without it. The pay-off is a mechanism for overcoming this by allowing B to pay agent A as much as it evaluates the resource, i.e. 5. Since the payoff is a direct gain of utility agent A will have a utility 5 if he decides to exchange the resource, and considering that its utility for the initial allocation is 2, it will have an incentive to do the exchange.

When many agents are allowed to negotiate at once, the negotiations are called *multilateral*. Alternatively, if they are restricted to be between two agents, the negotiations are called *bilateral*. An interesting fact is that only multilateral treading can guarantee an overall optimal allocation . It is easy to come up with an example of a non-optimal allocation in which a rational bilateral deal does not exist (consider three agent A, B and C with the following preferences $\{\langle A,\{a, b\},5\rangle,\langle B,\{a\},1\rangle,\langle C,\{b\},1\rangle\}$, and a starting allocation in which *a* is allocated to B and *b* is allocated to C). Another result

states that in multilateral negotiations with side payments any sequence of multibenifical deals will lead to an overall optimal allocation [18], while any sequence of multibenifical deals without side payments leads to an allocation which is Pareto-optimal [19].

What follow are two examples of negotiation protocols, the Contract-net (multilateral trading of single resource) and the Monotonic Concession protocol (bilateral trading of bundles).

Contract-net [20] is perhaps the most popular negotiation protocol. It consists of the following four phases involving two roles, the manager and the bidders:

- *Announcement phase:* The manager advertises the (single) resource to a number of agents (the bidders).
- *Bidding phase:* The bidders send their proposals to the manager.
- *Assignment phase:* The manager elects the best bid and assigns the resource accordingly.
- *Confirmation phase:* The elected bidder confirms its intention to obtain the resource.

The contract-net protocol is one-to-many in the sense that there is only one manager and many bidders. However it leads to a one to one single item contract between the manager and the elected bidder. There are several extensions of this protocol. For example, *TRACONER* [21] allows exchanges of bundles as opposed to single resources. Golfarelli [22] has proposed another extension where instead of monetary payment agents bid resources that they are willing to exchange. There is also a version of the contract net where the bidders exchange constraints between them prior to making their bids [23]. *Concurrent contract-net* [24] is yet another variant where a pre-bidding and a pre-assignment phase are added before the bidding and the assignment phase in the standard Contract-net.

An alternative protocol for negotiation is the **Monotonic Concession** protocol [25]. This protocol is designed for negotiations between two agents only, i.e. bilateral trading. It consists of a sequence of rounds where each agent makes a proposal within the deal space that is represented with some real or integer valued dimension, the utility. The protocol starts with simultaneous proposals of the two agents. If the proposals overlap a deal is reached; else, the agents proceed in a next around in which each agent can propose the same deal as previously or propose a deal which is closer to the one of the opponent (concede). If in a given round none of the agents concede the negotiations finish with a conflict deal.

The main decisions that an agent needs to make while it participates in this protocol is whether to concede in a given round, and if so, how much. There are various strategies developed for deciding this. One which is of special interest is the *Zeuthen* [26] strategy. According to this strategy the agent which is less willing to risk conflict (the difference between its loss of utility in fully conceding and accepting the current offer of the opponent, relative to its loss of utility in the case of conflict deal) needs to concede just enough to change the balance of the risk. When both agents use this strategy the

negotiations end with a deal which is Pareto-optimal. However it is necessary that the agents know both their own preferences and the preferences of their opponent.

2.5 Complexity Results

There are two concepts of complexity studied by MARA: *communicational complexity* and *computational complexity*. This section focuses only on the latter since here the primary concern is finding solutions to computational problems in MARA, irrelevant to the communication (number of message exchanges between agents) necessary for the computation. Since the computational complexity of a decision problem in MARA is dependent on the preference representation used, complexity results are listed for the three different representation forms discussed earlier (section 1.2): the bundle form, the k-additive (with particular interest in 2-aditive) form and the SLP form.

For the decision problems below, the following definitions are used:

A *resource allocation setting* is a triple $\langle A,R,U\rangle$ where:

$A = \{1,2,\ldots,n\}$ is a set of n agents.
$R = \{r_1,r_2,\ldots,r_m\}$ is a set of m resources.
$U = \{u_1,u_2,\ldots,u_n\}$ is a set of utility functions $u_i : 2^R \rightarrow Q$ for the agent $i \in A$, where Q is the set of real numbers.

Every resource in R is assumed to be indivisible and non-sharable. An *allocation* is a mapping P: $A \rightarrow 2^R$ from the agents in A to a subset of the resources R, where $P(i) \cap P(j) = \varnothing$, for any two agents $i \neq j$. The set of all allocations of resources to agents is denoted by $Al_{n,m}$ because there are n choices of agents for each of the m resources (i.e. $|Al_{n,m}| = n^m$).

- **Welfare Optimization (WO)**
  - Instance : $\langle A,R,U\rangle$; $K \in Q$ (the set of real numbers),
  - Question: $\exists P \in Al_{n,m} : sw(P) \geq K$
- **Welfare Improvement (WI)**
  - Instance : $\langle A,R,U\rangle$; $P \in Al_{n,m}$,
  - Question: $\exists T \in Al_{n,m} : sw(T) > sw(P)$

Both the WI and WO are NP-complete for the representation of utility functions in the bundle form, the SLP form and for the 2-additive utility functions [2, 3]. This results hold even in a two agent setting. The problems remain NP-complete even if the utility functions of both agents are monotonic.

The decision problems concerning the qualitative measures Pareto-optimality and envy-freeness are the following:

- **Pareto-optimality(PO)**
  - Instance : $\langle A,R,U\rangle : P \in Al_{n,m}$,
  - Question: is P Pareto-optimal?
- **Envy-Freeness(EF)**

- Instance : $\langle A,R,U \rangle$,
- Question: $\exists P \in Al_{n,m} : P$ is envy free?

PO is coNP-complete for both the SLP form and the 2-aditive utility functions [2, 3]. Same as previously, the results also hold in a 2-agents setting, and in the SLP case also when monotonic utility functions are used.

EF is NP-complete even in the 2-agents case for both the SLP form and for concise logic-based descriptions of agent preferences [27]. The complexity results for the PO and EF combined (EEF efficient envy free) vary from NP-complete to Σ^P_2 –complete, depending on the restrictions placed on the preference relations [28].

2.6 Summary

This overview induces a wide range of problems that are a particular instantiations of the general problem of finding an allocation satisfying certain criteria. Examples are finding Pareto-optimal, envy free, or optimal solutions to a resource allocation problems (with centralized allocation procedure), as well as, determining the winner in combinatorial auction (for the different betting languages) and determining the next proposal under some complex strategy in negotiations. With the exception of combinatorial auctions, most of the problems still lack practical solutions. These problems are the main focus of the dissertation and will be addressed in the chapters which follow.

Some of the aspects of MARA, especially in negotiations and agents strategies, were not covered in the overview. This is not because ASP can not be used for solving problems in those areas. In fact the problem of finding a sequence of deals (the result of a negotiation cycle between agents) that lead to an optimal allocation of resources is NP-complete, and thus it can be expressed in ASP. However the focus in this dissertation is on preference representation and finding solutions in centralized allocation of single unit, indivisible and not sharable resources.

3. Answer Set Programming - Overview

Answer Set Programming (ASP) is a kind of logic programming based on the stable model (answer set) semantics of logic programs. Unlike other logic programming paradigms (e.g. Prolog), ASP is fully declarative and nonmonotonic thus competent for knowledge representation, reasoning and declarative problem solving. It is a result of the interaction between two research directions in the field of Artificial Intelligence (AI): *logic programming* (more specifically semantics of logic programs with default negation), which led to the definition of the answer set semantics in 1988 [29]; and the application of satisfiability (SAT) solvers to search problems, which motivated the way problems are solved using ASP [30]. Soon after the successful implementation of answer set solvers in late 1990's (e.g. Smodels[31], Clasp[43], and DLV [32]), ASP was recognized as a distinct logic programming formalism in 1999[33,34, 35].

Aside to the research that led to the development of ASP, there has been a substantial work done within the AI community on nonmonotonic formalisms for knowledge representation and reasoning. In the late 1970s and early 1980s several nonmonotonic logics were developed such are circumscription, default logic and auto epistemic logic. An important result that relates this work with ASP is the equivalence between ASP and some of these logics, particularly Default and Autoepistemic logic [36]. However, ASP programs are simpler and more restrictive than the theories of these logics in the sense that they do not allow arbitrary formulas. This, together with the implementation of efficient answer set solvers, has made ASP to be considered as the primary candidate for an effective knowledge representation tool.

When it comes to declarative problem solving, the main idea of ASP is to encode a computational problem as a logic program whose answer sets correspond to the solutions of that problem and then use an answer set solver for returning the answer sets[1]. In general ASP is oriented to solving difficult search problems. The complexity of the problem that can be represented in ASP corresponds to the complexity of the problem of finding an answer set of a logic program written in a particular language. Accordingly, finding an answer set of a logic program without disjunction is an NP-Complete problem, implying that all NP and coNP problems can be compactly represented with such programs. On the other hand, disjunctive logic programs are strictly more expressive (subject to the belief that P≠ NP), and can be used for solving problems up to the Σ^P_2 and the Δ^P_2 complexity class [36].

This chapter is an overview of ASP. It starts with the definition of the syntax and semantics of ASP programs in the language of DLV. Smodels and Clasp accept a subclass of this language (do not allow disjunction) and its definition is contained in the one of DLV. This is followed by an example of an ASP program for solving the famous n-queens problem. Then the relationship between ASP and other logic programming paradigms is discussed, particularly Prolog and logic programming under the well founded semantics. The chapter concludes with a brief overview of the Smodels, Clasp and the DLV answer set solvers.

[1]This corresponds to the approach in satisfiability checking, where the problem is encoded as a propositional theory whose models correspond to the solutions of that problem [30]

3.1 ASP Syntax (The language of DLV)

An answer set framework consists of two languages, an axiom language for writing rules, and a query language for writing queries. The latter is not of particular importance in the dissertation and it is omitted. What follows is the definition of an axiom language $\mathcal{L}^{DLV}$ which corresponds to the language of answer set programs accepted by the *DLV* answer set solver.

$\mathcal{L}^{DLV}$ can be used for expressing disjunctive extended logic programs, i.e. logic programs containing disjunctions in the head of the rules (defined at the end of the section) and both default and explicit negation (and thus extended).

Definition 1 The alphabet of $\mathcal{L}^{DLV}$ consists of seven classes of symbols: variables, constants, predicate symbols, connectives, punctuation symbols and the special symbol $\bot$, where the connective symbols are fixed to the {¬, ←, or, not, ‘,’} and punctuation symbols to {‘(’,’)’ ,’.’}.

In ASP ‘or’ is used instead of ‘$\vee$’ and ‘,’ instead of ‘$\wedge$’ in order to better differentiate them from first order theories . The ‘not’ stands for negation as failure (default negation) while ‘¬’ stands for explicit negation.

Definition 2 An *atom* is of the form $p(t_1,\ldots,t_n)$ where p is a predicate symbol and $t_1 \ldots t_n$ are either variables or constants. If all $t_1,\ldots,t_n$ are constants then the atom is set to be ground.

Definition 3 A *literal* is either an atom, or an atom preceeded by the symbol ‘¬’. The former is referred to as positive while the latter as negative literal. The literal is ground if no variables occur in it.

Definition 4 A *rule* is of the form:

$$L_0 \text{ or} \ldots \text{or } L_k \leftarrow L_{k+1},\ldots, L_m, \textbf{ not } L_{m+1},\ldots, \textbf{ not } L_n$$

where L_is are literals, or if $k = 0$, L_0 may be the symbol $\bot$, and $k > 0$, $m \geq k$, and $n \geq m$. A rule is ground if it composed by ground literals only.

Intuitively the meaning of the above rule is that if $L_{k+1},\ldots, L_m$ are true, and there is no evidence of the truthfulness of $L_{m+1},\ldots, L_n$ (i.e. they can be safely assumed false), then one of the literals $L_0 \ldots L_k$ must be true.

In general, an answer set program Π (in the language of DLV) is a set of rules of the above form. However, here (for simplicity) logic programs are assumed to be ground, thus a logic program Π stands for the corresponding ground instantiation *ground*(Π) where each non ground rule r in Π is replaced with the set of its ground instantiations *ground*(r, $\mathcal{L}^{DLV}$) obtained by all possible substitutions of constants in $\mathcal{L}^{DLV}$ for the variables in r. Accordingly, the language $\mathcal{L}^{DLV}$ consists of ground rules only.

Definition 5 Given a set of variables symbols, a set of constant symbols and a set of predicate symbols, the *answer set language* $\mathcal{L}^{DLV}$ consists of the set of all ground rules constructed from the symbols of the alphabet.

Terminology: The left side of ← is the *head* of the rule while the right is the *body*. A rule with no body and a single disjunct in the head is a *fact*, and it’s written without the ←

connective (ex: 'L_0. '). A rule with the symbol $\perp$ in the head is a *constraint* (since its body must not hold in any answer set, which is apparent in the next section where the semantics is defined) and it is written as a rule with empty head (ex: '$\leftarrow L_1,\ldots, L_m$, **not** $L_{m+1},\ldots,$ **not** L_n').

3.2 ASP Semantics

There are several ways for defining the answer set semantics of a logic program. Here the Model Theoretical characterization is used [29,36], while others can be found in [36]. This definition uses a transformation of the original program (known as Gelfond-Lifschitz transformation) to a default-negation-free program, and then uses the definition of the semantics of default-negation-free programs for defining the semantics of the original program.

Definition 6 (*Gelfond-Lifschitz transformation*) Let Π be a logic program in the language of $\mathcal{L}^{DLV}$. For any set of literals S, the program Π^S (referred to as reduct of Π) is obtained from Π by deleting:

- each rule that contains a literal L preceded by 'not'('**not** L') in the body and $L \in S$
- all literals preceded by 'not' from the bodies of the remaining rules

Note that Π^S is a program which does not contain default negation. Let us call the language of such a programs $\mathcal{L}^{DLV\text{-}\mathbf{not}}$. What follows is the definition of the semantics of programs in $\mathcal{L}^{DLV\text{-}\mathbf{not}}$.

Definition 6 An *interpretation* S of a $\mathcal{L}^{DLV\text{-}\mathbf{not}}$ program Π is any subset of the set of literals in Π. An interpretation S of Π is said to satisfy the $\mathcal{L}^{DLV\text{-}\mathbf{not}}$ rule 'L_0 or…or $L_k \leftarrow L_{k+1},\ldots, L_m$' if:

- if $L_0 = \perp$ and $k=0$ then $\{L_{k+1},\ldots, L_m\} \not\subseteq S$,
- Otherwise: $\{L_{k+1},\ldots, L_m\} \subseteq S$ implies that $\{L_0,\ldots,L_k\} \cap S \neq \varnothing$

Definition 7 A *model* A of an $\mathcal{L}^{DLV\text{-}\mathbf{not}}$ program Π is an interpretation S of Π which satisfies all rules in Π.

Note that the above definition allows models which contain a pair of complementary literals (i.e. an inconsistent model). In most answer set solvers, this is the default option.

Definition 8 An *answer set* of an $\mathcal{L}^{DLV\text{-}\mathbf{not}}$ program Π is a model of Π which is minimal among the models of Π.

This definition applies the closed world assumption (CWA) in answer sets. Consider a simple default-negation-free program consisting of the following rules $\{a,\ b \leftarrow c\}$. The models of this program are $\{a\}$, $\{a,b\}$, and $\{a,c,b\}$ since all of them satisfy the two rules. However there is no evidence of the truthfulness of atoms b and c and thus they should be assumed false. This is done by taking only the minimal models to be answer sets, i.e. the models which are super set of other models are not answer sets. Accordingly the single answer set of the above program is $\{a\}$ which is also the intended meaning of the program.

Let us denote the answer sets of an $\mathcal{L}^{DLV\text{-}\mathbf{not}}$ program Π as $\mathcal{A}s(\Pi)$.

Definition 9 A set of literals S is an *answer set* of an $\mathcal{L}^{DLV}$ program Π if $S \in \mathcal{AS}(\Pi^S)$, where Π^S is an $\mathcal{L}^{DLV\text{-}\mathbf{not}}$ program obtained from Π according to transformation in definition 6.

The semantics is indeed nonmonotonic. Consider a knowledge base containing two rules stating that all birds fly and that Tweety is a bird. An obvious conclusion is that tweety flies. However later a new knowledge arrives and now it is known that tweety is a penguin. The knowledge base should not imply that tweety flies since penguins do not fly. This is a nonmonotonic property and can be captured by using default negation in the 'all birds fly' rule for assuming that the bird which flies is not a penguin. More specifically by the following program:

fly(X) ← bird(X), **not** penguin(X).
bird(tweety).

where X is a variable, and tweety is a constant. The corresponding ground instantiation is:

fly(tweety) ← bird(tweety), **not** penguin(tweety).
bird(tweety).

The only answer set of the above program is {bird(tweety), fly(tweety)}). However if the program is extended with a new fact stating that Tweety is a penguin 'penguin(tweetty)', the only answer set of the new program is {bird(tweety), penguin(tweety) } and thus fly(tweety) does no longer belong to any of the answer sets. Accordingly, when extending the knowledge base some of the previously drown conclusions may be no longer valid, which is the definition of nonmonotonicity.

For proving that a set of literals is an answer set of a logic program later on, it is useful to define the relationship between the rules of the program and its answer sets. For programs in the $\mathcal{L}^{DLV}$ language, this can be formalized with the following proposition [36].

Proposition 3.1: Let S be an answer set of an $\mathcal{L}^{DLV}$ program Π. For any ground instance of a rule of type

$$L_0 \; or \ldots \; or \; L_k \leftarrow L_{k+1}, \ldots, L_m, \; \mathbf{not} \; L_{m+1}, \ldots, \; \mathbf{not} \; L_n$$

where $L_{0..n}$ are literals as defined in definition 3.

(a) **Forced disjunction proposition:**

(i) if k=0 and $L_0 = \bot$ then $\{L_{k+1}, \ldots, L_m\} \not\subseteq S$, or $\{L_{m+1}, \ldots, L_n\} \cap S \neq \varnothing$.

(ii) if $\{L_{k+1}, \ldots, L_m\} \subseteq S$ and $\{L_{m+1}, \ldots, L_n\} \cap S \neq \varnothing$ then there exist an i, $0 \leq i \leq k$ such that $L_i \in S$.

(b) **Exclusive supporting rule proposition:**

If S is an answer set of an $\mathcal{L}^{DLV}$ program Π and $L \in S$ then there is a ground instance of a rule in Π of the form above such that $\{L_{k+1}, \ldots, L_m\} \subseteq S$, and $\{L_{m+1}, \ldots, L_n\} \cap S = \varnothing$, and $\{L_0, \ldots, L_k\} \cap S = \{L\}$.

3.3 Solving problems with ASP (the guess and check method)

A common method for solving problems using the answer set semantics of logic program is the so *called guess and check method* [36]. In general, when this method is applied the logic program consists of the following three parts:

- Domain specification, where the specific instance of the problem is defined
- Guess part - Generating answer sets corresponding to the potential solutions.
- Check part - Pruning the wrong answer sets with constrain statements.

To illustrate the above a well known problem, the n-queens, is used. The problem is to place n queens on an n x n chess board such that the queens do not attack each other given the standard queen's moves in chess. In other words, a solution requires that no column, row or diagonal has more than one queen

Accordingly, firstly the domain of the problem is defined, in this case by two sets of facts, one representing the n columns and one representing the n rows.

```
column(1). row(1).
column(2). row(2).
…
column(n). row(n).
```

The next step is to generate models for enumerating the potential solutions. This is enabled by the fact that logic programs can have many answer sets and each potential solution is encoded with an answer set of the program. A straightforward approach for the n-queens problem is to enumerate every possible combination of queens on a chess board. This is done by the following rule:

```
hasQueen(X,Y) or  empty(X,Y) ← row(X), column(Y).
```

where hasQueen(X,Y) has the intuitive meaning that a queen is placed in the slot with coordinates X and Y, while the empty(X,Y) means that the corresponding slot is empty.

The above rule has the meaning that each position on the chess board is either empty or it contains a queen. Because of the minimality of answer sets, the rule does not generate models in which a position is both empty and occupied, which is adequate in this case since positions should not be both occupied and empty. Note that here disjunction is not necessary since the same can be expressed by the following two disjunction-free rules, however in a less declarative manner.

```
hasQueen(X,Y) ← row(X), column(Y), not empty(X,Y).
empty(X,Y) ← row(X), column(Y), not hasQueen(X,Y).
```

This method can not be applied in general, the rule 'a or b.' is not equivalent to the rules '$a \leftarrow$ not b.' and '$b \leftarrow$ not a.'. For example, the program $P = \{a$ or $b, a \leftarrow b, b \leftarrow a\}$ has a single answer set A = $\{a, b\}$, while the program $P' = \{a \leftarrow$ not $b, b \leftarrow$ not a, a $\leftarrow b, b \leftarrow a\}$ has no answer sets. This is evidence that disjunctive logic programs are strictly more expressive than disjunction-free logic programs (under the assumption that P ≠ NP).

It is obvious that the rules above generate many invalid models, including ones which have more than or less than 1 queen in a row, column or diagonal. These models should be pruned if an answer set is to correspond to a valid solution. This is enabled by the fact

that some logic programs do not have an answer set. For example a logic program containing a rule of the form '*a*←not *a.*' does not have any answer sets since there is no set of literals which satisfies this rule. Similarly '*a*←*b*, not *a.*' will imply that *b* must not hold in an answer set of the program, which equivalent to the rule '⊥←*b*'[2]. In such case *b* is considered as a constraint and is simple written as a rule with empty head '←*b.*' Constraints comprise the check part of the answer set program and for the n-queens problem this part contains the following three constrains meaning that no two queens can appear in the same row, column, and diagonal repetitively.

```
← row(X), column(Y), row(XX), hasQueen(X,Y), hasQueen(XX,Y), X≠XX.
← row(X),column(Y), column(YY), hasQueen(X,Y),  hasQueen(X,YY),  Y≠YY.
← row(X), column(Y), row(XX), column(YY), X≠XX, Y≠YY, hasQueen(X,Y),
hasQueen(XX,YY), X-XX = Y-YY.
```

Note that the inequality metters only for grounding the rules (i.e. the ground program does not contain inequality), and thus it is not defined in the semantics in the above chapter. This is almost the solution of the n-queens problem. What is left is to assure that at least n queens are placed on the board. This can be done by the following two rules which state that each row has at least one queen (and since there are n rows, there will be at least n queens).

```
← row(X), not hasRowQueen(X).
hasRowQueen(X) ← row(X), column(Y), hasQueen(X,Y).
```

The answer sets of the above program are valid solution to the n-queens problem. However, in the guess part many invalid models are generated, which can be improved using techniques for more careful generation of models [36]. For example, generate X and Y in such a way that for every X there is at least one Y (*general enumeration of at least one*) or for every X exactly one Y (*general enumeration of exactly one*). The latter corresponds to the requirements of the n-queens problem (for each row, column exactly one queen) and it corresponds the following rules:

```
hasQueen(X,Y) ←row(X),column(Y), not noQueen(Q,X,Y).
noQueen(X,Y) ←row(X), column(Y), column(YY), YY≠Y, hasQueen(X,YY).
noQueen(X,Y) ←row(X), column(Y), row(XX), X≠XX, hasQueen(XX,Y).
```

These rules generate models where every row and column has exactly one queen and thus only one constrain needs to be added (the one for the diagonals stated above) in order the answer sets of the program to be valid solutions of the n-queens problem. Accordingly the final program will consist of the domain specification and the following rules:

Guess

```
hasQueen(X,Y) ←row(X),column(Y), not noQueen(Q,X,Y).
noQueen(X,Y) ←row(X), column(Y), column(YY), YY≠Y, hasQueen(X,YY).
noQueen(X,Y) ←row(X), column(Y), row(XX), X≠XX, hasQueen(XX,Y).
```

Check

```
← row(X), column(Y), row(XX), column(YY), X≠XX, Y≠YY, hasQueen(X,Y),
hasQueen(XX,YY), X-XX = Y-YY.
```

[2]This implies that the symbol ⊥ does not increase the expressiveness of a logic program with default negation under the answer set semantics, however, it gives more direct and intuitive definition of constraints.

3.4 ASP and other Logic Programming paradigms

In this section the differences between ASP and other Logic Programming paradigms are briefly discussed, particularly Prolog and logic programming under the well founded semantics (WFS) [37]. Before comparing the syntax and semantics of ASP with each of these paradigms, it is important to note that in ASP the solutions of a particular program are given as models, while in the WFS and Prolog, the solutions are given as substitutions in a single model. Thus, the problem solving method of ASP is fundamentally different, and in this aspect, ASP is much closer to satisfiablity checking than to Prolog and the WFS.

Although, Prolog grew from programming with Horn logic (a subset of classical logic), several nondeclarative features were added in order to make it programming friendly. For example, the *cut* operator '!' is extra logical and is still not characterized by any logic [36]. ASP is full declarative and thus does not contain this operator. Another important difference, also supporting the fact that Prolog is not fully declarative, is that the order of the rules and the literals in the rules' bodies matters in Prolog. In essence Prolog processes rules from top to bottom and body literals from left of right. This is not the case in ASP where a program is considered to be a set (not a sequence) of rules, and each body of a rule is characterized by two sets of literals, one for the positive and one for the negative literals. In addition, query processing in Prolog is done in a top down fashion from queries to facts. This feature makes Prolog vulnerable to loops even for simple programs such as the following program consisting of just one rule {a ← a.}. This is intensified when dealing with negation as failure. The answer set semantics do not suffer from such deficiencies.

Aside from answer sets, another fully declarative paradigm is logic programming under the well founded semantics [37]. The well founded semantics is three valued where each literal can be true, false or undefined. The main idea behind this semantics is that the literals for which it is not certain to be true or false (by default) are considered undefined. For example consider the program P= {a ← not b, b← not a, c ← a, c ← b}, its unique well founded model assigns all atoms as undefined. Intuitively this is because from the first two rules it can not be decided which one of a and b is true and which one is false and thus they are both considered as undefined, and since they are both undefined c is undefined as well. This is not the intended meaning of the program, in fact c should be true since no matter which one from a and b is true, there is a rule for deriving c. The answer set semantics overcome this since c is true in all answer sets of the above program. Furthermore, all logic programs have a unique well-founded model. This is certainly not the case with answer sets. For example {a ← not a.} has no stable model while its unique well founded model assigns a as undefined. Also the program {a← not b, b← not a} has two answer sets {a} and {b} while its unique well founded model assigns a and b as undefined. This is an advantage of the answer set semantics since the non determinism introduced by the multiple answer sets is useful for enumerating different choices and the fact that some programs do not have an answer set is used for defining constrains that eliminate the wrong choices. Thus the answer set semantics is more powerful than the well founded, however, the latter is more tractable and in fact computable in polynomial time.

3.5 Answer Sets Solvers

Answer set solvers are tools that receive as input a logic program (in some logic programming language) and give as output its answer sets. A number of such solvers exist nowadays, some of which are reviewed next.

3.5.1 Smodels and CLASP (The language of LParse)

The Smodels and the CLASP answer sets solver consist of two separate modules, the LParse module which accepts and grounds the users' programs, and the actual answer set solver. Thus, they both accept programs in the language of LParse.

The language of LParse $\mathcal{L}^{LP}$ is a subclass of the $\mathcal{L}^{DLV}$ language in which the head of the rule is restricted to a single literal or the special symbol $\bot$. Accordingly, a rule in LParse is of the form:

$$L_0 \leftarrow L_1,\ldots, L_m, \textbf{not } L_{m+1},\ldots, \textbf{not } L_n$$

where $L_{1..n}$ are literals as defined previously and L_0 is either a literal or the symbol $\bot$.

LParse has a Prolog like syntax for writing rules, i.e. 'head:-body.', where the literals in the body are separated with ',', and the head is a single literal or it is omitted in the case of $\bot$.

Aside from these basic rules LParse offers a number of extended rules which enlarge its applicability. The most important ones are reviewed next; others together with a detailed explanation can be found in [38].

Weights and weight constraints are of particular interest since together with optimization statements can be used for modeling optimization problems with LParse. Basically a weight constraint has the following form:

$$L \leq [\ L_0=w_0,\ldots, L_m=w_m, \textbf{not } L_{m+1}=w_{m+1},\ldots, \textbf{not } L_n=w_n\ ] \leq U$$

where $L_{0..n}$ are literals, $w_{1..n}$ are the corresponding weights (a positive integer value), and L and U are the lower and the upper limit respectively. Intuitively such a constraint is satisfied when the sum of the weights of the satisfied literals in the brackets is between the upper and the lower limit inclusive.

In LParse weight constraints are treated syntactically as literals, thus such a constraint can appear both as head and in the body of a basic rule. A *weight constraint rule* (also referred to as choice rule) is a rule that has a weight constraint (restricted to positive literals only) as head. Such a rule is satisfied if the weight constraint (the head) is satisfied whenever the body is satisfied.

Optimization statements are used to maximize or minimize the sum of weights of the satisfied literals. In LParse optimization statements are of the form:

maximize[$L_0=w_0,\ldots, L_m=w_m,$ **not** $L_{m+1}=w_{m+1},\ldots,$ **not** $L_n=w_n$]
maximize{ $L_0,\ldots, L_m,$ **not** $L_{m+1},\ldots,$ **not** L_n }
minimize [$L_0=w_0,\ldots, L_m=w_m,$ **not** $L_{m+1}=w_{m+1},\ldots,$ **not** $L_n=w_n$]
minimize { $L_0,\ldots, L_m,$ **not** $L_{m+1},\ldots,$ **not** L_n }

where $w_{0..n}$ and $L_{0..m}$ are the same as above previously. When such optimization statements are given, Smodels and Clasp try to find the answer sets with as many or as few of the literals in the optimization statement (in the case of curly brackets), or maximize or minimize the sum of the weights of the literals (in the case of square brackets)

Weights can also be assigned to literals using the *weight* declaration. This is done as follows:

```
#weight L = expr
#weight L = L'
```

where *L and L'* are literals and *expr* is an arithmetic expression that may contain variables which occur in *L.*

For example consider the following program with two answer sets {a} and {b}:

```
a :- not b.
b:- not a.
#weight a=5.
#weight b=3.
```

An optimization statement '`maximize{a,b}`' will return the stable model {a} since its weight sum is 5 which is bigger than the weight sum of the other alternative {b} which is 3. Similarly '`minimize{a,b}`' will return the stable model{b}.

A specific type of weight constraint is the cardinality constraint where all the weights have the value 1. In this case the weights are omitted and the constraint has the following form:

$$L \leq \{ L_0,\ldots, L_m, \textbf{not } L_{m+1},\ldots, \textbf{not } L_n \} \leq U$$

where L,U and $L_{0..m}$ are the same as above.

Such constrains can be used for generating models for the n-gueens where each row and column have exactly one queen in a more elegant fashion than the one described at the end of the section 3.3. For example given a column there is exactly one row such that the queen is in that row and column corresponds to the following rule:

```
1 {hasQueen(X,Y):row(X)} 1 :- column(Y).
```

and given a row there is exactly one column such that the queen is in that row and column:

```
1 {hasQueen(X,Y):column(Y)} 1 :- row(X).
```

The above cardinality constraints use a conditional literal. Such literal is of the form $p(X):q(X)$, which is an abbreviation for the set of the literals $p/1$ derived from the valid instantiations of the variable X in the predicate $q/1$. For example, if the extension of q is $\{q(a_1), q(a_2), q(a_3), \ldots, q(a_n)\}$, then $p(X):q(X)$ is equivalent to writing $p(a_1), p(a_2), p(a_3), \ldots, p(a_n)$.

3.5.2 DLV

DLV is another answer set solver that provides similar functionalities as Smodels and Clasp. The main difference is that DLV accepts a more expressive language than the other two by allowing for disjunction in the head of the rules. Thus a basic rule in DLV has the following form:

$$L_0 \, or \ldots \, or \, L_i \leftarrow L_{i+1},\ldots, L_m, \textbf{not } L_{m+1},\ldots, \textbf{not } L_n$$

where $L_{0..n}$ are literals as defined previously and in the case i=0, L_0 is either a literal or the symbol $\perp$. Same as LParse, DLV rules have a Prolog like syntax, 'head:-body.', where the head is a possibly empty (in the case of $\perp$) set of literals separated with 'v' (instead of '*or*') and the body is a possibly empty (in the case of facts) set of literals separated with ','.

In order to achieve similar effect as the optimization statements in LParse, DLV uses week constrains of the following form:

$$:\sim L_0,\ldots, L_m, \textbf{not } L_{m+1},\ldots, \textbf{not } L_n. \quad [\mathit{weights}: \mathit{level}]$$

where $L_{0..n}$ are literals, and *weights* and *level* are integer values or integer variables appearing in $L_{0..m}$.

Given a problem with weak constraints DLV computes all the answer sets (without considering the weak constraints) before ordering them with respect to the weights and the priority of the week constraints that they violate. The answer sets which violate the minimum are then selected and returned. In presence of different priority levels, the minimization is done with respect to the weak constraints with the highest priority, then the second highest priority and so on.

Accordingly, a LParse optimization statement

$$\text{maximize}[\, L_0=w_0,\ldots, L_m=w_m, \textbf{not } L_{m+1}=w_{m+1},\ldots, \textbf{not } L_n=w_n \,]$$

can be represented in DLV with the following week constraints:

$:\sim L_0.$ $[w_0: 1]$
…
$:\sim L_m.$ $[w_m: 1]$
$:\sim \textbf{not } L_{m+1}.$ $[w_{m+1}: 1]$
…
$:\sim \textbf{not } L_n.$ $[w_n: 1]$

Aside the weak constrains DLV also supports aggregate functions of the form:

$$\#\text{name}\{V_0,\ldots, V_n : L_0=w_0,\ldots, L_m=w_m, \textbf{not } L_{m+1}=w_{m+1},\ldots, \textbf{not } L_n=w_n\}$$

where name is the function(count, sum, min, max, or times), $L_{0..n}$ are the same as previously, and $V_0,\ldots, V_n$ are variables occurring in $L_{0..m}$. For example '#count {X : *a*(X)}' will return the number of ground instantiations of the predicate *a*, while '#sum {X : *a*(X)}' will return the sum of all values of the first argument in the ground instantiations of the predicate *a*. Similarly, #times returns the product of values, #min the minimal value and #max the maximal value.

For example consider the program consisting of the following rules:

```
number(3).
number(5).
number(8).
sum(S):-#sum{X:number(X)}=S.
times(P):-#times{X:number(X)}=P.
min(M):-#min{X:number(X)}=M.
max(M):-#max{X:number(X)}=M.
```

The single answer set of the above program is the following set of literals {number(3), number(5), number(8), sum(16), times(120), min(3), max(8) }

Aggregates appear to be useful for solving several MARA problems. As will become apparent later on, they can also be compactly expressed in LParse with weight constraints.

4. Application of Answer Set Programming in Multiagent Resource Allocation

This chapter covers the application of ASP in MARA. This comes quite naturally taking into account that both research areas are closely related to the AI field, and that the need for declarative specification is often emphasized in Multi Agent Systems. The focus is on the elementary MARA problems covered in chapter 2. All of the preference representations are taken into account, with the exception of SLPs, which are procedural in nature and thus inconvenient for ASP, and the ceteris paribus preferences, which are rarely used in practice.

In general, an ASP solution to a particular MARA problem is defined in terms of a translation from the problem to a logic program together with a proof of the correspondence between the program's answer sets and the problem's solutions. The method used for solving MARA problems, is the guess and check method described in section 3.2. The main idea behind this method is to enumerate the valid allocations (guess) and then remove the ones which do not satisfy certain criteria (check). When the guess and check method is applied to MARA problems, the problem's input corresponds to the data (domain) part of the logic program while the problem's solution comprises the other two parts (guess and check). Since the guess and data parts are common for many MARA problems, they are discussed prior to the concrete solutions.

The chapter begins with the motives for applying ASP in MARA, followed by an ASP formalization of the main MARA concepts including the formalization of agents, resources, agents' preferences and social welfare. Afterwards, rules are defined for generating answer set corresponding to valid allocation, i.e. the guess part of the logic programs, followed by concrete solutions to several MARA problems. This includes: the decision and optimization versions of the Welfare Improvement(WI), Welfare Optimization(WO), Pareto-optimality(PO) and the Envy Free(EF) problem, the problem of determining the next proposal under the Zeuthen strategy in the monotonic consensus protocol, as well as a brief insight in ASP for combinatorial auctions. The chapter finishes with the formalization of some common constraints in resource allocation.

4. 1 Motivation

Before proceeding with the ASP solution for the problems in MARA, in this section it is established that indeed ASP can be used for solving MARA problems, together with the benefits from this approach.

In general MARA problems fall in two categories: decision problems and optimization problems. A decision problem is a problem for which the solution is a boolean value which evaluates to true or false, in other words a question with a yes/no answer. For example: is there an envy free allocation; is a given allocation Pareto-optimal; is a given allocation optimal; and so on. When investigating whether ASP can be used for solving such problems their complexity is crucial. As stated previously, all NP and coNP problems can be declaratively specified with a logic program in the ASP framework. The difference between the two is that an NP decision problem evaluates to true if a stable model having certain property exists, while it is coNP if no stable models having certain property exists. Since the decision problems discussed in chapter 2 (section 2.5) belong to

either of these two complexity classes, all of them can be expressed with a logic program under the answer set semantics. For example, consider a single answer set to correspond to a particular allocation, there is an envy free allocation if a single allocation can be found which is envy free (the envy free problem, NP-Complete), or, an allocation is Pareto-optimal if no other allocation (answer set) can be found that Pareto dominates the first (the decision version of Pareto-optimality which is coNP-complete).

An optimization problem is a problem for which the solution is an allocation that maximizes (or minimize) the value of some function. Examples of such problems in MARA are: determine the winner in combinatorial auctions; find an allocation that maximizes a collective utility function; find an allocation that minimizes the number of envious agents; and so on. In general optimization problems are NP-Hard and thus beyond the NP complexity class and the expressiveness of the answer set semantics[3]. In order to overcome this, common answer set solvers have integrated weights and optimization statements in their systems. Thus, consider a single answer set to correspond to a particular allocation, a weight is assigned to each allocation according to some criteria and then an optimization statement is used for returning the best allocations. For example, in the case where the goal is to minimize the number of envious agents a weight can be added for each envious agent. An optimization statement to minimize this weight will then lead to allocations with the minimal number of envious agents.

Having convinced that ASP can be, at least theoretically, used for solving problems in MARA, some of the benefits of using this approach are discussed next. To begin with, and what is actually the biggest power of ASP, is that it is fully declarative, thus the problem solution resembles the problem specification. The semantics is very intuitive leading to understandable and compact solutions, which is apparent in the previous chapter when the solution to the n-queens problem is presented. Furthermore, there is extensive theoretical work done on the answer set semantics that enables proving different properties of the logic programs under the answer semantics, and thus proving correctness of the solutions. Finally, the reasonably efficient answer set solvers give a promise for efficient solutions to the problems in MARA.

Aside from the above benefits, which are from applying ASP in general, there are advantages which arise from the nature of the problems in MARA. In essence, combining different problems as well as adding constrains is very practical with ASP. For example, consider that an application is interested in the best envy free solution in a general resource allocation problem. This can be achieved simply by combining the ASP programs for finding an envy free allocation and for finding an optimal allocation. Also the optimization problems can be combined using the different priority levels in DLV leading to solutions for problems like finding the allocation which minimize the envy degree among the allocations that have the maximum social welfare, or vice versa. Furthermore, most of the ASP solutions are independent from the agents' preference representation. Thus the same solutions can be applied to different preference representation as long as they have the same structure (ordinal or cardinal). Also CUFs

[3] Often the winner determination problem in the literature is said to be NP-Complete, this refers to the decision version of this problem, i.e. the welfare optimization problem in chapter 2

can be compactly expressed using the aggregate functions in DLV making the difference between egalitarian, elitist or utilitarian social welfare a matter of changing a line.

Last, but not least, the solutions in ASP are very flexible in terms of constrains. Sentences like: two resource must be allocated to a single agent, a resource must not be allocated to a specific agent, this is a minimum price of a resource (in auctions), and so on, in most of the cases amount to adding a single constraint rule to the program.

Given the knowledge representation roots of ASP, it is reasonable to assume that it is adequate for representation of preferences. In essence, ASP has sufficient expressivity to represent preferences in most of the ways discussed in the previous chapter. Logic based preference representation is expectedly the most adequate for ASP, but the k-additive and the bundle form can also be compactly represented.

Finally, MARA is not the only one that can benefit from this work. In fact, most of the programs discussed in this chapter can be used as benchmarks for testing answer set solvers, and also provide an example of using ASP for solving problems in different complexity classes.

4.2 Formalization of the resource allocation setting

A resource allocation setting, which is an input to most of the MARA problems, is defined as a triple $\langle A,R,U\rangle$ where:

$A = \{1,2,\ldots,n\}$ is a set of n agents.

$R = \{r_1,r_2,\ldots,r_m\}$ is a set of m (if not stated differently single-unit) indivisible and non-sharable resources.

$U = \{ur_1, ur_2,\ldots, ur_n\}$ is the set of preference representations of the agents (unlike chapter 2 where U was a set of complete preference functions or relations). For example, if the preferences are in k-additive form ur_a is a function from bundles to coefficients, while in the bundle form ur_a is a function from bundles to utility values.

Recall that an *allocation* is a mapping P: $A \rightarrow 2^R$ from the agents in A to a subset of the resources R, where $P(a)\cap P(b)= \varnothing$ for any two agents $a \neq b$. The set of all allocations of resources to agents is denoted by $Al_{n,m}$.

While the modeling of the preference relations depends on the preference representation used, the set of agents and the set of resources can be universally modeled in ASP as a set of facts of the predicates `agent`/1 and `resource`/1 respectively. More formally, given a set of agents A and a set of resources R, a translation $\Pi_{AR}(A,R)$ is defined such that:

- For all $a \in A$, a fact `agent(`a`)` $\in \Pi_{AR}(A,R)$
- For all $r \in R$, a fact `resource(`r`)` $\in \Pi_{AR}(A,R)$

All of the programs discussed in the rest of this section are assumed to contain this set of facts and thus the translation of preferences coincides with the translation of the resource allocation setting.

The formalization of the agents' preferences in a particular representation may also differ from problem to problem. In general there are two cases:

- The preferences of an agent are defined for the bundle that an agent receives in a particular allocation. Such preferences are referred to as simple preferences.
- The preferences of an agent are defined for the bundles allocated to all agents (in the case of ordinal preferences they are compared). Such preferences are necessary for problems including the notion of envy free and are referred to as extended preferences.

In the rest of this section preferences are formalized for each of these cases. The formalizations define the agents' preferences for a single allocation (except of ordinal preferences where such formalization does not make sense since there should be at least two allocations in order to be compared). However, as shown later, the formalizations can be modified to represent agents' preferences for several allocations by simply adding an extra argument to the predicates for denoting the allocation. This is necessary for solving the MARA problems which require two allocations to be encoded in each answer set (e.g. Pareto-optimality, Welfare Improvement).

Prior to proceeding, it is useful to define formally the correspondence between an allocation P and a logic program (set of facts) Π_P, since it will be used in most of the theorems which follow. For the case where a single allocation P is represented in an answer set:

- for every agent-bundle pair $\langle a, B_a \rangle \in P$, and for every resource $r \in B_a$, a fact `has(`a`, `r`)` $\in \Pi_P$.

Intuitively, the atom `has(`a`, `r`)` in an answer set means that the corresponding allocation P allocates the resource r to the agent a. The full function (e.g. a set of facts `utility(`a`,`b`,`u`)`, meaning that the agent a has utility u for the bundle b) is not represented in the answer sets since such a function is exponential in the number of resources.

4.2.1 Explicit preferences

When preferences are given in the explicit form, utilities are defined for every agent-bundle pair, and thus the full utility function coincides with its representation. Formally, for every agent $a \in A$, and for every bundle $B \subseteq R$, there is a value $v \in$ I (the set of integers), such that $\langle B, v \rangle \in ur_a$.

For the simple preferences (agents utilities are defined only for the bundles they receive) the explicit representation is formalized in ASP with the following logic program $\Pi_{EF}(A,R,U)$:

- For every $a \in A$, and for every tuple $\langle B,v \rangle \in ur_a$, a rule $rl \in \Pi_{EF}(A,R,U)$, where head(rl) =`utility(`a`, `v`)`, and for every resource r if $r \in B$ then `has(`a`, `r`)` $\in$ body(rl); else `not has(`a`, `r`)` $\in$ body(rl).

For example, consider the set of resources to be {*r1*, *r2*, *r3*}, and an agent a with a utility 3 for the bundle {*r2*, *r3*}. This corresponds to the following rule:

`utility(`a`,3):-not has(`a`, r1), has(`a`, r2) , has(`a`, r3).`

Proposition 4.1: Given a resource allocation setting $\langle A,R,U\rangle$ with the explicit preference representations in U and particular allocation P which allocates a bundle $B_a \subseteq$ R to each agent a, for any answer sets $S \in \mathcal{AS}(\Pi_{EF}(A,R,U) \cup \Pi_P)$, `utility(`$a$`,`$v$`)` $\in S$ if and only if $u_a(B_a)=v$.

Proof: Trivial.

For the extended preferences (agents' utilities are defined for the bundles received by all agents) the explicit representation is formalized with the translation $\Pi_{EFex}(A,R,U)$ defined as follows:

- For every $a \in A$, and for every tuple $\langle B,v\rangle \in ur_a$, a rule $rl \in \Pi_{EFex}(A,R,U)$, where head(rl) =`utility(`a`,`v`,Ag)`, and for every resource r if $r \in B$ then `has(A,`r`)` $\in$ body(rl); else `not has(Ag,` r`)` $\in$ body(rl), and `agent(Ag)` $\in$ body(rl).

Thus the translation of the previous example corresponds to the following rule:

```
utility(a,3,Ag):-
     not has(Ag, r1),has(Ag, r2) ,has(Ag, r3),agent(Ag).
```

Proposition 4.2: Given a resource allocation setting $\langle A,R,U\rangle$ with the explicit preference representations in U and particular allocation P which allocates a bundle $B_a \subseteq R$ to each agent a, for any answer sets $S \in \mathcal{AS}(\Pi_{EFex}(A,R,U) \cup \Pi_P)$, `utility(`$a$`,`$v$`,`$b$`)` $\in S$ if and only if $u_a(B_b)=v$.

Proof: Trivial.

4.2.2 Preferences in bundle form

The bundle form is similar to the explicit form with a difference that utilities are only defined for bundles for which an agent has a nonzero utility value. Formally, for every agent $a \in A$, and for every bundle $B \subseteq R$ for which $u_a(B) = v$ and $v \neq 0$, $\langle B,v\rangle \in ur_a$.

For the simple preferences, the bundle form is formalized in ASP with a translation $\Pi_{BF}(A,R,U)$ defined as follows.

- For every $a \in A$, and for every tuple $\langle B, v\rangle \in ur_a$, a rule $rl \in \Pi_{BF}(A,R,U)$, where head(rl) =`utility(`a`, `v`)`, and for every resource r, if $r \in B$ then '`has(`a`, `r`)` ' $\in$ body(rl); else `not has(`a`, `r`)` $\in$ body(rl). (1)
- a rule for marking the agents which have a nonzero utility

  `has_utility(A):- utility(A,U), U!=0.` $\in \Pi_{BF}(A,R,U)$ (2)
- a rule assigning 0 utility for the rest of the agents

  `utility(A,0):-agent(A),not has_utility(A).` $\in \Pi_{BF}(A,R,U)$ (3)

Proposition 4.3: Given a resource allocation setting $\langle A,R,U\rangle$ with the preference representations in U in the bundle form and particular allocation P which allocates a bundle $B_a \subseteq$ R to each agent a, for any answer sets $S \in \mathcal{AS}(\Pi_{BF}(A,R,U) \cup \Pi_P)$,

`utility(`a`,`v`)` $\in S$ if and only if $\langle B_a,v\rangle \in ur_a$, and `utility(`$a$`,0)` $\in S$ if and only if $\langle B_a,v\rangle \notin ur_a$ for any value of v.

Proof: For every agent a which is allocated a bundle B_a in P such that $\langle B,v\rangle \in ur_a$, the atoms `utility(`$a$`,`$v$`)` and `has_utility(`$a$`)` $\in S \in \mathcal{AS}(\Pi_{BF}(A,R,U)\cup\Pi_P)$ from[4] the rules 1 and 2 respectively. Also a fact '`utility(`a`,0)`' $\notin S \in \mathcal{AS}(\Pi_{BF}(A,R,U)\cup \Pi_P)$ since the body of rule 4 will not be satisfied.
On the other hand, for every agent a which is allocated a bundle B_a in P such that $\langle B,v\rangle \notin ur_a$ for any value of v, an atom `utility(`a`,0)` $\in S \in \mathcal{AS}(\Pi_{BF}(A,R,U)\cup\Pi_P)$ from rule 4, since `utility(`a`,`v`)` $\notin S \in \mathcal{AS}(\Pi_{BF}(A,R,U)\cup\Pi_P)$ for any value of v except 0 as non of the bodies of rules 1 is satisfied, and consequently the body 2 is not satisfied. ■

Since the formalizations of MARA problems later on define aggregations or weak constrains (optimization statements in LParse) on the `utility` /2 atoms, it is often the case that an agent a having a 0 utility explicitly represented with an atom `utility(`a`,0)` in the answer sets, is equivalent to not having any `utility` /2 atom for the agent a. Accordingly, rules 2 and 3 can be removed. Such translation of the bundle preference representation is referred to as Π_{BFS}.

For the extended preferences, the bundle form is formalized with the translation $\Pi_{BFex}(A,R,U)$ defined as follows:

- For every $a \in A$, and for every tuple $\langle B, v\rangle \in ur_a$, a rule $rl \in \Pi_{BFex}(A,R,U)$ where head(rl) =`utility(`a`,`v`,A)`, and for every resource r, if $r \in B$ then '`has(`a`,A)`' $\in$ body(rl); else `not has(`a`,A)` $\in$ body(rl), and `agent(A)` $\in$ body(rl).
- The rules:
```
has_utility(A,B):- utility(A,U,B), U!=0.
utility(A,0,B):-
     agent(A),agent(B),not has_utility(A,B).
```
$\in \Pi_{BFex}(A,R,U)$

Proposition 4.4: Given a resource allocation setting $\langle A,R,U\rangle$ with the preference representation U in the bundle form and particular allocation P which allocates a bundle $B_a \subseteq R$ to each agent a, for any answer sets $S \in \mathcal{AS}(\Pi_{BFex}(A,R,U)\cup \Pi_P)$, `utility(`$a$`,`$v$`,`$b$`)` $\in S$ if and only if $\langle B_b,v\rangle \in ur_a$, and `utility(`$a$`,0,`$b$`)` $\in S$ if and only if $\langle B_b,v\rangle \notin ur_a$ for any value of v.

Proof: Analogous to the one for Π_{BF}. ■

4.2.3 K-additive preferences

In the k-additive preference representation the utility of a bundle is represented as the sum of the coefficients assigned to its subsets (with size at most k). Similar to the bundle

[4]When it is stated that a set of atoms belong to an answer set from a certain rule, it is because of **proposition 3.1.a**. in the preceding chapter. This is common for all proves.

form, if a coefficient (the synergetic value of having the resources in the bundle together) is not stated for a particular bundle it is assumed to be 0. Formally, if preferences are given in the k-additive form, for every agent $a \in A$, and for every bundle $B \subseteq$ R such that the synergetic value of having the resources in that bundle is not 0, there is a unique coefficient α such that $\langle B,\alpha\rangle \in ur_a$. Note that k can be induced from the specification of the coefficients as the cardinality of the largest bundle B for which there is a coefficient α such that $\langle B,\alpha\rangle \in ur_a$. In order to formalize k-additive utilities in ASP, an identifier should be defined for each tuple $\langle B,\alpha\rangle \in ur_a$ in order to differentiate (when aggregating) between two coefficients with the same value. For example, if there are n tuples $\langle B,\alpha\rangle \in ur_a$, the numbers from 0 to n can be used as identifiers.

Accordingly, the k-additive representation for the simple preferences, is formalized in ASP with the following translation $\Pi_{KA}(A,R,U)$:

- A rule for grounding the utilities, given the maximum utility an agent can have $u_{\max}$.

  `utg(0..`$u_{\max}$`).`

- For every $a \in A$, and for every tuple $\langle B,\alpha\rangle \in ur_a$ with an identifier i, a fact `coefg(`a`,`α`,`i`)` $\in \Pi_{KA}(A,R,U)$, and a rule $rl \in \Pi_{KA}(A,R,U)$, where head(rl) =`coef(`a`,`α`,`i`)`, and for every resource $r \in B$ '`has(`a`,` r`)`' $\in$ body(rl).

  (1)

- And a rule for summing the coefficients:

  in DLV

  ```
  utility(A,U):-utg(U),agent(A),#sum{C,I:coef(A,C,I)}=U.
  ```

  $\in \Pi_{KA}(A,R,U)$ (2)

  in LParse

  ```
  utility(A,U):-
       U[coef(A,C,I):coefg(A,C,I)=C]U,utg(U),agent(A).
  ```

  $\in \Pi_{KA}(A,R,U)$ (2)

  Consider a single agent scenario with an agent a, a set of resources $R=\{r_1, r_2, r_3\}$, and a set of k-additive utility representations $ur_a = \{\langle\{r_1\},2\rangle, \langle\{r_1,r_2\},2\rangle, \langle\{r_3\},1\rangle\}$. The resulting program in DLV $\Pi_{KA}(\{a\},\{r_1, r_2, r_3\},\{ur_a\})$ consists of the following rules:

  ```
  coef(a,2,1):- has(a, r1).
  coef(a,2,2):- has(a, r1),has(a, r2).
  coef(a,1,3):- has(a, r3).
  utility(A,U):-utg(U),agent(A),#sum{C,I:coef(A,C,I)}=U.
  ```

Proposition 4.5: Given a resource allocation setting $\langle A,R,U\rangle$ with the preference representations in U given in the k-additive form and a particular allocation P which allocates a bundle $B_a \subseteq R$ to an each agent a, for any answer sets $S \in \mathcal{A}s(\Pi_{KA}(A,R,U)\cup\Pi_P)$, `utility(`$a$`,`$v$`)` $\in S$ if and only if v is the sum of all coefficients α such that $\langle B,\alpha\rangle \in ur_a$ and $B \subseteq B_a$.

Proof: For every agent a which is allocated a bundle B_a in P, and for every bundle $B \subseteq B_a$ such that $\langle B,\alpha\rangle \in ur_a$, an atom `coef(`$a$`,`$\alpha$`,`$i$`)` $\in S \in \mathcal{AS}(\Pi_{KA}(A,R,U) \cup \Pi_P)$ from rules 1 (where i is the identifier of the tuple $\langle B,\alpha\rangle$). Then an atom `utility(`a`,`v`)` $\in S$ from rule 2, where v is the sum of all coefficients α such that the atom `coef(`a`,`α`,`i`)` $\in S$ (by the semantics of the #sum aggregate function/ the semantics of weight constraints).■

For the extended preferences, the k-additive form is formalized in ASP with the translation $\Pi_{KAex}(A,R,U)$ defined as follows

- A rule for grounding the utilities, given the maximum utility an agent can have $u_{\max}$.

  `utg(0..`$u_{\max}$`).`

- For every $a \in A$, and for every tuple $\langle B,\alpha\rangle \in ur_a$ with an identifier i, a fact `coefg(`a`,`α`,`i`)` $\in \Pi_{KAex}(A,R,U)$, and a rule $rl \in \Pi_{KAex}(A,R,U)$, where head(rl) =`coef(`a`,`α`,`i`,A)`, and for every resource $r \in B$ '`has(`A`,` r`)`' $\in$ body(rl), and `agent(A)` $\in$ body(rl). (1)

- And a rule for summing the coefficients:

  in DLV
  ```
  utility(A,U,B):-
        utg(U),agent(A),agent(B),
        #sum{C,I:coef(A,C,I,B)}=U.
  ```
  $\in \Pi_{KAex}(A,R,U)$ (2)

  in LParse
  ```
  utility(A,U,B):-
        U[coef(A,C,I,B):coefg(A,C,I)=C]U,
        utg(U),agent(A)agent(B).
  ```
  $\in \Pi_{KAex}(A,R,U)$ (2)

Thus for the example above the translation $\Pi_{KA'}$ consist of the following rules:

```
coef(a,2,1,A):- has(A, r1), agent(A).
coef(a,2,2,A):- has(A, r1),has(A, r2), agent(A).
coef(a,1,3,A):- has(A, r3), agent(A).
utility(A,U,B):-
      utg(U),agent(A),agent(B),
      #sum{C,I:coef(A,C,I,B)}=U.
```

Proposition 4.6: Given a resource allocation setting $\langle A,R,U\rangle$ with the preference representations in U in the k-additive form and a particular allocation P which allocates a bundle $B_a \subseteq R$ to an each agent a, for any answer sets $S \in \mathcal{AS}(\Pi_{KAex}(A,R,U) \cup \Pi_P)$, `utility(`$a$`,`$v$`,`$b$`)` $\in S$ if and only if v is the sum of all coefficients α such that $\langle B,\alpha\rangle \in ur_a$ and $B \subseteq B_b$.

Proof: Analogous to the one for Π_{KA}.■

4.2.4 Logic based preference representation of quantitative preferences

In logic based preference representation preferences are represented in terms of goals expressed as propositional formulas over literals of the form p_r (meaning that an agent owns a resource r). The utility values are defined with respect to the satisfied goals by a particular bundle. There are several ways of extracting utilities from goals each of which is formalized with ASP in this section. Unsurprisingly, logic based preferences are the easiest to express in ASP.

The simplest logic based representation is to have a single goal for each agent and a utility 1 for the bundles which satisfy the goal and 0 for the ones which don't. Formally for each agent $a \in A$, $ur_a = G_a$ where G_a is a propositional formula comprised of literals p_r where $r \in R$.

For the simple preferences, the simplest logic representation is formalized in ASP with the following translation $\Pi_{LS}(A,R,U)$:

- For every agent $a \in A$ with a goal G_a in disjunctive normal form $d_1 \vee ... \vee d_n$, for every disjunct $d_{1..n}$ which is a conjunction $p_{ri} \wedge ... \wedge p_{rj} \wedge \neg p_{rz} \wedge ... \wedge \neg p_{re}$ (meaning that a owns resources $r_i \ldots r_j$ and does not own resources $r_z \ldots r_e$) the rule

  ```
  goal(a):-
        has(a, r_i),…,has(a, r_j),
        not has(a, r_j),…,not has(a, r_e).  ∈ Π_LS(A,R,U)
  ```

- Two rules for defining the utility :

  ```
  utility(A,1):-goal(A).
  utility(A,0):-agent(A),not goal(A).  ∈ Π_LS(A,R,U)
  ```

Consider a single agent a, a set of three resources $R=\{r_1, r_2, r_3\}$, and a goal $G_a = (p_{r1} \wedge \neg p_{r2}) \vee p_{r3}$. The translation $\Pi_{LS}(\{a\},\{r_1, r_2, r_3\},\{G_a\})$ will consist of the following rules:

```
goal(a):- has(a, r1),not has(a, r2).
goal(a):- has(a, r3).
utility(A,1):-goal(A).
utility(A,0):-agent(A),not goal(A).
```

Proposition 4.7: Given a resource allocation setting $\langle A,R,U \rangle$ with the preference representations in U given as a single propositional goal, and a particular allocation P which allocates a bundle $B_a \subseteq R$ to each agent a, for any answer sets $S \in \mathcal{A}s(\Pi_{LS}(A,R,U) \cup \Pi_P)$, `utility(a,1)` $\in S$ if and only if B_a satisfies the goal of a G_a, and `utility(a,0)` $\in S$ if and only if B_a does not satisfy G_a

Proof: Trivial.

For the extended preferences, the formalization is done with the following translation $\Pi_{LSex}(A,R,U)$

- For every agent $a \in A$ with a goal G_a in disjunctive normal form $d_1 \vee ... \vee d_n$, for every disjunct $d_{1..n}$ which is a conjunction $p_{ri} \wedge ... \wedge p_{rj} \wedge \neg p_{rz} \wedge ... \wedge \neg p_{re}$ the rule

```
goal(a,A):-
      has(A, r_i),… ,has(A, r_j),
      not has(A, r_j),… ,not has(A, r_e),
      agent(A).
```
$\in \Pi_{LSex}(A,R,U)$

- The rules :

```
utility(A,1,B):- goal(A,X), agent(B).
utility(A,0,B):- agent(A), agent(B) not goal(A,X).
```
$\in \Pi_{LSex}(A,R,U)$

Proposition 4.8: Given a resource allocation setting $\langle A,R,U\rangle$ with the preference representations in U given as a single propositional goal, and a particular allocation P which allocates a bundle $B_a \subseteq R$ to each agent a, for any answer sets $S \in \mathcal{AS}(\Pi_{LS}(A,R,U) \cup \Pi_P)$, `utility(`$a$`,1,`$b$`)` $\in S$ if and only if B_b satisfies the goal of a G_a, and `utility(`a`,0,`b`)` $\in S$ if and only if B_b does not satisfy G_a

Proof: Trivial.

Another alternative is, instead of a single goal, to have a set of goals and a utility defined as the number of goals satisfied by a particular bundle. Formally, for each agent $a \in A$, $ur_a = \{G_{a1} \ldots G_{an}\}$ where each $G_{a1 \ldots an}$ is comprised of propositional literals p_r and $r \in R$.

For the simple preferences, the formalization of the above representation in ASP is done with the following translation $\Pi_{LC}(A,R,U)$:

- A rule for grounding the utilities, given the maximum utility an agent can have $u_{\max}$.

```
utg(0..u_max).
```

- For every agent $a \in A$, and for every goal G_{ai} of a in disjunctive normal form $d_1 \vee ... \vee d_n$, a fact for grounding `goalg(`a`,`i`)` $\in \Pi_{LC}(A,R,U)$ and, for every disjunct $d_{1..n}$ which is a conjunction $p_{ri} \wedge ... \wedge p_{rj} \wedge \neg p_{rz} \wedge ... \wedge \neg p_{re}$ the rule

```
goal(a,i):-
      has(a, r_i),… ,has(a, r_j),
      not has(a, r_j),… ,not has(a, r_e).
```
$\in \Pi_{LC}(A,R,U)$

- A rule for defining the utility:

  In DLV
```
utility(A,U):- utg(U), agent(A),#count{G:goal(A,G)}=U.
```
$\in \Pi_{LC}(A,R,U)$

  In Smodels
```
utility(A,U):-
      U[goal(A,G):goalg(A,G)]U,utg(U),agent(A).
```
$\in \Pi_{LC}(A,R,U)$

Proposition 4.9: Given a resource allocation setting $\langle A,R,U\rangle$ with the preference representations in U given as a set of propositional goals (i.e. for each agent $a \in A$, $ur_a = \{G_{a1} \ldots G_{an}\}$), and a particular allocation P which allocates a bundle $B_a \subseteq$ R to each agent a, for any answer set $S \in \mathcal{AS}(\Pi_{LC}(A,R,U) \cup \Pi_P)$, `utility(`$a$`,`$v$`)` $\in S$ if and only if v is the number of goals of a satisfied by the bundle B_a.

Proof: Analogous to the one for Π_{KA}.■

For the extended preferences, the formalization is done with the translation $\Pi_{LCex}(A,R,U)$ defined as follows:

- A rule for grounding the utilities, given the maximum utility $u_{\max}$ an agent can have.

  `utg(0..`$u_{\max}$`).`

- For every agent $a \in A$, and for every goal G_{ai} of a in disjunctive normal form $d_1 \vee \ldots \vee d_n$, `goalg(`a`,`i`)` $\in \Pi_{LCex}(A,R,U)$ and, for every disjunct $d_{1..n}$ which is a conjunction $p_{ri} \wedge \ldots \wedge p_{rj} \wedge \neg p_{rz} \wedge \ldots \wedge \neg p_{re}$ the rule

  `goal(`a`,`i`,A):-`
  `has(A,` r_i`),… ,has(A,` r_j`),`
  `not has(A,` r_j`),… ,not has(A,` r_e`),`
  `agent(A).` $\in \Pi_{LCex}(A,R,U)$

- A rule for defining the utility:

  In DLV
  `utility(A,U,B):-`
  `utg(U),agent(A),agent(B),`
  `#count{G:goal(A,G,B)}=U.` $\in \Pi_{LCex}(A,R,U)$
  In LParse
  `utility(A,U,B):-`
  `U[goal(A,G):goalg(A,G)]U,`
  `utg(U),agent(A),agent(B).` $\in \Pi_{LCex}(A,R,U)$

Proposition 4.10: Given a resource allocation setting $\langle A,R,U\rangle$ with the preference representations in U given as a set of propositional goals, and a particular allocation P which allocates a bundle $B_a \subseteq R$ to each agent a, for any answer set $S \in \mathcal{AS}(\Pi_{LCex}(A,R,U) \cup \Pi_P)$, `utility(`$a$`,`$v$`,`$b$`)` $\in S$ if and only if v is the number of goals of a satisfied by the bundle B_b.

Proof: Analogous to the one for Π_{KA}.■

Yet another variant of logic based preferences are the weighted goals, where a weight is associated to each goal, and the final utility is a value opposite proportional to the sum of the unsatisfied goals. Since DLV does not support negative numbers (for negating the sum of unsatisfied weights), for simplicity, the final utility is considered to be the sum of the weights of the satisfied goal. Formally, when utilities are represented with weighted goals, for each agent $a \in A$, $ur_a = \{\langle G_{a1},\alpha_{a1}\rangle \ldots \langle G_{an},\alpha_{an}\rangle\}$ where each $G_{a1\ldots an}$ is comprised

of propositional atoms p_r where $r \in R$, and $\alpha_{a1\ldots an}$ are numerical values representing the weights.

For the simple preferences represents as weighted goals the ASP formalization is done with the following translation $\Pi_{WG}(A,R,U)$:

- A rule for grounding the utilities, given the maximum utility $u_{\max}$ an agent can have.

  `utg(0..`$u_{\max}$`).`

- For every agent $a \in A$, for every goal G_{ai} in disjunctive normal form $d_1 \vee \ldots \vee d_n$ with a weight α_{ai}, `goalg(`a`,`i`,`α_{ai}`)` $\in \Pi_{WG}(A,R,U)$ and, for every disjunct $d_{1..n}$ which is a conjunction $p_{ri} \wedge \ldots \wedge p_{rj} \wedge \neg p_{rz} \wedge \ldots \wedge \neg p_{re}$ the rule:

  `goal(`a`,`i`,`α_{ai}`):-`
  `has(`a`,`r_i`),… ,has(`a`,`r_j`),`
  `not has(`a`,`r_j`),… ,not has(`a`,`r_e`).` $\in \Pi_{WG}(A,R,U)$

- And a rule for defining the utility:

  In DLV
  `utility(A,U):-utg(U),agent(A),#sum{W,G:goal(A,G,W)}=U.`
  $\in \Pi_{WG}(A,R,U)$
  In LParse
  `utility(A,U):-`
  `U[goal(A,G,W):goalg(A,G,W)=W]U,`
  `utg(U),agent(A).` $\in \Pi_{WG}(A,R,U)$

Proposition 4.11: Given a resource allocation setting $\langle A,R,U\rangle$ with the preference representations in U given as a set of propositional goals with weights, and a particular allocation P which allocates a bundle $B_a \subseteq$ R to each agent a, for any answer sets $S \in \mathcal{AS}(\Pi_{WG}(A,R,U) \cup \Pi_P)$, `utility(`$a$`,`$v$`)` $\in S$ if and only if v is the sum of the weights of the goals of a satisfied by the bundle B_a.

Proof: Analogous to the one for Π_{KA}.■

For the extended preferences represented as weighted goals the ASP formalization is done with the translation $\Pi_{WGex}(A,R,U)$ defined as follows:

- A rule for grounding the utilities, given the maximum utility an agent can have $u_{\max}$.

  `utg(0..`$u_{\max}$`).`

- For every agent $a \in A$, for every goal G_{ai} in disjunctive normal form $d_1 \vee \ldots \vee d_n$ with a weight α_a, `goalg(`a`,`i`,`α_{ai}`)` $\in \Pi_{WG}(A,R,U)$ and, for every disjunct $d_{1..n}$ which is a conjunction $p_{ri} \wedge \ldots \wedge p_{rj} \wedge \neg p_{rz} \wedge \ldots \wedge \neg p_{re}$ the rule

  `goal(`a`,`i`,`α_{ai}`,A):-`
  `has(A,`r_i`),… ,has(A,`r_j`),`
  `not has(A,`r_j`),… ,not has(A,`r_e`),`

```
agent(A).
```
$\in \Pi_{WGex}(A,R,U)$

- And the rule:

  In DLV
```
utility(A,U,B):-
      utg(U),agent(A), agent(B),
      #sum{W,G:goal(A,G,W,B)}=U.
```
$\in \Pi_{WGex}(A,R,U)$

  In LParse
```
utility(A,U,B):-
      U[goal(A,G,W,B):goalg(A,G,W)=W]U,
      utg(U),agent(A),agent(B).
```
$\in \Pi_{WGex}(A,R,U)$

Proposition 4.12: Given a resource allocation setting $\langle A,R,U\rangle$ with the preference representations in U given as a set of propositional goals with weights, and a particular allocation P which allocates a bundle $B_a \subseteq$ R to each agent a, for any answer sets $S \in \mathcal{AS}(\Pi_{WG}(A,R,U) \cup \Pi_P)$, `utility(`$a$`,`$v$`,`$b$`)` $\in S$ if and only if v is the sum of the weights of the goals of a satisfied by the bundle B_b.

Proof: Analogous to the one for Π_{KA}.■

4.2.5 Ordinal Preference representation, prioritized goals

Recall that prioritized goals are very similar to weighted goals in which, instead of weights, the goals are associated with priorities. Accordingly, for each agent $a \in A$, $ur_a = \{\langle G_{a1}\rangle \ldots \langle G_{an}\rangle, r\}$ where each $G_{a1\ldots an}$ is the same as before, and r is a function from integers to integers such that $r(i) = j$ when j is the rank of the goal G_i.

In ASP, prioritized goals are formalized with the translations $\Pi_{PG}(A,R,U)$ and $\Pi_{PGex}(A,R,U)$ (for simple and extended preferences respectively) defined as follows:

- For every agent $a \in A$, for every goal G_{ai} in disjunctive normal form $d_1 \vee \ldots \vee d_n$ with a priority $r(i) = j$ the rule:

  `goal(`a`,`i`,`j`).` $\in \Pi_{PG}(A,R,U)$ and $\Pi_{PGex}(A,R,U)$

- and for every disjunct $d_{1..n}$ which is a conjunction $p_{ri} \wedge \ldots \wedge p_{rj} \wedge \neg p_{rz} \wedge \ldots \wedge \neg p_{re}$ the rule:

```
sat(a,i,X):-
      allocation(X),
      has(a, r_i,X),… ,has(a, r_j,X),
      not has(a, r_j,X),… ,not has(a, r_e,X).
```
$\in \Pi_{PG}(A,R,U)$ (1)

  and
```
sat(a,i,A):-
      agent(A),
      has(A, r_i),… ,has(A, r_j),
      not has(A, r_j),… ,not has(A, r_e).
```
$\in \Pi_{PGex}(A,R,U)$ (1')

The formalization for the simple preferences uses `has`/3 atoms instead of `has`/2 for defining the preferences. This allows several allocations to be represented in a single

answer set depending on the third argument of the has/3 atoms. Accordingly, an alternative correspondence between an allocation P and program $\Pi_{P'}$ (to differentiate from Π_P which uses has/2 atoms) is defined as follows:

- Given an allocation P, for every agent a and bundle B such that $\langle a,B\rangle \in P$, and for every resource $r \in B$, a fact

  `has(`a`, `r`, `p`).` $\in \Pi_{P'}$

- And an additional fact

  `allocation(`p`).` $\in \Pi_{P'}$

Now, for each agent a preference relation can be defined over the allocations (or the bundles allocated to different agents in extended preferences) with respect to the ordinal preferences which are being formalized. This corresponds to an atom `prefers(`a`,`p`,`q`)` meaning that an agent a prefers, or is at least as satisfied with the received bundle in the allocation P compared to the one in allocation Q, or to an atom `prefers(`a`,`a`,`b`)` meaning that an agent a prefers, or is at least as satisfied with, the bundle allocated to itself than to the one allocated to agent b.

As discussed in chapter 2, there are three ways of extending prioritized goals to a preference relation: the best-out, the dicsrimin and the leximin orderings, all of which are discussed in what follows.

Recall the definition of the **best out** ordering from chapter 2:

$$P \preceq^{bo}_{GB} P' \text{ iff } min\{\, r(i) \mid P \not\models G_i\} \leq min\{\, r(i) \mid P' \not\models G_i\}$$

For the simple preferences, the formalization of this ordering in ASP is done with the program Π_{BO} consisting of the following rules:

- rules for finding for each agent the highest priority of a goal which is not satisfied in a particular allocation.

```
notSat(A,P,X):-
     allocation(X),goal(A,G,P),not sat(A,G,X).     (2)
notmin(A,P,X):-notSat(A,P,X),notSat(A,PP,X), PP<P. (3)
min(A,P,X):-notSat(A,P,X),not notmin(A,P,X).       (4)
```

- rules for defining the preference relation, with the latter one covering the case when all goals of an agent have been satisfied by a particular allocation.

```
prefers(A,X,Y):-
     allocation(X),allocation(Y),Y!=X,agent(A),
     min(A,PI,X),min(A,PG,Y), PI>=PG.              (5)
prefers(A,X,Y):-
     allocation(X),allocation(Y),Y!=X,agent(A),
     not hasmin(A,X).                              (6)
hasmin(A,X):-min(A,P,X).                           (7)
```

The formalization for the extended preferences is done by a program Π_{BOex} which is equal to Π_{BO} where each `allocation`/1 predicate is replaced by the predicate `agent`/1. Thus for example rule 5 becomes:

```
prefers(A,X,Y):-
     agent(X),agent(Y),Y!=X,agent(A),
     min(A,PI,X),min(A,PG,Y), PI>=PG.
```

Proposition 4.13: Given a resource allocation setting $\langle A,R,U\rangle$ with the preferences in U given as a set of propositional goals with priorities, and two (or more) allocations P and Q. An atom `prefers(`a,p,q`)` $\in S \in \mathcal{AS}(\Pi_{PG}(A,R,U) \cup \Pi_{BO} \cup \Pi_P \cup \Pi_Q)$ if and only if the agent a prefers P over Q in respect to the best out ordering over the prioritized goals (i.e. $Q \preceq^{bo}_{GB} P$). Similarly given allocation P, `prefers(`a,b,c`)` $\in S \in \mathcal{AS}(\Pi_{PGex}(A,R,U) \cup \Pi_{BOex} \cup \Pi_P)$ if and only if the agent a prefers the bundle allocated to agent b compared to the one allocated to agent c (i.e. $B_a \preceq^{bo}_{GB} B_b$).

Proof: From rule 1, for every goal G_{ai} of an agent a which is satisfied by an allocation P (i.e. $P \models G_{ai}$) an atom `sat(`a, i, p`)` $\in S \in \mathcal{AS}(\Pi_{PG}(A,R,U) \cup \Pi_{BO} \cup \Pi_P \cup \Pi_Q)$.From rule 2, for every priority j of an unsatisfied goal of an agent a in an allocation P an atom `notSat(`a, j, p`)` $\in S$. Then, from rules 3 and 4, an atom `min(`a, m, p`)` $\in S$, where m is the minimal priority value (the highest priority) of an unsatisfied goal of an agent a in the allocation P, and thus `pref(`a,p,q`)` $\in S$ iff $min\ \{r(i) \mid P \not\models G_{ai}\} \leq min\{\ r(i) \mid Q \not\models G_{ai}\}$ or P satisfies all a's goals, from rule 5 and rules 6 and 7 respectively. Accordingly `pref(`a,p,q`)` $\in S$ if and only if $Q \preceq^{bo}_{GB} P$ in respect to the preferences of agent a. The other case is similar. ■

The **descrimin** ordering is defined as follows:

Let $d(P, P') = \min\{r(i) \mid P \not\models G_i \wedge P' \models G_i\}$. $P \preceq^{dis}_{GB} P'$ iff $d(P, P') < d(P', P)$ or $\{G_i \mid P \models G_i\} = \{\ G_i \mid P' \models G_i\}$.

This ordering is formalized in ASP with the program Π_{DO} consisting of the following rules:

- rules for defining the $d(R, R')$ for two allocations R and R' .

```
potD(A,X,Y,D):-
     allocation(X), allocation(Y),Y!=X,goal(A,G,D),
     notsat(A,G,X),sat(A,G,Y).                        (8)
notminD(A,X,Y,D):-
     potD(A,X,Y,D),potD(A,X,Y,DD), DD<D.              (9)
d(A,X,Y,D):-potD(A,X,Y,D),not notminD(A,X,Y,D) .  (10)
```

- rules for defining the preference relation, with the latter one covering the case when the goals satisfied by one allocation include all the goals satisfied by the other allocation and thus also the case when the two allocations are equal.

```
prefers(A,X,Y):-d(A,X,Y,D),d(A,Y,X,DD), D>DD.     (11)
prefers(A,X,Y):-
     allocation(X), allocation(Y),
     Y!=X, agent(A),not hasD(A,X,Y).              (12)
```

```
hasD(A,X,Y):-d(A,X,Y,_).                                    (13)
```

Same as before, Π_{DOex} is equal to Π_{DO} where each `allocation`/1 predicate is replaced by the predicate `agent`/1.

Proposition 4.14: Given a resource allocation setting $\langle A,R,U\rangle$ with the preferences in U given as a set of propositional goals with priorities, and two allocations P and Q. An atom `prefers(`a`,`p`,`q`)` $\in S \in \mathcal{AS}(\Pi_{PG}(A,R,U)\cup\Pi_{DO}\cup\Pi_P\cup\Pi_Q)$ if and only if the agent a prefers (or at least as satisfied with) P over Q in respect to the descrimin ordering over the prioritized goals (i.e. $Q \preceq^{dis}_{GB} P$). Similarly given allocation P, `prefers(`a`,`b`,`c`)` $\in S \in \mathcal{AS}(\Pi_{PGex}(A,R,U)\cup\Pi_{DOex}\cup\Pi_P)$ if and only if the agent a prefers the bundle allocated to agent b compared to the one allocated to agent c (i.e. $B_a \preceq^{dis}_{GB} B_b$).

Proof: From rule 1, for every goal G_{ai} of an agent a which is satisfied by an allocation P (i.e. $P \models G_{ai}$) an atom `sat(`a`,` i`,` p`)` $\in S \in \mathcal{AS}(\Pi_{PG}(A,R,U)\cup\Pi_{DO}\cup\Pi_P\cup\Pi_Q)$. From rule 8, for every priority j of a goal G_{ai} of an agent a which is unsatisfied by an allocation P and satisfied by an allocation Q an atom `potD(`a`,`p`,`q`,`j`)` $\in S$. Then, from rules 9 and 10, an atom `d(`a`,`p`,`q`,`m`)` $\in S$ where m is the minimal priority value (the highest priority) such that `potD(`a`,`p`,`q`,`m`)` $\in S$ and thus $m = d(P, Q)$. An atom `prefers(`a`,`p`,`q`)` $\in S$ if and only if $d(Q,P) < d(P,Q)$, or all goals satisfied by Q are also satisfied by P, fromrule 11 and rules 12 and 13 respectively. Accordingly `prefers(`a`,`p`,`q`)` $\in S$ if and only if $Q \preceq^{dis}_{GB} P$ in respect to the preferences of agent a. The other case is analogous. ■

Recall the definition of the **leximin** ordering:

Let $d_k(P) = |\{ G_i \mid P \models G_i \wedge r(i) = k\}|$. $P \prec^{lex}_{GB} P'$ iff there exist a k such that $d_k(P) < d_k(P')$ and $\forall j < k$, $d_j(P) = d_j(P')$. $P \preceq^{lex}_{GB} P'$ iff $P \prec^{lex}_{GB} P'$ or $\forall j$, $d_j(P) = d_j(P')$.

In ASP this ordering is formalized with the logic program Π_{LO} consisting of the following rules:

- Given a priority scale form 1..n, rules for grounding and defining the priority limit (a dummy priority is added in order to avoid arithmetic for decreasing the priority level when checking that for a given a priority level k, $\forall j < k$, $d_j(R) = d_j(R')$ in rule 16 below)

  ```
  priority(1..n+1).
  priorityLimit(n+1).
  ```

- rules for defining the $d_k(R)$ for each allocations R, and every priority k. In DLV:

  ```
  dk(A,K,DK,X):-
     agent(A),allocation(X),priority(K),
     #count{G:sat(A,G,X),goal(A,G,K)}=DK.
  ```

  In Lpaese:

  ```
  dk(A,K,DK,X):-
     agent(A),allocation(X),priority(K),
     DK[sat(A,G,X):goal(A,G,K)]DK.
  ```

  (14)

- rules for defining the preference relation, with the latter one covering the case when the two allocations satisfy the same number of goals of each priority level.

```
difAboveK(A,X,Y,K):-
     priority(K),priority(KC),K>KC,
     allocation(X), allocation(Y),X!=Y,
    dk(A,KC,DKX,X),dk(A,KC,DKY,Y),DKX!=DKY.        (15)
prefers(A,X,Y):- allocation(X),allocation(Y),X!=Y,
     priority(K),dk(A,K,DKX,X),dk(A,K,DKY,Y),DKX>DKY,
     not difAboveK(A,X,Y,K).                       (16)
prefers(A,X,Y):-
     agent(A), allocation(X), allocation(Y),X!=Y,
     not difAboveK(A,X,Y,PL), priorityLimit(PL).   (17)
```

As before, Π_{LOex} is equal to Π_{LO} where each `allocation`/1 predicate is replaced by the predicate `agent`/1.

Proposition 4.15: Given a resource allocation setting $\langle A,R,U\rangle$ with the preferences in U given as a set of propositional goals with priorities, and two allocations P and Q. An atom `prefers(`a`,`p`,`q`)` $\in S \in \mathcal{AS}(\Pi_{PG}(A,R,U)\cup\Pi_{LO}\cup\Pi_{P}\cup\Pi_{Q})$ if and only if the agent a prefers P over Q in respect to the leximin ordering over the prioritized goals (i.e. $Q \preceq^{lex}_{GB} P$). Similarly given allocation P, `prefers(`a`,`b`,`c`)` $\in S \in \mathcal{AS}(\Pi_{PGex}(A,R,U) \cup \Pi_{LOex} \cup \Pi_{P})$ if and only if the agent a prefers the bundle allocated to agent b compared to the one allocated to agent c (i.e. $B_a \preceq^{lex}_{GB} B_b$).

Proof: From rule 1, for every goal G_{ai} of an agent a which is satisfied by an allocation P (i.e. $P \models G_{ai}$) an atom `sat(`a`, `i`, `p`)` $\in S \in \mathcal{AS}(\Pi_{PG}(A,R,U)\cup\Pi_{LO}\cup\Pi_{P}\cup\Pi_{Q})$. From rule 14, for every agent a, priority level k, and an allocation P `dk(`a`,`k`,`dk`,`p`)` $\in S$ where $dk = d_k(P)$ for the agent a. From rule 15, an atom `difAboveK(`a`,`p`,`q`,`k`)`, for every agent a, priority value k, and two allocations P and Q such that there is a priority level $k' > k$ for which $d_{k'}(P) \neq d_{k'}(Q)$ for the agent a. Then an atom `prefers(`a`,`p`,`q`)` $\in S$ if and only if exist a k such that $d_k(P) < d_k(Q)$ and $\forall j < k$, $d_j(P) = d_j(Q)$ or $\forall k$, $d_k(P) = d_k(Q)$ from rule 16 and rule 17 respectively. Accordingly `prefers(`a`,`p`,`q`)` $\in S$ if and only if $Q \preceq^{lex}_{GB} P$ in respect to the preferences of agent a. The other case is analogous. ■

For simplicity, $\Pi_{PG}(A,R,U)\cup\Pi_{BO}$ is referred to as $\Pi_{BO}(A,R,U)$, $\Pi_{PG}(A,R,U)\cup\Pi_{DO}$ is referred to as $\Pi_{DO}(A,R,U)$, and $\Pi_{PG}(A,R,U)\cup\Pi_{LO}$ is referred to as $\Pi_{LO}(A,R,U)$. Similarly $\Pi_{PGex}(A,R,U)\cup\Pi_{BOex}$ is referred to as Π_{BOex}, $\Pi_{PGex}(A,R,U)\cup\Pi_{DOex}$ is referred to as $\Pi_{DOex}(A,R,U)$, and $\Pi_{PGex}(A,R,U)\cup\Pi_{LOex}$ is referred to as $\Pi_{LOex}(A,R,U)$.

4.3 Social Welfare

Most of the MARA problems discussed in this chapter include the notion of social welfare which will be formalized in ASP with the translations Π_{SWut}, Π_{SWnp}, Π_{SWeg}, and Π_{SWel}, for the utilitarian, Nash, egalitarian and elitist social welfare respectively (for both Smodels and DLV). In all translations an atom `sw(`v`)` is used to denote the social welfare value v which is depended in one way or another (depending of the type of social welfare) on the agents utilities represented in ASP as a set of `utility/2`atoms.

The simplest way of representing social welfare in DLV is with the use of aggregate functions. Since the maximum social welfare value should be known for grounding and it is dependent on the number of agents n and the maximum utility value $u_{\max}$, the input of the translations is a set of agents $A = \{1...n\}$ and $u_{\max}$.

- For utilitarian social welfare, the translation $\Pi_{SWut}(A, u_{\max})$ contains the following rules:

  `swg(0..`$i_{\max}$`).` (where: $i_{\max} = n * u_{\max}$)
  `sw(SW):- swg(SW),#sum{U,A : utility(A,U)} = SW.`

- For the Nash product, the translation $\Pi_{SWnp}(A, u_{\max})$ contains the following rules:

  `swg(0..`$i_{\max}$`).` (where: $i_{\max} = u_{\max}^{n}$)
  `sw(SW):- swg(SW),#times{U,A : utility(A,U)} = SW.`

- For the egalitarian social welfare, the translation $\Pi_{SWeg}(A, u_{\max})$ contains the following rules:

  `swg(0..`$i_{\max}$`).` (where: $i_{\max} = u_{\max}$)
  `sw(SW):- swg(SW),#min{U,A : utility(A,U)} = SW.`

- For the elitist social welfare, the translation $\Pi_{SWel}(A, u_{\max})$ contains the following rules:

  `swg(0..`$i_{\max}$`).` (where: $i_{\max} = u_{\max}$)
  `sw(SW):-swg(SW),#max{U,A : utility(A,U)} = SW.`

Proposition 4.16: Given a set n of agents, and their utility values represented as a set of `utility/2` atoms Ut, with a maximum utility value $u_{\max}$, for any answer sets $S \in \mathcal{A}s(\Pi_{SWut}(n, u_{\max}) \cup \text{Ut})$, $\mathcal{A}s(\Pi_{SWnp}(n, u_{\max}) \cup \text{Ut})$, $\mathcal{A}s(\Pi_{SWeg}(n, u_{\max}) \cup \text{Ut})$, and $\mathcal{A}s(\Pi_{SWel}(n, u_{\max}) \cup \text{Ut})$, `sw(`$v$`)` $\in S$ if and only if v is the value of the utilitarian, Nash, egalitarian, and elitist social welfare respectively in respect to the utility values in Ut.

Proof: Immediate from the semantics of the aggregate functions in DLV.■

In LParse, the corresponding translations are the following, given a set of agents $A = \{1\ldots n\}$ with maximum utility $u_{\max}$:

- For utilitarian social welfare, the translation $\Pi_{SWut}(A, u_{\max})$ contains the following rules:

  `utg(0..`$u_{\max}$`).`
  `swg(0..`$i_{\max}$`).` (where: $i_{\max} = n * u_{\max}$)
  `sw(SW):-SW[utility(A,U):agent(A):utg(U)=U]SW,swg(SW).`

- For the Nash product, the translation $\Pi_{SWnp}(A, u_{\max})$ contains the following rules:

  `utg(0..`$u_{\max}$`).`
  `swg(0..`$i_{\max}$`).` (where: $i_{\max} = u_{\max}^{n}$)
  `sw(SW):- utility(1,U1),…, utility(`n`,U`n`),`
  `utg(U1),…, utg(U`n`), swg(SW),U1*…*U`n` = SW.`

- For the egalitarian social welfare, the translation $\Pi_{SWeg}(A, u_{\max})$ contains the following rules:

```
utg(0..umax).
notMin(U):-
      agent(A),utg(U),utility(A,U),agent(B),
      utg(UU),utility(B,UU),UU<U.
sw(SW):-agent(A),utility(A,U),utg(U),not notMin(U).
```

- For the elitist social welfare, the translation $\Pi_{SWel}(A, u_{\max})$ contains the following rules:

```
utg(0..umax).
notMax(U):-
      agent(A),utg(U),utility(A,U),utg(UU),
      agent(B),utility(B,UU),UU>U.
sw(SW):- agent(A),utg(U),utility(A,U),not notMax(U).
```

Proposition 4.17: Given a set of n agents, and their utility values represented as a set of `utility/2` atoms Ut, with a maximum utility value $u_{\max}$, for any answer sets $S \in \mathcal{AS}(\Pi_{SWut}(A) \cup Ut)$, $\mathcal{AS}(\Pi_{SWnp}(A) \cup Ut)$, $\mathcal{AS}(\Pi_{SWeg}(A) \cup Ut)$, and $\mathcal{AS}(\Pi_{SWel}(A) \cup$ Ut), `sw(`v`)` $\in S$ if and only if v is the value of the utilitarian, Nash, egalitarian, and elitist social welfare respectively, in respect to the utility values in Ut.

Proof: Immediate for the utilitarian social welfare and the Nash product. For the egalitarian social welfare, an atom `notMin(`v`)` $\in S \in \mathcal{AS}(\Pi_{SWeg}(A) \cup$ Ut) for every utility value v which is not minimal from the first rule, and an atom `sw(`v`)` $\in S$ with v being the utility value (since `not notMin(`v`)` must hold) of the least satisfied agent from the second rule. Similarly, for elitist social welfare, a atom `notMax(`v`)` $\in S \in \mathcal{AS}(\Pi_{SWel}(A) \cup$ Ut) for every utility value v which is not maximal, and a atom `sw(`sw`)` $\in S$ with sw being the utility value of the most satisfied agent. ■

Note that for some problems explicit representation of the social welfare as discussed in this section will not be necessary. For example, a rule for maximizing the value of utilitarian social welfare is included in a rule for maximizing the sum of agent's utility, which in the k-additive case is included in a rule for maximizing the sum of coefficients.

4.4 Generating models

In general, MARA problems are solved with ASP by generating the possible valid allocations and then eliminating the ones which do not satisfy certain criteria. Accordingly, a logic program needs to be defined for enumerating the possible allocations. The preference relations do not matter at this stage since they are only used for pruning the allocations. The following rule can be used in DLV for generating answer sets which correspond to all allocations:

```
has(X,Y) v notHas(X,Y):- agent(X),resource(Y).        (1)
```

The corresponding without disjunction, for LParse, consists of two rules:

```
has(X,Y):-agent(X),resource(Y),not notHas(X,Y).       (2)
```

```
notHas(X,Y):-agent(X),resource(Y),not has(X,Y).        (3)
```

The answer sets of the above program are the possible combination of agent-resource pairs such that the agent owns that resource (i.e. `has`/2 holds). However some of these answer sets are not valid allocations. Examples are the ones which allocate a single resource to two different agents. Such answer sets should be pruned which is done by the following constrain rule:

```
:-agent(X),agent(Z),resource(Y),has(X,Y),has(Z,Y),X!=Z.
                                                        (4)
```

or equivalent, but more efficient, the following rule:

```
notHas(X,R):-
      agent(X),agent(Z),resource(R),has(Z,R),X!=Z.      (4')
```

In some problems it is useful to eliminate also the allocations which do not distribute every resource i.e. allow only complete allocations. This is done by the following two rules:

```
allocated(R):- resource(R), has(_,R).                   (5)
:-resource (R),not allocated(R).                        (6)
```

Given a set of resources R and a set of agents A, consider $\Pi_G(A,R)$ to be the logic program consisting of $\Pi_{AR}(A,R)$ (the `agent`/1 and `resource`/1 facts) and rules 2, 3, 4, 5, 6.

Proposition 4.18.a: Given a set of m resources R, and a set of n agents A, a complete allocation P belongs to the set of all allocations $Al_{n,m}$ if and only if there exist an answer set $S \in \mathcal{A}s(\Pi_G(A,R))$ that corresponds to P i.e. for all agents $a \in A$, $r \in P(a)$ if and only if `has(`a`,` r`)` $\in S$.

Proof: *(if)* If a given allocation P is valid and complete then it corresponds to an answer set of the program $\Pi_G(A,R)$. From rule 2 and 3, every collection of `has`/2 atoms (with the complementary `notHas`/2 atoms) is an answer set of the program as long as the body of rule 4 does not hold (which is the case if P is valid i.e. there are no two agent a and b such that $P(a) \cap P(b) \neq \varnothing$, and thus `has`/2 does not hold for two different agents and one resource simultaneously) and the body of the rule 6 does not hold (which is the case if P is also complete, i.e. for every resource r there is an agent a such that $r \in P(a)$, and thus for all resources there is a `has`/2 atom and also `allocated/1` atom from rule 5). Accordingly there exist $S \in \mathcal{A}s(\Pi_G(A,R))$ which corresponds to P.
(only if) Analogous. ■

A more efficient way to achieve the same effect as the rules above, is by direct generation of answer sets which allocate each resource to exactly one agent, a technique known as *general enumeration of exactly one* [36] which was used in section 3.3. This corresponds to the following two rules:

```
has(X,Y):-agent(X),resource(Y),not notHas(X,Y).         (7)
notHas(X,R):-
      agent(X),agent(Z),resource(R),has(Z,R),X!=Z.      (8)
```

In LParse the same can be expressed with cardinality constrain stating that for each resource there should be exactly one agent such that the agent owns that resource.

```
1 {has(X,Y): agent(X)} 1 :- resource(Y).        (9)
```

Given a set of resources R and a set of agents A, the logic program consist of the program $\Pi_{AR}(A,R)$, and rules 7 and 8 (in case of DLV), or 9 (in case of LParse) is equivalent to the program $\Pi_G(A,R)$ discussed previously.

Proposition 4.18.b: Given a set of resources R and a set of agents A, the logic program consist of the program $\Pi_{AR}(A,R)$, and rules 7 and 8 (in case of DLV), or 9 (in case of LParse) is equivalent to the program $\Pi_G(A,R)$ discussed previously.

Proof: Trivial.

These rules (referred to as Π_G) are used extensively in the solutions discussed later in this chapter since they provide natural way of generating answer sets for a resource allocation problem. Exceptions are auctions which will be discussed later.

4.5 The Welfare Optimization Problem

The welfare optimization problem as discussed in section 2.5, is the problem of checking whether there is an allocation which exceeds a certain value in respect to a particular social welfare function. More formally:

- **Welfare Optimization (WO)**
  - Instance : $\langle A,R,U \rangle$; $K \in$ I,
  - Question: $\exists P \in Al_{n,m} : sw(P) \geq K$

It is obvious that the problem only applies for quantitative preference representation since it uses the notion of social welfare. Depending on the function used for measuring the social-welfare and the preference representation, there are different instantiations of the WO problem, all of which are captured with the following logic program $\Pi_{WO}(A,R,U,K)$ consisting of:

<u>*Data*</u>

- A translation of the utilities $\Pi_U(A,R,U)$, which depending on the preference representation used is equal to Π_{EF}, Π_{BF} (Π_{BFS} in case of utilitarian social welfare) , Π_{BF} Π_{KA}, Π_{LS}, Π_{LC} or Π_{WG} discussed in section 5.2. (recall that the atoms `agent`/1 and `resource`/1 are included in the translation of the utilities)
- A translation of the social welfare $\Pi_{SW}(A,R,U)$, which depending on the social welfare used is equal to Π_{SWut}, Π_{SWnp}, Π_{SWeg} or Π_{SWel} discussed in section 5.3.

<u>*Guess*</u>

- The rules for generating answer sets corresponding to valid allocations Π_G discussed in section 5.4.

<u>*Check*</u>

- A constraint part $\Pi_C(K)$ defined as follows:

```
:-sw(SW),SW<K.
```

Theorem 4.1: Given a resource allocation setting $\langle A,R,U\rangle$ with quantitative agents' preferences (in one of the representations discussed in section 5.2) and a integer value K, each answer set $S \in \mathcal{AS}(\Pi_{WO}(A,R,U,K))$ corresponds to an allocation P with $sw(P) \geq K$ (in respect to one of the social welfare function discussed in section 5.3), thus there exist an answer set $S \in \mathcal{AS}(\Pi_{WO}(A,R,U,K))$ if and only if there exist an allocation $P \in Al_{n,m}$, such that $sw(P) \geq K$,.

Proof: From for the rules in Π_G, each answer set $S \in \mathcal{AS}(\Pi_{WO}(A,R,U,K))$ corresponds to a valid allocation $P \in Al_{n,m}$, and `sw(`v`)` $\in S$ where $v = sw(P)$, from the rules in $\Pi_U(A,R,U)$ and $\Pi_{SW}(A,R,U)$). Also $v \geq K$ because of the constraint rule, and thus the answer sets of $\Pi_{WO}(A,R,U,K)$ correspond to the valid allocations $P \in Al_{n,m}$ with $sw(P) \geq K$. Note that the allocations generated by Π_G are not directly (without additional rules) influenced by the programs $\Pi_U(A,R,U)$ and $\Pi_{SW}(A,R,U)$) since none of the literals used in Π_G appears in the head of a rule in the latter two programs. For the same reason, rules in $\Pi_U(A,R,U)$ are not influenced by rules in $\Pi_{SW}(A,R,U)$. This is common for most of the proofs which follow. ■

The above formalization can be simplified in case of the egalitarian and elitist social welfare. In the case of egalitarian social welfare, instead of defining the social welfare as the utility value of the least satisfied agent and then ensuring that it is bigger or equal to K, it can be simply stated that there should be no agent with utility less than K with the following constraint rule.

```
:-utility(A,X),X<K.
```

Similarly for the elitist social welfare it can be stated that an agent must exist with a utility value greater than K with the following two rules.

```
exist:-utility(A,X),X>=K.
:-not exist.
```

These two translations are referred to as Π_{WOeg} and Π_{WOel} respectively.

Until now the decision version of the WO problem was discussed. Although the ASP formalization results in an allocation which exceeds a certain value of social welfare, still the essence of the problem was checking whether such an allocation exist. On the other hand, the optimization version of the WO problem (WOO) is the problem of finding an allocation which maximizes the value of social welfare. This problem is outside the NP and coNP complexity classes and it can be solved with the use of optimization statements (in the case of LParse), or with weak constraints (in case of DLV). In fact the formalization is very similar to the one above with the difference that instead of constraint, an optimization statement or a weak constraint is used.

Accordingly the WOO problem is formalized in ASP with the program $\Pi_{WOO}(A,R,U)$ in which the Data and the Guess parts are the same as above (as $\Pi_{WO}(A,R,U)$), while the Check part Π_C consists of the following weak constrain in DLV which minimizes the sum of not accepted social welfares, thus maximizing the single one which is accepted.

```
:~ swg(X),not sw(X).[X:1]
```

or the following optimization statement in LParse

```
maximize[sw(X):swg(X)=X].
```

Theorem 4.2: Given a resource allocation setting $\langle A,R,U\rangle$ with quantitative agents' preferences (in one of the representations discussed in section 5.2) , an optimal answer set[5] $S \in \mathcal{AS}(\Pi_{WOO}(A,R,U,K))$ corresponds to an allocation P which maximizes the value of social welfare (i.e. there does not exist $Q \in Al_{n,m}$, $sw(P) < sw(Q)$).

Proof: Analogous to the previous, only, the value v such that $sw(v) \in S \in \mathcal{AS}(\Pi_{WOO}(A,R,U))$ must be maximized from the meaning of the weak constraint or the optimization statement. Recall that DLV minimizes the sum of the weights of the satisfied weak constraints in the answer set, and minimizing the unsatisfied atoms `sw(`v`)` with a respective weight v (the constraint Π_C), maximizes the sum of the satisfied one. Since only a single atom $sw(v)$ is satisfied (i.e. $sw(v) \in S$) the value of v is maximized.

While the above formalization is convenient for the Nash, egalitarian, and elitist social welfare, for the utilitarian social welfare it can be simplified resulting in much more efficient solution. In essence, social welfare does not need to be explicitly represented since maximizing the sum of agents utilities will lead to an allocation that maximizes utilitarian social welfare. Thus, the WOO problem for the utilitarian social welfare can be formalized with the logic program $\Pi_{WOOut}(A,R,U)$ consisting of:

Data

- A translation of the utilities $\Pi_U(A,R,U)$, which depending on the preference representation used is equal to Π_{EF}, Π_{BFS}, Π_{KA}, Π_{LS}, Π_{LC} or Π_{WG} discussed in section 5.2. (recall that the atoms `agent`/1 and `resource`/1 are included in the translation of the utilities)
- A set of facts for grounding the utility values, given the maximum utility $u_{\max}$:

  `utg(0..`$u_{\max}$`).`

Guess

- The rules for generating answer sets corresponding to valid allocations Π_G discussed in section 5.4.

Check

- A constraint part Π_C which in DLV consists of the following weak constraint:

  ```
  :~ utg(U), agent(A), not utility(A,U). [U:1]
  ```

  in LParse Π_C consists of the following optimization statement

  ```
  maximize[utility(A,U):agent(A):utg(U)=U].
  ```

[5]By optimal answer sets for LParse it is referred to the answer sets with maximum or minimum sum of the weights of the literals in the corresponding optimization statement, while for DLV the answer set which minimize the sum of weights of the satisfied weak constraints. Recall from section 3.5.1 and 3.5.2 the definition of optimization statements in LParse and weak constraints in DLV.

Theorem 4.3: Given a resource allocation setting $\langle A,R,U\rangle$ with quantitative agents' preferences (in one of the representations discussed in section 5.2) , an optimal answer set $S \in \mathcal{AS}(\Pi_{WOOut}(A,R,U))$ corresponds to an allocation P which maximizes the value of utilitarian social welfare (i.e. there does not exist $Q \in Al_{n,m}$, $sw(P) < sw(Q)$).

Proof: Contained in the proof of $\Pi_{WO}(A,R,U,K))$, only, the sum of values u such that `utility` $(a,u) \in S \in \mathcal{AS}(\Pi_{WOOut}(A,R,U))$, where a is any agent, must be maximized from the meaning of the weak constraint or the optimization statement. ■

The translation Π_{WOOut} can be further simplified when utilities are given in the K-additive form, as the cardinality of satisfied goals, or as weighted goals. In the case of the K-additive preferences maximizing the sum of satisfied agents' coefficients will result in an allocation which maximizes utilitarian social welfare. Similarly, the cardinality, or the sum of weights, of the satisfied goals in the latter two cases.

Accordingly, the optimization version of the WO problem in case of k-additive utilities is solved by the ASP with the program $\Pi_{WOOKAut}(A,R,U)$ consisting of:

Data

- For every $a \in A$, and for every tuple $\langle B, \alpha\rangle \in ur_a$ with an identifier i, a fact for grounding `coefg(`a`,`α`,`i`)` $\in \Pi_{WOOKAut}(A,R,U)$, and a rule rl, where head(rl)=`coef(`a`,`α`,`i`)`, and for every resource $r \in B$ the literal `has(`a`,` r`)` $\in$ body(rl).

Guess

- The rules for generating answer sets corresponding to valid allocations Π_G discussed in section 5.4.

Check

A constraint part Π_C consisting of the following weak constraint in DLV:

`:~ coefg(A,U,I),not coef(A,U,I). [U:1]` $\in \Pi_{WOOKAut}(A,R,U)$

in LParse Π_C consists of the following optimization statement

`maximize[coef(A,U,I):coefg(A,U,I)=U].` $\in \Pi_{WOOKAut}(A,R,U)$,

Theorem 4.4: Given a resource allocation setting $\langle A,R,U\rangle$ with k-additive utilities, an optimal answer set $S \in \mathcal{AS}(\Pi_{WOOKAut}(A,R,U))$ corresponds to an allocation P which maximizes the value of utilitarian social welfare.

Proof: Contained in the previous proofs.

Note that, the above formalization when used in LParse applies for any utility function (not only monotonic), since, unlike DLV, LParse supports negative intigers.

For the second type of logic representation, were the cardinality of satisfied goals is the final utility, a program $\Pi_{WOOLCut}(A,R,U)$ for solving the WOO problem is defined as follows.

Data

- For every agent $a \in A$, and for every goal G_{ai} of a in disjunctive normal form $d_1 \vee ... \vee d_n$, a fact for grounding `goalg(a,i)` $\in \Pi_{WOOLCut}(A,R,U)$, and for every disjunct $d_{1..n}$ which is a conjunction $p_{ri} \wedge ... \wedge p_{rj} \wedge \neg p_{rz} \wedge ... \wedge \neg p_{re}$ the rule

```
goal(a,i) :-
      has(a, r_i) ,… ,has(a, r_j) ,
      not has(a, r_j) ,… ,not has(a, r_e) .
```
$\in \Pi_{WOOLCut}(A,R,U)$

Guess

- The rules for generating answer sets corresponding to valid allocations Π_G discussed in section 5.4.

Check

- A constraint part Π_C consisting of the following weak constraint in DLV:

  `:~ goalg(A,I), not goal(A,I). [1:1]` $\in \Pi_{WOOLCut}(A,R,U)$

  in LParse Π_C consists of the following optimization statement

  `maximize[goal(A,I):goalg(A,I)].` $\in \Pi_{WOOLCut}(A,R,U)$

Theorem 4.5: Given a resource allocation setting $\langle A,R,U \rangle$ with utilities represented as a set of goals, and the final utility being the cardinality of the satisfied goals, an optimal answer set $S \in \mathcal{A}s(\Pi_{WOOLCut}(A,R,U))$ corresponds to an allocation P which maximizes the value of utilitarian social welfare.

Proof: Contained in the previous proofs.

When utilities are given as weighted goals, a program $\Pi_{WOOWGut}(A,R,U)$ for solving the WOO problem is defined as follows.

Data

- For every agent $a \in A$, for every goal G_{ai} in disjunctive normal form $d_1 \vee ... \vee d_n$ with a weight α_{ai}, a fact for grounding `goalg(a,i,` α_{ai}`)` $\in \Pi_{WOOWGut}(A,R,U)$, and for every disjunct $d_{1..n}$ which is a conjunction $p_{ri} \wedge ... \wedge p_{rj} \wedge \neg p_{rz} \wedge ... \wedge \neg p_{re}$ the rule:

```
goal(a, i, α_ai) :-
      has(a,r_i) ,… ,has(a, r_j) ,
      not has(a, r_j) ,… ,not has(a, r_e) .
```
$\in \Pi_{WOOWGut}(A,R,U)$

Guess

- The rules for generating answer sets corresponding to valid allocations Π_G discussed in section 5.4.

Check

- A constraint part Π_C consisting of the following weak constraint in DLV:

  `:~ goal(A,I,U), not goalg(A,I,U). [U:1]` $\in \Pi_{WOOWGut}(A,R,U)$

  in LParse Π_C consists of the following optimization statement

`maximize[goal(A,U,I): goalg(A,I,U)=U].` $\in \Pi_{WOOWGut}(A,R,U)$

Theorem 4.6: Given a resource allocation setting $\langle A,R,U\rangle$ with utilities represented as weighted goals. An optimal answer set $S \in \mathcal{AS}(\Pi_{WOOWGut}(A,R,U))$ corresponds to an allocation P which maximizes the value of utilitarian social welfare.

Proof: Contained in the previous proofs.

In a similar manner to the above, the decision version of the WO problem for the utilitarian social welfare can be solved without explicit aggregation, but with the use of weight constraints (in LParse). This is done by replacing the constraint rule in the above translations $\Pi_{WOOBFut}(A,R,U)$, $\Pi_{WOOKAut}(A,R,U)$, $\Pi_{WOOLCut}(A,R,U)$, and $\Pi_{WOOWGut}(A,R,U)$ with the following rules respectively. Given an input value K, and a value K'=K-1:

For the bundle form: `:- [utility(A,U):agent(A):utg(U)=U] K'.`

For the k-additive form: `:- [coef(A,U,I):coefg(A,U,I)=U] K'.`

For cardinality of goals: `:- [goal(A,I):goalg(A,I)] K'.`

For the weighted goals: `:- [goal(A,U,I): goalg(A,I,U)=U] K'.`

As will become apparent later on, the translations defined above (denoted by $\Pi_{WOBFut}(A,R,U)$, $\Pi_{WOKAut}(A,R,U)$, $\Pi_{WOLCut}(A,R,U)$, and $\Pi_{WOWGut}(A,R,U)$), are much more efficient compared to the translation Π_{WO} previously defined.

In DLV, with the use of priorities on weights, a translation Π_{WOOlo} can be defined for returning the leximin optimal solution (defined in section 2.3) in a resource allocation problem. Given a resource allocation setting $\langle A,R,U\rangle$, $\Pi_{WOOlo}(A,R,U)$ consists of the following:

Data

- A translation of the utilities $\Pi_U(A,R,U)$, which depending on the preference representation used is equal to Π_{EF}, Π_{BF} , Π_{BF} Π_{KA}, Π_{LS}, Π_{LC} or Π_{WG} discussed in section 5.2. (recall that the atoms `agent`/1 and `resource`/1 are included in the translation of the utilities)
- A fact for defining the maximum integer value $m = u_{\max}+1$, where $u_{\max}$ is the maximum utility value. This is necessary for using arithmetic in the check part below.

  `#maxint=` m`.`

Guess

- The rules for generating answer sets corresponding to valid allocations Π_G discussed in section 5.4.

Check

- A constraint part Π_C for setting for each `utility/2 atom`, a weight value 1 on a level opposite proportional to the utility value, defined as follows:

  `:~ utility(A,U),#maxint=U+P. [1:P]`

Theorem 4.7: Given a resource allocation setting $\langle A,R,U\rangle$ with quantitative agents' preferences (in one of the representations discussed in section 5.2). An optimal answer set $S \in \mathcal{AS}(\Pi_{WOOlo}(A,R,U))$ corresponds to an allocation P which is leximin optimal (as defined in section 2).

Proof: As before, each answer set $S \in \mathcal{AS}(\Pi_{WOOlo}(A,R,U,K))$ corresponds to a valid allocation $P \in Al_{n,m}$, from the rules in Π_G. By the proofs for the formalizations of quantitative preference, for each agent a there will be exactly one `utility`/2 atom belonging to each answer set $S \in \mathcal{AS}(\Pi_{WOOlo}(A,R,U))$. Recall from section 4.5.2, that in presence of priorities, DLV compares the answer set with respect to the violated weak constraints of the highest priority value, then the second highest and so on. The weak constraints defined by the rule Π_C (its ground instantiations) set the priority value opposite proportional to the utility value in each `utility`/2 atom, i.e. the smaller the utility, the higher the priority. Accordingly, DLV will firstly minimize the number of `utility`/2 atoms with the smallest utility value, then the second smallest, and so on, thus returning a leximin optimal allocation as defined in section 2.3. ■

4.6 The Welfare Improvement Problem (WI)

The welfare improvement is the problem of checking whether an allocation is optimal with respect to a particular social welfare function. In order to solve this problem in ASP the social welfare, and thus also the agents' utilities should be defined not only for the allocation that is generated, but also for the allocation which is given as input. This can be done by including an extra argument to the `has`/2, `utility`/2 and `sw`/1 predicates that will serve as a flag for differentiating between the input allocation and the one which is guessed in the logic program (similarly to what was done in the case of ordinal preferences). Thus, for example `has(`*a*`,`*r*`,`*i*`)` will mean that, in the allocation which is given as input, resource *r* is allocated to the agent *a*, and `has(`*a*`,`*r*`,`*g*`)` will mean the same for the allocation which is generated by the logic program. Similarly, the predicates `utility(`*a*`,`*v*`,`*i*`)` and `utility(`*a*`,`*v*`,`*g*`)` will represent the utility of an agent *a* in the corresponding allocations, wile `sw(`*sw*`,`*i*`)` and `sw(`*sw*`,`*g*`)` will represent the social welfare. Accordingly, the transitions of the preference representation, the translation of the social welfare function as well as the program for generating allocations will need to be modified before used in the solution of this problem.

The utility translations Π_{EF}, Π_{BF}, Π_{BFS},Π_{KA}, Π_{LS}, Π_{LC} and Π_{WG} are modified by adding two facts `allocation(`*i*`)` and `allocation(`*g*`)` for denoting the two allocations, and then adding a fresh variable symbol 'X' as the last argument of all `has`/2, `utility`/2, `coef`/2 (in the k-additive preferences) and `goal`/2 predicates(in logic based preferences) and also adding the predicate `allocation` (X) in the body of each rule. For example, the rules:

```
utility(a,3):-not has(a, r1),has(a, r2).
goal(a):- has(a, r3).
coef(a,1,3):- has(a, r3).
utility(A,U):-agent(A),#sum{C,I: coef(A,C,I)}=U.
utility(A,U):-agent(A), #count{G:goal(A,G)}=U.
```

become

```
utility(a,3,X):-
     not has(a, r1,X),has(a, r2,X),allocation (X).
goal(a,X):- has(a, r3,X), allocation(X).
coef(a,1,3,X):- has(a, r3,X), allocation(X).
utility(A,U,X):-
     agent(A),#sum{C,I:coef(A,C,I,X)}=U,allocation (X).
utility(A,U,X):-
     agent(A),#count{G:goal(A,G,X)}=U, allocation (X).
```

The same is done with the predicates utility/2, notMin/1, notMax/1 and sw/1 and the rules of the social welfare translations Π_{SWut} Π_{SWnp}, Π_{SWeg} and Π_{SWel}. For example, the rules:

```
notMin(U):-
     agent(A),utility(A,U),agent(B),
     utility(B,UU),UU<U.
sw(SW):-agent(A),utility(A,U),not notMin(U).
sw(SW):- swg(SW),#max{U,A : utility(A,U)} = SW.
```

become

```
notMin(U,X):-
     agent(A),utility(A,U,X),agent(B),
     utility(B,UU,X),UU<U,allocation (X).
sw(SW,X):-
     agent(A),utility(A,U,X),
     not notMin(U,X),allocation(X).
sw(SW,X):-
     swg(SW),#max{U,A :utility(A,U,X)} = SW,
     allocation (X).
```

The rules for generating allocations Π_G are modified by adding the constant '*g*' as a third argument of all has/2 and notHas/2 predicates. For example, the rules:

```
has(X,Y):-agent(X),resource(Y),not notHas(X,Y).
notHas(X,Y):-
     agent(X),agent(Z),resource(Y),has(Z,Y),X!=Z.
1 {has(X,Y): agent(X)} 1 :- resource(Y).
```

become

```
has(X,Y,g):-agent(X),resource(Y),not notHas(X,Y,g).
notHas(X,Y,g):-
     agent(X),agent(Z),resource(Y),has(Z,Y,g),X!=Z.
1 {has(X,Y,g): agent(X)} 1 :- resource(Y).
```

The new translations are denoted as $\Pi_{EF'}$, $\Pi_{BF'}$, $\Pi_{BFS'}$, $\Pi_{KA'}$, $\Pi_{LS'}$, $\Pi_{LC'}$, $\Pi_{WG'}$, $\Pi_{SWut'}$, $\Pi_{SWnp'}$, $\Pi_{SWeg'}$, $\Pi_{SWel'}$, and $\Pi_{G'}$. The ASP solution of the WI problem for different preference representations and different social welfare functions is formalized with the following logic program $\Pi_{WI}(A,R,U,P)$ consisting of:

Data

- A translation of the utilities $\Pi_U(A,R,U)$, which depending on the preference representation used is equal to $\Pi_{EF'}$, $\Pi_{BFS'}$ ($\Pi_{BF'}$ is only necessary for the egalitarian social welfare), $\Pi_{KA'}$, $\Pi_{LS'}$, $\Pi_{LC'}$ or $\Pi_{WG'}$.
- A translation of the social welfare $\Pi_{SW}(A,R,U)$, which depending on the social welfare used is equal to $\Pi_{SWut'}$, $\Pi_{SWnp'}$, $\Pi_{SWeg'}$ or $\Pi_{SWel'}$.
- The translation of the input allocation P consisting of the following facts:

  `allocation(i).`

  for every agent-bundle pair $\langle a, B_a\rangle \in P$, and for every resource $r \in B_a$, the fact

  `has(`*a*`,` *r* `,i).`

Guess

- The rules for generating answer sets corresponding to valid allocations $\Pi_{G'}$.

Check

- A constraint rule Π_C for eliminating the generated allocations with social welfare less or equal to the social welfare of the allocation given as input:

```
:-sw(SWI, i), sw(SWG,g),SWI >=SWG.
```

Theorem 4.8: Given a resource allocation setting $\langle A,R,U\rangle$ with quantitative agents' preferences and an allocation P, the answer sets $S \in \mathcal{AS}(\Pi_{WI}(A,R,U,P))$, correspond to the allocations Q with $sw(Q)>sw(P)$, i.e. there exist an answer set $S \in \mathcal{AS}(\Pi_{WI}(A,R,U,P))$ if and only if there exist an allocation $Q \in Al_{n,m}$, such that $sw(Q)>sw(P)$.

Proof: Each answer set $S \in \mathcal{AS}(\Pi_{WO}(A,R,U,P))$ corresponds to a valid allocation $Q \in Al_{n,m}$ from the rules in $\Pi_{G'}$. Then, `sw(`*vg*`,`*g*`)` $\in S$ where $vg = sw(Q)$ and `sw(`*vi*`,`*i*`)` $\in S$ where $vi = sw(P)$ from the rules in $\Pi_U(A,R,U)$ and $\Pi_{SW}(A,R,U)$). Also $sw(Q)>sw(P)$ because of the constrain rule Π_C. Accordingly, the answer sets of $\Pi_{WO}(A,R,U,K)$ correspond to the valid allocations $Q \in Al_{n,m}$ with $sw(Q) > sw(P)$.■

4.7 The Pareto-Optimality Problem (PO)

Pareto-optimality is the problem of checking whether a particular allocation is Pareto-optimal, i.e. it is not Pareto dominated (there is no other allocation which increases the utilities of at least one agent without decreasing the utility of the others). The main idea when solving the PO problem using ASP is that each answer set of the logic program corresponds to an allocation which Pareto dominates the allocation which is checked for Pareto-optimality. If the logic program does not have answer sets the input allocation is Pareto-optimal; else it is not and the answer sets correspond to a counter examples. Similar to the WI problem, there are two allocations present in each answer set, one which is checked whether it is Pareto-optimal (the input allocation), and one which is generated by the logic program. Thus `has`/3 and `utility`/3 atoms are used. Unlike the previous two problems which include the notion of social welfare, PO is a purely ordinal

concept applying to both quantitative and ordinal preference representations. Accordingly, what follows is the ASP solution to the decision version of the PO problem for both types of preference representations.

In the case of quantitative preference representation, the PO problem is formalized with the following logic program $\Pi_{POq}(A,R,U,P)$:

Data

- A translation of the utilities $\Pi_U(A,R,U)$, which depending on the quantitative preference representation used is equal to $\Pi_{EF'}$, $\Pi_{BF'}$, $\Pi_{KA'}$, $\Pi_{LS'}$, $\Pi_{LC'}$ or $\Pi_{WG'}$.
- The translation of the input allocation P consisting of the following facts:

  `allocation(i).`

  for every agent-bundle pair $\langle a, B_a\rangle \in P$, and for every resource $r \in B_a$, the fact

  `has(`*a*`,` *r* `,i).`

Guess

- The rules for generating answer sets corresponding to valid allocations $\Pi_{G'}$.

Check

- A constraint part Π_C for eliminating the answer sets corresponding to allocations which do not Pareto dominate the input one:

```
npd:-
   agent(A),utility(A,UG,g), utility(A,UI,i), UI>UG.
                                                  (1)
pd:-
   agent(A),utility(A,UG,g),
   utility(A,UI,i), UG>UI, not npd.               (2)
:-not pd.                                         (3)
```

Theorem 4.9: Given a resource allocation setting $\langle A,R,U\rangle$ with qualitative agents' preferences (in one of the preference representations discussed in section 5.2) and an allocation P, there exist an answer set $S \in \mathcal{AS}(\Pi_{POq}(A,R,U,P))$ if and only if P is not Pareto-optimal allocation, i.e. P is Pareto-optimal if and only if the logic program $\Pi_{PO}(A,R,U,P)$ does not have any answer sets.

Proof: It is enough to prove that the answer sets $S \in \mathcal{AS}(\Pi_{POq}(A,R,U,P))$ corresponds to the allocations Q which Pareto dominates P**.** Accordingly, from rules in $\Pi_{G'}$ each answer set $S \in \mathcal{AS}(\Pi_{POq}(A,R,U,P))$ corresponds to a valid allocation $Q \in Al_{n,m}$ and for every agent a atoms `utility(`a`,` vg `,`g`)` and `utility(` a`,` vi `,`i`)` $\in S$ where $vg=u_a(Q)$ and $vi=u_a(P)$ from rules in Π_U. Furthermore, `pd` $\in S$ if and only if there is no agent a such that $u_a(P) > u_a(Q)$ (otherwise `npd` will be derived by rule 1) and exist an agent a such that $u_a(Q) > u_a(P)$ (for deriving `pd` rule 2). Accordingly `pd` is derived if and only of Q Pareto dominates P. Because of the constrain rule 3 `pd` must belong to all answer sets of $\Pi_{POq}(A,R,U,P)$ and thus Q must Pareto dominate P. ■

The solution of the PO problem for the ordinal preferences is very similar to the one above and it is formalized by the following logic program $\Pi_{POo}(A,R,U,P)$:

Data

- A translation of the utilities $\Pi_U(A,R,U)$, which depending on the ordinal preference representation used is equal to Π_{BO}, Π_{DO}, Π_{LO}.
- The translation of the input allocation P consisting of the following facts:

  `allocation(i).`

  for every agent-bundle pair $\langle a, B_a\rangle \in P$, and for every resource $r \in B_a$, the fact

  `has(`*a*`,` *r* `,i).`

Guess

- The rules for generating answer sets corresponding to valid allocations $\Pi_{G'}$.

Check

- A constraint part Π_C for eliminating the answer sets corresponding to allocations which do not Pareto dominate the input one:

```
npd:-
   agent(A),prefers(A,i,g),
   not prefers(A,g,i).                    (1')
pd:-
   agent(A),prefers(A,g,i),
   not prefers(A,i,g), not npd.           (2')
:-not pd.                                 (3')
```

Theorem 4.10: Given a resource allocation setting $\langle A,R,U\rangle$ with agents' preferences in one of the ordinal preference representations discussed in section 5.2, and an allocation P, there exist an answer set $S \in \mathcal{A}s(\Pi_{POo}(A,R,U,P))$ if and only if P is not Pareto-optimal allocation, i.e. P is Pareto-optimal if and only if the logic program $\Pi_{POo}(A,R,U,P)$ does not have any answer sets.

Proof: From rules in $\Pi_{G'}$ each answer set $S \in \mathcal{A}s(\Pi_{POo}(A,R,U,P))$ corresponds to a valid allocation $Q \in Al_{n,m}$ and for every agent a atoms `pref` (*a*`,`*i* `,`*g*) $\in S$ if and only if a prefers (or is as satisfied with) P over Q and `pref` (*a*`,`*g* `,`*i*) $\in S$ for the reverse from rules in Π_U. Furthermore, `pd` $\in S$ if and only if there is no agent a such that a strictly prefers P over Q (otherwise `npd` will be derived by rule 1') and there exist an agent a that strictly prefers Q over P (for deriving `pd` rule 2'). Accordingly `pd` is derived if and only if Q Pareto dominates P. Because of the constrain rule 3' `pd` must belong to all answer sets of $\Pi_{POo}(A,R,U,P)$ and thus Q must Pareto dominate P. ■

The optimization version of the Pareto-optimality problem (POO) is in the Σ^P_2 complexity class and it can be captured by a strictly disjunctive logic program under the answer set semantics (and thus applying to DLV only). However, the guess and check formalization of such problems in ASP is not as intuitive as in the case of NP or CoNP problems since only restricted use of default negation is allowed in the check part of the program. In

essence, when representing Σ^P_2 problems in ASP, the minimal property of ASP is explicitly exploited by a method known as saturation [42, 44, 45]. This method forces truthfulness on atoms in the check part of the program, and thus a rule in which such an atom will be negated by default will be never satisfied, and thus will not belong in the reduct of the program. This is clearer in the proof of the correctness which follows the formal definition of the formalization for the POO problem.

For the POO problem, the translation of the bundle preferences differs from what has been presented so far and it is in correspondence with the formalization used for solving the winner determination problem in combinatorial auctions with XOR bids (presented in section 4.8 which follows). More specifically, a nonzero utility v of an agent a for a bundle b (in one to one correspondence with an XOR bid) is represented with a fact `utility(`a`,`b`,`v`)`, and each bundle b of an agent a is represented as a set of facts `in(`b`,`r`)` for each resource r in b. Allocation are then generated by accepting (allocating) or not accepting a bundle for which an agent has a nonzero utility value. Accordingly, bundles with a zero utility are not taken into consideration, which is sound since it does make sense to consider allocating to an agent a bundle for which it has a 0 utility. This might give an intuition that such a formalization offers better performance compared to the one presented in section 4.4 (in the generation all bundles are taken into account, and thus also the ones for which an agent has a 0 utility). However, as will be apparent in the next chapter in which results are discussed, this is not the case because ensuring the validity of the allocation is much easier in the latter.

Intuitively, the POO problem is formalized as follows. Guess an allocation G (a set of mutually acceptable bundles, i.e. no two bundles are accepted if they have a common element, and at most a single bundle is accepted for each agent) to be checked for Pareto-optimality and then do the check as follows: generate allocation P that potentially Pareto dominate G, and assign an atom `good` if an allocation P does not Pareto dominate G. If for every possible P an atom `good` is derived, then G is Pareto-optimal and it corresponds to an answer set of the program. If there is at least one P for which an atom `good` is not derived, then there is no answer set corresponding to G.

The POO problem for a resource allocation setting $\langle A,R,U\rangle$, with preferences in U given in the bundle form, is formalized with the translation $\Pi_{POOBF}(A,R,U)$ defined as follows:

Data

A translation of the utilities $\Pi_U(A,R,U)$, defined as follows: for every $a \in A$, and for every tuple $\langle B_i, v_i\rangle \in ur_a$ the fact `utility(`a`,`v_i`,`i`).` $\in \Pi_U(A,R,U)$ and for every resource $r \in B_i$ the fact `inBundle(`i`,`r`).` $\in \Pi_U(A,R,U)$, where i is an unique bundle identifier for a bundle B. For example the bundle identifier can be the concatenation of the order resources (or just the resource identifiers) in the bundle, i.e. if $B_i=\{r_1,r_2,r_3\}$ then i =123.

Guess

- The rules for guessing an allocation G

```
acc(A,B) v notAcc(A,B):-utility(A,B,U).          (1)
```

- A rule for not allowing two utilities to be accepted for a single agent in G.

```
notAcc(A,B):-utility(A,B,U),acc(A,BB),B!=BB.        (2)
```

- A rule for ensuring that in G the bundles of the accepted utilities do not share any resource, i.e. every resource is allocated to at most one agent.

```
notAcc(A,B):-
    utility(A,B,U),acc(AA,BB),A!=AA,
    inBundle(B,R),inBundle(BB,R).                   (3)
```

Check

- Rules for generating allocation P which potentially Pareto dominate G.

```
acc2(A,B) v notAcc2(A,B):-utility(A,B,U).           (4)
notAcc2(A,B):-utility(A,B,U),acc2(A,BB),B!=BB.      (5)
notAcc2(A,B):-
    utility(A,B,U),acc2(AA,BB),A!=AA,
    inBundle(B,R),inBundle(BB,R).                   (6)
notAcc2(A,B):-
    utility(A,B,U),utility(A,BB,UU),
    acc(A,BB),U<UU.                                 (7)
```

Rules 4,5 and 6 are analogous to 1, 2 and 3, while rule 7 ensures that the accepted utility of an agent in P is not less than the accepted utility of the same agent in G. However, there might be the case that for an agent no utilities are accepted in P while the same agent has an accepted utility in G, and thus suffer a loss in utility. This is handled by deriving the atom `good`, with rules 8 and 9.

- For every agent $a \in A$, a rule rl for denoting when an agent a does not have an accepted utility in P, defined as follows: head(rl)=`noneAcc2(`a`)`, and for every bundle B_i for which a has a specified utility (i.e. $\langle B_i, v_i \rangle \in ur_a$) `notAcc2(`$a$`,` i`)` $\in$ body(rl). (8)

- A rule for deriving `good` if an agent suffers a loss of utility in P compared to G.

```
good:-acc(A,B),noneAcc2(A).                         (9)
```

- Rules for denoting that an agent has no increase in utility in P compared to G.

```
noInrease(A):-
    acc(A,B), acc2(A,BB),
    utility (A,B,U), utility (A,BB,U).              (10)
noInrease(A):-noneAcc2(A).                          (11)
```

- Rule rl for deriving `good` if none of the agents has increased utility P compared to G, defined as follows: head(rl)=`good`, and for every agent $a \in A$, `noInrease(`a`)` $\in$ body(rl). (12)

- Rules for saturation

```
:-not good.                                         (13)
acc2(A,B):- utility(A,B,U),good.                    (14)
notAcc2(X,B,U):-utility(A,B,U),good.                (15)
```

Consider a resource allocation setting consisting of two agents and three resources $\langle\{a,b\},\{r_1, r_2, r_3\}, \{u_a, u_b\}\rangle$ with $u_a = \{\langle\{r_1\},2\rangle, \langle\{r_1, r_2\},2\rangle,\langle\{r_3\},1\rangle\}$ and $u_b = \{\langle\{r_2\},1\rangle, \langle\{r_1, r_3\},3\rangle\}$. $\Pi_{POOBF}(\{a,b\},\{r_1, r_2, r_3\}, \{u_a, u_b\})$ consists of the following rules:

Data

```
utility(a,1,2). inBundle(1,r1).
utility(a,12,2). inBundle(12,r1). inBundle(12,r2).
utility(a,3,1). inBundle(3,r3).
utility(b,2,1). inBundle(2,r2).
utility(b,13,3). inBundle(13,r1). inBundle(13,r3).
```

Guess

```
acc(A,B) v notAcc(A,B):- utility(A,B,U).
notAcc(A,B):-utility(A,B,U),acc(A,BB),B!=BB.
notAcc(A,B):-utility(A,B,U),acc(AA,BB),
             A!=AA, inBundle(B,R), inBundle(AA,BB,R).
```

Check

```
acc2(A,B) v notAcc2(A,B):- utility(A,B,U).
notAcc2(A,B):-utility(A,B,U),acc2(A,BB),B!=BB.
notAcc2(A,B):-
     utility(A,B,U),utility(A,BB,UU),acc(A,BB),U<UU.
notAcc2(A,B):-
     utility(A,B,U),acc2(AA,BB),A!=AA,
     inBundle(A,B,R), inBundle(AA,BB,R).


noneAcc2(a):- notAcc2(a,1),notAcc2(a,12),notAcc2(a,3).
noneAcc2(b):- notAcc2(b,2), notAcc2(b,13).
noInrease(A):-
     acc(A,B),acc2(A,BB),
     utility(A,B,U),utility(A,BB,U).
noInrease(A):-noneAcc2(A).

good:-acc(A,B),noneAcc2(A).
good:- noInrease(a), noInrease(b).

:-not good.
acc2(A,B):- utility(A,B,U),good.
notAcc2(X,B,U):-utility(A,B,U),good.
```

Theorem 4.11: Given a resource allocation setting $\langle A,R,U\rangle$ with agents' preferences in the bundle form, the answer sets $S \in \mathcal{AS}(\Pi_{POOBF}(A,R,U))$ restricted to acc/2 atoms correspond to the Pareto-optimal allocations $P \in Al_{n,m}$.

Proof: Consider $\Pi_{POOBF\text{-}}$ to be the translation equal to Π_{POOBF} without the rules for saturation. Because of rules in 1, each answer set $S \in \mathcal{AS}(\Pi_{POOBF\text{-}}(A,R,U))$ corresponds in terms of acc/2 atoms to an allocation G of bundles to agents, while, rules 2 and 3 assure that G is valid. Note that no other rules contain acc/2 or notAcc/2 in their head

and thus any answer set will necessarily correspond to an allocation. Similarly a valid allocation P is represented in S with the `acc2`/2 or `notAcc2`/2 atoms from rules 4, 5 and 6. Firstly it is proven that `good`$\in S$ if and only if P does not Pareto dominate G. By definition of Pareto-optimality P does not Pareto dominate G in two cases, either there is an agent who suffers a decrease in utility P, or there is no agent whose utility is increased in P compared to G. Because of rule 7, a bundle for an agent can not be accepted in P with a smaller utility than the bundle for the same agent accepted in G. Thus the only way an agent can suffer a loss in utility is when none of its bundles are accepted in P, while a bundle is accepted in G. (if) When this is the case an atom `good`$\in S$ from rule 9 (and 8). In the case when P does not increase the utility of any agent, `good`$\in S$ from rule 12 (8,10 and 11). (only if) Since there are no other rules for deriving `good` except of 9 and 12, if `good`$\in S$ then either an agent suffered a loss in utility, or the utility was not increased for any agent in P. Accordingly `good`$\in S$ if and only if P does not Pareto dominate G.
Now, assume $S \in \mathcal{AS}(\Pi_{POOBF}(A,R,U))$. Because of rule 13 `good` $\in S$, and thus for every agent bundle pair the atoms `acc2(A,B)` and `notAcc2(A,B)` $\in S$ (rule 14 and 15). The reduct of $\Pi_{POOBF}(A,R,U)$ in respect to S, $\Pi_{POOBF}^{S}(A,R,U)$ is then equal to $\Pi_{POOBF}(A,R,U)$ without rule 13. As before, because of rules 1,2 and 3, S corresponds to a valid allocation G in terms of `acc`/2 atoms. If G is not Pareto optimal, by the above, there exist a P for which `good` is not derivable. Then $G \cup P$ will be a model (with other auxiliary predicates) of $\Pi_{POOBF}^{S}(A,R,U)$ which is smaller than S, and S can not be an answer set of $\Pi_{POOBF}(A,R,U)$ since it is not minimal. On the other hand, if G a Pareto optimal allocation, then S is a minimal model of $\Pi_{POOBF}^{S}(A,R,U)$ and thus also an answer set. Accordingly, $S \in \mathcal{AS}(\Pi_{POOBF}(A,R,U))$ if and only if S corresponds to a Pareto optimal allocation. ■

Unlike other problems, the POO problem will require a separate formalization for every type of preference representation, and such formalizations are in general hard to come up with. Moreover, additional constrains containing default negations are not straightforward to be incorporated in the solutions. An alternative is to use a transformation presented in [42]. The translation combines two disjunction-free programs, a guess program (for the Pareto optimality problem, a program for guessing an allocation like $\Pi_{G'}$) and a check program (a program for checking whether an allocation is Pareto optimal, i.e. Π_{POq} or Π_{POo}), in a single program with disjunction. This approach preserves the intuitive formalization, the flexibility in terms of preference representation and additional constrains, and can also be used for solving the WOO problem without optimization statements (extending the above solution from POO to WOO is not strait forward, since the latter requires the use of aggregate functions, and thus default negation in the check part). However, it is likely to lead to inefficient solutions due to the many new predicates which are introduced in the transformation.

4.8 The Envy free problem (EF)

The envy free problem is the problem of checking whether there exist an allocation which is envy free, i.e. an allocation in which all agents prefer the bundle allocated to them compared to the bundles allocated to other agents. Similarly to PO, the EF problem is also a purely ordinal concept applying to both quantitative and ordinal preference

representation. However, in order to solve this problem it is necessary to know the agent's satisfaction not only for the bundle allocated to it, but also for the bundles allocated to other agents. Accordingly, the second type of preference formalizations will be used.

The ASP solution of the EF problem for the quantitative preferences is formalized with the logic program $\Pi_{EFq}(A,R,U)$ consisting of the following rules:

Data

- A translation of the utilities $\Pi_U(A,R,U)$, which depending on the quantitative preference representation used is equal to Π_{EFex}, Π_{BFex}, Π_{KAex}, Π_{LSex}, Π_{LCex} or Π_{WGex}.

Guess

- The rules for generating answer sets corresponding to valid allocations Π_G.

Check

- A constraint rule Π_C for eliminating the answer sets corresponding to allocations which are not envy free

```
:-agent(A),agent(B),A!=B,
utility(A,UA,A),utility(A,UB,B),UB>UA.
```

Theorem 4.12: Given a resource allocation setting $\langle A,R,U\rangle$ with agents' preferences in one of the quantitative preference representations discussed in section 5.1, the answer sets $S \in \mathcal{AS}(\Pi_{EFq}(A,R,U))$ correspond to the envy free allocations $P \in Al_{n,m}$, i.e. there exist an answer set $S \in \mathcal{AS}(\Pi_{EFq}(A,R,U))$ if and only if there exist $P \in Al_{n,m}$ such that P is envy free.

Proof: From the rules in Π_G each answer set $S \in \mathcal{AS}(\Pi_{EFq}(A,R,U))$ corresponds to a valid allocation $P \in Al_{n,m}$ and for every agent a and agent b an atom `utility`$(a,vb,b) \in S$ where vb is the utility of agent a for the bundle allocated to agent b (i.e. $vb = u_a(B_b)$) from rules in $\Pi_U(A,R,U)$. Because of the constraint rule Π_C there must not exist an agent a and an agent b such that a is more satisfied with the bundle allocated to b in P than its own bundle, and thus P must be envy free. ■

Similarly for ordinal preferences the EF problem is formalized with ASP with the logic program $\Pi_{EFo}(A,R,U)$ consisting of the following rules:

Data

- A translation of the utilities $\Pi_U(A,R,U)$, which depending on the ordinal preference representation used is equal to Π_{BOex} Π_{DOex}, Π_{LOex}.

Guess

- The rules for generating answer sets corresponding to valid allocations Π_G.

Check

- A constraint rule Π_C for eliminating the answer sets corresponding to allocations which are not envy free

```
:-agent(A),agent(B),A!=B,pref(A,B,A),not pref(A,A,B).
```

Theorem 4.30: Given a resource allocation setting $\langle A,R,U\rangle$ with agents' preferences in one of the ordinal preference representations discussed in section 5.1, the answer sets $S \in \mathcal{AS}(\Pi_{EFo}(A,R,U))$ correspond to the envy free allocations $P \in Al_{n,m}$, i.e. there exist an answer set $S \in \mathcal{AS}(\Pi_{EFo}(A,R,U))$ if and only if there exist $P \in Al_{n,m}$ such that P is envy free.

Proof: Analogous to the previous.■

In the case where an envy free solution does not exist, a common approach is to look for allocations that minimize the envy degree of the envious agents (the distance between an envious agent and the agent it envies), or minimize the number of envious agents. Both cases are optimization problems and are solved with weak constraints or optimization statements. The former, which applies only for quantitative preferences, is formalized with the logic program $\Pi_{EFOI}(A,R,U)$ consisting of:

Data

- A translation of the utilities $\Pi_U(A,R,U)$, which depending on the quantitative preference representation used is equal to Π_{EFex}, Π_{BFex}, Π_{KAex}, Π_{LSex}, Π_{LCex} or Π_{WGex}.

Guess

- The rules for generating answer sets corresponding to valid allocations Π_G.

Check

- A constraint rule Π_C for minimizing the envy degree between agents consisting of a rule for defining the envy degree of the envious agents

```
envy(A,B,Deg):-
   agent(A),agent(B),A!=B,
   utility(A,UA,A),utility(A,UB,B),UB>UA,UB-UA=Deg.
                                                    (1)
```

  and a weak constraint for minimizing the envy degree in DLV

```
:~envy(A,B,Deg).[Deg:1]
```

  or an optimization statement in LParse with a corresponding grounding of the envy degree value depended on the maximum utility an agent can receive $u_{\max}$

```
deg(0.. u_max).
minimize[envy(A,B,Deg):agent(A)
         :agent(B):deg(Deg)= Deg].
```

Theorem 4.13: Given a resource allocation setting $\langle A,R,U\rangle$ with agents' preferences in one of the quantitative preference representations discussed in section 5.1, the answer sets $S \in \mathcal{AS}(\Pi_{EFOI}(A,R,U))$ correspond to the allocations $P \in Al_{n,m}$ which minimizes the sum of the envy degrees of the envious agents.

Proof: From the rules in Π_G each answer set $S \in \mathcal{AS}(\Pi_{EFOI}(A,R,U))$ corresponds to a valid allocation $P \in Al_{n,m}$ and for every agent a and agent b an atom `utility`(a, vb, b)

$\in S$ where *vb* is the utility of agent *a* for the bundle allocated to agent *b* (i.e. $vb = u_a(B_b)$) from rules in $\Pi_U(A,R,U)$. Also for each pair of agents *a* and *b*, such that *a* envies *b* an atom `envy(`a`, `b`, `d`)` $\in S$, where *d* is the difference between the utility of *a* and the utility of *b* from rule 1. The weak constraint (the optimization statement) then ensures that the returned answer set correspond to allocations $P \in Al_{n,m}$ which minimize the sum of the envy degree of the envious agents.■

Regarding the minimization of envious agents, this is done with programs very similar to the ones above for solving the EF problem. In essence, the logic programs $\Pi_{EFO2o}(A,R,U)$ and $\Pi_{EFO2q}(A,R,U)$ that formalize this problem for quantitative preference and ordinal preferences repetitively are equal to the programs $\Pi_{EFo}(A,R,U)$ and $\Pi_{EFq}(A,R,U)$ except that the constraint rule in Π_C is replaced by an equivalent week constraint in DLV :

for quantitative preferences

```
:~agent(A),agent(B),A!=B,utility(A,UA,A),
  utility(A,UB,B),UB>UA.                    ∈ Π_C
```

for ordinal preferences

```
:~agent(A),agent(B),A!=B,pref(A,B,A),not pref(A,A,B).
                                            ∈ Π_C
```

In LParse, this is achieved with a rule for denoting the envy agents and an optimization statement:

for quantitative preferences

```
envy(A,B):-
  agent(A),agent(B),A!=B,utility(A,UA,A),
  utility(A,UB,B),UB>UA.                    ∈ Π_C
```

for ordinal preferences

```
envy(A,B):-
     agent(A),agent(B),
     A!=B,pref(A,B,A),not pref(A,A,B).      ∈ Π_C

minimize[Envy(A,B):agent(A):agent(B)]       ∈ Π_C
```

Theorem 4.14: Given a resource allocation setting $\langle A,R,U\rangle$ with agents' preferences in one of the preference representations discussed in section 5.1, the answer sets $S \in \mathcal{AS}(\Pi_{EFO2o}(A,R,U))$ (if preferences are ordinal) and the answer sets $S' \in \mathcal{AS}(\Pi_{EFO2q}(A,R,U))$ (if preferences are quantitative) correspond to the allocations $P \in Al_{n,m}$ which minimize the number of envious agents.

Proof: Analogous to the proof of Π_{EFo} and Π_{EFq}. ■

All variations of the envy free problem can be easily coupled with the WO, the WOO, and the POO problem. For the first two, consider the translation $\Pi_{WO'}(A,R,U,K)$, $\Pi_{WOO'}(A,R,U)$, and $\Pi_{WOOut'}(A,R,U)$ to be equal to $\Pi_{WO}(A,R,U,K)$, $\Pi_{WOO}(A,R,U)$, and $\Pi_{WOOut}(A,R,U)$ without the representation of preference (contained in the translation of

the envy free problem), without the part for generating answer sets (also contained in the translation of the envy free problem), and with all appearances of the binary predicates `utility`(Ag,Ut), replaced with the predicate `utility`(Ag,Ut, Ag). Then:

- $\Pi_{WO'}(A,R,U,K) \cup \Pi_{EFq}(A,R,U)$ returns envy free allocations with a value for social welfare greater than K.
- $\Pi_{WO'}(A,R,U,K) \cup \Pi_{EFO1q}(A,R,U)$ returns the allocations with the smallest envy degree, among the allocations with a social welfare value greater than K.
- $\Pi_{WO'}(A,R,U,K) \cup \Pi_{EFO2q}(A,R,U)$ returns the allocations with the least number of envious agents, among the allocations with a social welfare value greater than K.
- $\Pi_{WOO'}(A,R,U,K) \cup \Pi_{EFq}(A,R,U)$ returns the optimal envy free allocations.
- $\Pi_{WOO'}(A,R,U,K) \cup \Pi_{EFO1q}(A,R,U)$ (for DLV only), with a higher priority on the optimization statement in $\Pi_{WOO'}$, returns the allocations with the smallest envy degree among the optimal allocations.
- $\Pi_{WOO'}(A,R,U,K) \cup \Pi_{EFO2q}(A,R,U)$ (for DLV only), with a higher priority on the optimization statement in $\Pi_{WOO'}$, returns the allocations with the least number of envious agents among the optimal allocations.
- $\Pi_{WOO'}(A,R,U,K) \cup \Pi_{EFO1q}(A,R,U)$ (for DLV only), with a higher priority on the optimization statement in Π_{EFO1q}, returns the allocations with best social welfare among the allocations which minimize the envy degree.
- $\Pi_{WOO'}(A,R,U,K) \cup \Pi_{EFO2q}(A,R,U)$ (for DLV only), with a higher priority on the optimization statement in Π_{EFO2q}, returns the allocations with best social welfare among the allocations with minimum number of envious agents.
- The $\Pi_{WOOut'}$ can be applied instead $\Pi_{WOO'}$ in all cases above.

Note that for checking whether an optimal envy free allocation exists, a formalization of the WOO problem without weak constraints (or weights) is required, and such was not presented. However, the formalization of the POO problem with preferences in the bundle form (the translation Π_{POOBF}), can be extended with the notion of envy for returning Pareto-optimal envy free (the efficient envy free) solutions. This is done by adding the following rules:

```
hasUtilit(A):-acc(A,B).

envy(A1,A2):-
     agent(A1),not hasUtilit(A1),utility(A1,B,_),
     agent(A2),acc(A2,B),A1!=A2.

envy(A1,A2):-
     acc(A1,B),utility(A1,B,U),utility(A1,BB,UU),
     acc(A2,BB), UU>U,A1!=A2.
```

Then, for retuning the Pareto optimal allocations which are envy free (i.e. the efficient envy free allocations) the following constraint should be added.

```
:-envy(A1,A2).
```

For retuning the Pareto-optimal solutions with the minimal number of envious agents, the following weak constraint should be added.

```
:~envy(A1,A2).
```

The above translation is referred to as $\Pi_{POOEFBF}$.

4.9 Combinatorial Auctions – The Winner Determination Problem

Combinatorial auction is a centralized allocation protocol in which a special agent, the auctioneer, announces a set of resources it wants to sell and receives bids from other agents for subset of those resources. The goal of the auctioneer in such protocol is to select the set of mutually exclusive bids which maximize its revenue, a problem known as the winner determination problem in combinatorial auctions. This is the most extensively studied problem in MARA when it comes to its computational aspect and there have been many approaches for solving it including an ASP approach presented in [39] for single unit combinatorial auctions and several extensions. For the purpose of completeness as well as illustration of how this problem differs from the conventional welfare optimization problem in MARA, in this section an ASP formalization is presented for single unit combinatorial auctions with bids expressed in the OR and XOR languages.

Although combinatorial auctions can be reduced to the WOO problem solved in section 4.5 (preferences in the bundle form correspond to the XOR language, while preferences in the k-additive form correspond to the OR language), that approach does not seem intuitive. In essence, the set of valid allocations in the case of WOO is the set of all the possible assignments of resources to agents such that a resource is assigned to exactly one agent. This is not necessarily the case in combinatorial auctions where a set of resources can not be assigned to an agent if the agent has not made a bid for that set. The set of valid allocations, in this case, is the set of all combinations of mutually acceptable bids (two bids can not be accepted at the same time if they are for bundles which share a common resource). Accordingly, the method for generating models in most of the MARA problems solved prior to this will not be used in the formalization which follows. The method used here corresponds to the one used for the POO problem in the end of section 4.6.

Recall form chapter 2 that in the XOR and the OR language bids are of the form $\langle B_1,v_1\rangle$ XOR $\langle B_2, v_2\rangle$ XOR … XOR $\langle B_n, v_n\rangle$ and $\langle B_1, v_1\rangle$ OR $\langle B_2, v_2\rangle$ OR … OR $\langle B_n, v_n\rangle$ respectively, where each B_i is a bundle (i.e. subsets of the sets of resources) and v_i is a value the agent is prepared to pay for that bundle. Given a resource allocation setting $\langle A,R,U\rangle$ where U is the set of the bids of the agents expressed in one of these language, the bids are formalized with the following translation $\Pi_B(A,R,U)$ defined as follows:

- For every $a \in A$, with its bids of the form:
  $\langle B_1,v_1\rangle$ XOR $\langle B_2,v_2\rangle$XOR … XOR $\langle B_n, v_n\rangle$ or
  $\langle B_1, v_1\rangle$ OR $\langle B_2, v_2\rangle$ OR … OR $\langle B_n, v_n\rangle$, for every disjunct $1 \leq i \leq n$: a facts denoting the bundle `bid(`a`,`i`,`v_i`)` $\in \Pi_B(A,R,U)$, a fact for grounding the bundle identifier `bundle(`i`)` $\in \Pi_B(A,R,U)$ and for every resource $r \in B_i$ the

fact `inBundle(`i`,`r`).` $\in \Pi_B(A,R,U)$, where i is a bundle identifier as in the solution to the POO problem in section 4.6.

- A rule for grounding the bid values, given the maximum bid value bid_{max}
  `bidg(0`… bid_{max}`).`

Note that the translation is the same for both the XOR and the OR language, the only difference is that the former requires an additional constraint (rule number 3 below) for ensuring that at most one bid is accepted for each agent.

Consider an agent a with the following bid expressed in the OR language

$\langle\{r1\},2\rangle$ OR $\langle\{ r2\},2\rangle$ OR $\langle\{ r3\},1\rangle$ OR $\langle\{ r1, r2\},1\rangle$

The above translation will produce the following rules.

```
bid(a,1,2). bundle(1). inBundle(1,r1).
bid(a,2,2). bundle(2). inBundle(2,r2).
bid(a,3,1). bundle(3). inBundle(3,r3).
bid(a,12,1).bundle(12).
inBundle(12,r1).inBundle(12,r2).
```

Accordingly, given a resource allocation setting $\langle A,R,U\rangle$, the winner determination problem is formalized in ASP with the logic programs $\Pi_{Aor}(A,R,U)$ and $\Pi_{Axor}(A,R,U)$ for the bids in the OR and the XOR languages respectively, defined as follows:

- A translation $\Pi_B(A,R,U)$, of the bids defined above is included in both $\Pi_{Aoe}(A,R,U)$ and $\Pi_{Axor}(A,R,U)$
- A rule for generating answer set corresponding to all possible combinations of accepted bids, i.e. each bid being accepted or not:

  ```
  acc(X,Y) v notAcc(X,Y):- bid(X,Y,Z).                    (1)
  ```
  $\in \Pi_{Aor}(A,R,U), \Pi_{Axor}(A,R,U)$

  or the corresponding two rules without disjunction

  ```
  acc(X,Y):- bid(X,Y,Z),not notAcc(X,Y).
  notAcc(X,Y):- bid(X,Y,Z),not acc(X,Y).                  (1)
  ```
  $\in \Pi_{Aor}(A,R,U), \Pi_{Axor}(A,R,U)$

- A rule for ensuring that no two bids of different agents containing the same item are selected.

  ```
  :-acc(A,B), acc(AA,BB),A!=AA,
  inBundle(B,R), inBundle(BB,R).
  ```
  $\in \Pi_{Aor}(A,R,U), \Pi_{Axor}(A,R,U)$ (2)

- A rule for ensuring that no two bids of the same agent are selected, applying only for bids in the XOR language

  ```
  :-acc(A,B), acc(A,BB),B!=BB.
  ```
  $\in \Pi_{Axor}(A,R,U)$ (3)

- A weak constrain for minimizing the sum of not accepted bids, and thus maximizing the sum of accepted bids

  `:~ bid(A,U,B),not acc(A,B). [U:1]` $\in \Pi_{Aor}(A,R,U), \Pi_{Axor}(A,R,U)$

  or in case of Smodels the following optimization statement.

```
maximize[acc(A,B): bundle(B):
         agent(A):bidg(U):bid(A,B,U)=U].          (4)
```

Theorem 4.15: Given a resource allocation setting $\langle A,R,U\rangle$ with agents' bids expressed in the OR language, an optimal answer set $S \in \mathcal{AS}(\Pi_{or}(A,R,U))$ corresponds to a set of mutually acceptable bids $P \in Al_{n,m}$ which maximize the overall revenue. Similarly if the agents bids are given in the XOR language, an optimal answer set $S \in \mathcal{AS}(\Pi_{Axor}(A,R,U))$ corresponds to a set of mutually acceptable bids $P \in Al_{n,m}$ which maximize the overall revenue.

Proof: From rule 1 and the translation of the bids $\Pi_B(A,R,U)$, the answer sets $S \in \mathcal{AS}(\Pi_{AOR}(A,R,U))$ correspond to a set of accepted bids $P \in Al_{n,m}$, such that an accepted bid B_i of an agent a belongs to P if and only if `acc(`a`, `i`)` $\in S$. Because of the constraint 2, there are no two bids of different agents in P sharing the same resource. Thus P is a set of mutually acceptable bids. Because of the weak constraint 4 (or an optimization in LParse), the sum of the values of the bids in P must be maximized and thus P corresponds to a set of mutually acceptable bids which maximize the overall revenue. The proof for $\Pi_{Axor}(A,R,U)$ is analogous, with the additional constraint 3 ensuring that for each agent there is at most one bid in the corresponding set of mutually acceptable bids P. ■

4.10 Zeuthen Strategy, and the Monotonic Concession Protocol

Unlike the problems presented up till now, which are relevant to centralized allocation of resources, computing the next proposal under the Zeuthen strategy is a problem relevant to distributed protocols, more specifically to the Monotonic Concession protocol. Recall from chapter 2 that the Monotonic Concession protocol involves two agents which negotiate in rounds in which each agent can either make a new proposal (closer to the opponent's proposal) within the available deal space, i.e. concede, or stick to the proposal given in the previous round. In the beginning the agents are free to give any proposals. Taking into account the rationality assumption (the agents will agree only on deals which do not result in loss of utility), the proposals must not decrease the utilities of either of the agents, and must increase the utility of at least one. The obvious decisions that an agent needs to make while participating in the Monotinic Concession protocol is whether to concede in a given round, and if so, how much. Under the Zeuthen strategy the agent which is less willing to risk conflict is the one which needs to concede enough to change the balance of the risk. The agent whose proposal results in smaller product of utilities is the one less willing to risk conflict [25]. More specifically, consider two agents a and b, with the respective proposals P_a, P_b, and willingness to risk conflict Z_a and Z_b, $Z_a \leq Z_b$ is equivalent to $u_a(P_a) * u_b(P_a) \leq u_a(P_b) * u_b(P_b)$.

Taking this into account, what follows is a definition of a logic program for implementing an agent's behavior under the Zeuthen strategy in the Monotonic Concession protocol. The problem is as follows. If the agents are in the initial round of negotiation, the program will have as its answer sets all rational proposals (if such does not exist, the program will have no answer sets, i.e. the initial allocation is already Pareto-optimal). If it is an intermediate round in which a deal has been reached the program will have a single answer set containing an atom `dealRached`. If it is an intermediate round in which the agent whose behavior is modeled needs to stand, the program will have no answer sets, while if it needs to concede, it will have as its answer sets all rational proposals which result in change of the risk balance.

Since more than one allocation needs to be represented in each answer set (the initial allocation, and the most recent proposals) utilities will be represented with `utility`/3 atoms with the third argument denoting the allocation. In addition, in order to avoid recursion in the head of rules containing aggregate functions, a separate translation will be used for encoding the utilities of the new proposal. Such translation Π_{EFnp}, Π_{BFnp}, Π_{KAnp}, Π_{LSnp}, Π_{LCnp} and Π_{WGnp} (for the different utility representation) is derived from the translations Π_{EF}, Π_{BF}, Π_{KA}, Π_{LS}, Π_{LC} and Π_{WG} by adding the suffix '_np' (standing for new proposal) to every predicate name except of the *agent*/1 and *resource*/1 predicates . Thus for example the rules

```
utility(a,3):-not has(a, r1),has(a, r2).
goal(a):- has(a, r3).
coef(a,1,3):- has(a, r3).
utility(A,U):-agent(A),#sum{C,I: coef(A,C,I)}=U.
utility(A,U):-agent(A), #count{G:goal(A,G)}=U.
```

become

```
utility_np(a,3):-not has_np(a, r1),has_np(a, r2).
goal_np(a):- has_np(a, r3).
coef_np(a,1,3):- has_np(a, r3).
utility_np(A,U):-agent(A),#sum{C,I: coef_np(A,C,I)}=U.
utility_np(A,U):-agent(A), #count{G:goal_np(A,G)}=U.
```

Accordingly, given a resource allocation setting $\langle A,R,U\rangle$ with two agents a and b (i.e. $A=\{a,b\}$), an initial allocation P_i, and the agent's most recent proposals of P_a and P_b (in case it is not the initial round), the program $\Pi_{ZS}(A,R,U,P_a,P_i,P_b)$, for determining the next move of agent a, and in case of conceding the next proposal, consists of the following rules:

Data

- A translation $\Pi_U(A,R,U)$, which depending on the quantitative preference representation used is equal to $\Pi_{EF'}$, $\Pi_{BF'}$, $\Pi_{KA'}$, $\Pi_{LS'}$, $\Pi_{LC'}$ or $\Pi_{WG'}$.
- A translation $\Pi_{Unp}(A,R,U)$ for encoding the utilities of the new proposal, which depending on the quantitative preference representation used is equal to Π_{EFnp}, Π_{BFnp}, Π_{KAnp}, Π_{LSnp}, Π_{LCnp} or Π_{WGnp}.

- The translation of the initial allocation P_i consisting of the following facts:

  ```
  allocation(init).
  ```

  and for every agent-bundle pair $\langle a, B_a \rangle \in P_i$, and for every resource $r \in B_a$, the fact

  ```
  has(a, r ,init).
  ```

- The translation of the proposal P_a (in case it is not the initial round) consisting of the following facts:

  ```
  allocation(ap).
  ```

  for every agent-bundle pair $\langle a, B_a \rangle \in P_a$, and for every resource $r \in B_a$, the fact

  ```
  has(a, r ,ap).
  ```

- The translation of the proposal P_b (in case it is not the initial round) consisting of the following facts:

  ```
  allocation(bp).
  ```

  for every agent-bundle pair $\langle a, B_a \rangle \in P_b$, and for every resource $r \in B_a$, the fact

  ```
  has(a, r ,bp).
  ```

- A fact for denoting if it is initial round (in case it is not the initial round this fact is omitted):

  ```
  initialRound.
  ```

- A rule for denoting when a deal is reached

  ```
  dealReached:-
      not initialRound,utility(a,OP,bp),
      utility(a,CP,alp),OP >= CP.                    (1)
  ```

- A rule for denoting whether agent a should concede

  ```
  concede:-
      not initialRound,not dealReached,
      utility(a,N,alp),utility(b,NN,alp),
      Zi = N * NN,
      utility(a,N2,bp),utility(b,NN2,bp),
      Zo = N2 * NN2, Zi <= Zo.                       (2)
  ```

- Rules for denoting when to propose a new deal

  ```
  propose:-concede.
  propose:-initialRound.                             (3)
  ```

Guess

- Rule for guessing an allocation in the initial round, and in the case agent a concede

  ```
  hasNP(a,X) v hasNP(b,X):-res(X), propose.
  ```

This is equivalent to the following two rules not containing disjunction:

```
hasNP(a,X):-res(X),not hasNP(b,X), propose.
hasNP(b,X):-res(X),not hasNP(a,X), propose.      (4)
```

Check

- Rules for removing answer sets in which a deal is not reached and the agent does not concede, i.e. it stands to its previous proposal:

```
:-not initialRound, not concede, not dealReached.
                                                  (5)
```

- Rules for checking that the proposal is rational, i.e. it increase the utility of at least on agent without decreasing the utility of the others

```
:-utility_np(a,X),utility(a,Y,init), propose, Y > X.
:-utility_np(b,X),utility(b,Y,init), propose, Y > X.
:-utility_np(b,X),utility(b,X,init),
  utility_np(a,Y),utility(a,Y,init),propose.     (6)
```

- In the case agent *a* concedes, rules for ensuring that the new proposal changes the risk balance

```
riskBalancedChanged:-
     utility_np(a,N),utility_np(b,NN),
     Zi = N * NN,
     utility(a,N2,bp),utility(b,NN2,bp),
     Zo = N2 * NN2, Zi > Zo, concede.

:-not riskBalancedChanged, concede.              (7)
```

Theorem 4.16: Given a resource allocation setting $\langle A,R,U\rangle$ with two agents *a* and *b* (i.e. $A=\{a,b\}$), and an initial allocation P_i:

a) If the agents are in the initial round of negotiations, the answer sets $S\in\mathcal{AS}(\Pi_{ZS}(A,R,U,P_a,P_i,P_b)$ correspond to the rational proposals $P_{new}\in Al_{n,m}$, such that `has_np(`a`,`r`)` $\in S$ if and only if $\langle a, B_a\rangle\in P_{new}$ and $r\in B_a$

b) If it is not the initial round of negotiations, and the most recent agents' proposals are P_a and P_b there exist an answer set $S\in\mathcal{AS}(\Pi_{ZS}(A,R,U,P_a,P_i,P_b))$ if and only if a deal has been reached, or the agent *a* needs to concede under the Zeuthen strategy and there exist a rational proposal P_{new} which changes the risk balance. Also, for the latter case (deal is not reached), the answer sets $S\in\mathcal{AS}(\Pi_{ZS}(A,R,U,P_a,P_i,P_b))$ correspond to the rational proposals $P_{new}\in Al_{n,m}$ that change the risk balance such that `has_np(`a`,`r`)` $\in S$ if and only if $\langle a, B_a\rangle\in P_{new}$ and $r\in B_a$.

Proof: a) When the agents are in the initial state of negotiations an atom `InitialRound` $\in S\in\mathcal{AS}(\Pi_{ZS}(A,R,U,P_a,P_i,P_b))$ and thus `propose` $\in S$ from one of the rules in 3. The rules 1, 2, and the constraint 5 are irrelevant in this case since their body is never satisfied. Also the atoms `dealReached`, `concede` $\notin S$ since rules 1 and 2 are the only ones for deriving them respectively. The constraint 7 is also irrelevant since it

contain the atom `concede` in its body. From rule 4, and by the minimality of answer sets, each answer set S corresponds to a valid proposal (allocation) $P_{new} \in Al_{n,m}$. Given the translation of the initial allocation P_i , from the rules in $\Pi_U(A,R,U)$ for every agent i the atom `utility`$(i, vi, \texttt{init}) \in S$ where vi is the utility of the agent i for the initial allocation. Similarly, from the rules in $\Pi_{Unp}(A,R,U)$, for every agent i an atom `utility_np`$(i, vnp) \in S$ where vnp is the utility of agent i for the proposal P_{new}. Because of the constraint rules 5, P_{new} must increase the utility of at least one agent without decreasing the utility of the others, and thus the answer set S correspond to the rational proposals $P_{new} \in Al_{n,m}$.

b) When it is not the initial round, given the translations of P_i, P_a and P_b, from the rules in $\Pi_U(A,R,U)$ for every agent i the atoms `utility`$(i, vi, \texttt{init})$, `utility`$(i, vap, \texttt{ap})$, `utility`$(i, vbp, \texttt{bp}) \in S \in \mathcal{A}s(\Pi_{ZS}(A,R,U,P_a,P_i,P_b)$ where vi, vap and vbp are the utilities of the agent i for the initial allocation, the most recent proposal of agent a and the most recent proposal of agent b. If a deal is reached, i.e. the last proposal of agent b is at least as good for a as its own last proposal, a fact `dealReached`$\in S$ from rule 1. Nothing else belongs to S since the atom `propose` is not derived, and none of the bodies of the rules 2-7 is satisfied. On the other hand, when a deal has not been reached and the agent a should not concede under the Zeuthen strategy, i.e. $u_a(P_a) * u_b(P_a) > u_a(P_b) * u_b(P_b)$, the body of rules 1 and 2 will not be satisfied and the program will have no answer sets because of the constraint 5 as there are no rules for deriving either of the atoms `initialRound`, `coincide` or `dealReached`. When agent a should concede under the Zeuthen strategy, i.e. $u_a(P_a) * u_b(P_a) \leq u_a(P_b) * u_b(P_b)$, the body of rule 2 is satisfied and the atom `concede`$\in S$ and thus `propose`$\in S$ from one of the rules in 3. The valid proposals P_{new} are generated from rule 4, and from the rules in $\Pi_{Unp}(A,R,U)$, for every agent i an atom `utility_np`$(i, vnp) \in S$ where vnp is the utility of agent i for the proposal P_{new}. Because of constrain rules in 6 P_{new} must be rational proposal, and because of constraint 7 it must change the risk balance, i.e. $u_a(P_{new}) * u_b(P_{new}) > u_a(P_b) * u_b(P_b)$. ■

4.11 Constraints and Generalization

As stated previously, one of ASP's major advantages compared to alternative algorithms is the ease of defining and incorporating constraints in the solutions. To suoort this statement, what follow is some examples of common constraints with the corresponding ASP formalization, i.e. the rules which need to be added to the programs for incorporating the constraint.

Resource $r1$ must be allocated.

For the conventional resource allocation problems (except POO) :

```
allocated(R):-resource(R), agent(A), has(A,R).
:-not allocated(r1).
```

For auctions:

```
allocated(R):-acc(A,B), agent(A),
              bundle(B),inBundle(B,R).
:-not allocated(r1).
```

The resources *r*1 and *r*2 must be allocated together (i.e. to the same agent).

For the conventional resource allocation problems (except POO):

```
:-agent(A), has(A,r1),not has(A,r2).
:-agent(A), not has(A,r1),has(A,r2).
```

For auctions:

```
:-agent(A),acc(A,B),bundle(B),
  inBundle(B,r1),not inBundle(B,r2).
:-agent(A), acc(A,B),bundle(B),
  not inBundle(B,r1),inBundle(B,r2).
```

Resource *r*1 can only be allocated to the agents *a*1 and *a*2.

For the conventional resource allocation problems (except POO) :

```
:-agent(A),has(A,r1),A!=a1,A!=a2.
```

For auctions:

```
:-agent(A),bundle(B),acc(A,B),
  inBundle(B,r1),A!=a1,A!=a2.
```

All agents need to end up happy, i.e. they have a positive utility or they have at least one bid accepted.

For the conventional resource allocation problems (except POO):

```
happy(A):-agent(A),utility(A,U),U>0.
:- agent(A),not happy(A).
```

For auctions:

```
happy(A):-agent(A),acc(A,B).
:-agent(A),not happy(A).
```

A positive utility assigned to an agent should not be less than 10 or, in case of auctions, no bids of value less than 10 are accepted.

For the conventional resource allocation problems (except POO) :

```
:- agent(A),utility(A,U),U>0, U<10.
```

For auctions:

```
happy(A):- agent(A),acc(A,B).
:-agent(A),not happy(A).
```

The number of resources which are not allocated should be less than 5.

For the conventional resource allocation problems (except POO) :

```
allocated(R):-resource(R),agent(A),has(A,R).
free(R):-resource(R),not allocated(r1).
:-5 [free(R):resource(R)].
```

For auctions:

```
allocated(R):-acc(A,B), agent(A), inBundle(B,R).
free(R):-resource(R) , not allocated(r1).
:-5 [free(R):resource(R)].
```

Every agent should be allocated least 5 resources.

For the conventional resource allocation problems (except POO) :

```
:-agent(A),[has(A,R):resource(R)] 4.
```

For auctions:

```
:-agent(A), bundle(B),acc(A,B),[inBundle(B,R)] 4.
```

Beside constraints, generalization can also be made without significant changes in the program structure. For example, in order to move from single unit to multi unit auctions (in either the XOR or the OR language) the following changes should be made.

- The `resource`/1 atoms are extended with another argument denoting the number of resource units available.
- The `inBundle`/2 atoms are extended with another argument (aside from the bundle identifier and the resource) denoting the number of units of a specific resource in a bundle.
- Rules are added, for defining the number of units of each resource allocated for an accepted bid of a specific agent.

```
allocated(A,B,R,Amount):-
     agent(A),bundle(B),acc(A,B),
inBundle(B,R, Amount),A!=a1,A!=a2.
```

- A constraint is added for ensuring that the total amount of units allocated for a resource allocation, does not exceed the limit.

```
:-[allocated(A,B,R,Amount):agent(A):bundle(B):
inBundle(B,R,Amount)=Amount] Limit, resource(R,Limit).
```

The above are just few examples of how constraints which can be added to a program without any modification of the original program and how the original programs can be modified in order to capture a generalization. There is no doubt that there are many more examples in real world scenarios which are not addressed here but can be incorporated in a similar manner as presented above.

Beside the easy of adding constraints and making generalizations, it is important to note that most of the formalizations presented in this chapter are universal (or differ little) for different preference representations. Also, translations like $\Pi_{WOOut}(A,R,U)$, $\Pi_{WOO}(A,R,U)$, $\Pi_{WO}(A,R,U)$, and $\Pi_{POq}(A,R,U)$ apply in scenarios in which the preference representation might differ from agent to agent.

5. Results

This chapter covers experimental results for the ASP solutions presented in the previous chapter. The intention is not to compare the solutions to alternative algorithms (which are still to be published for most of the problems), nor to compare the performances of different answer set solvers, but rather to give an estimate of the size of instances the solution can be applied to, taking into account that a solution should be returned within a reasonable amount of time. Special attention is drawn on the welfare optimization and the Pareto-optimality problem (the optimization version), since these two are the most significant for the practical application of MARA. For the quantitative preferences, the bundle and the k-additive form are considered. Weighted goals are very similar to k-additive when formalized in ASP, and thus the results for the k-additive form are also relevant for weighted goals. The other two types of logic based preferences (single goal and cardinality of satisfied goals) are simplifications of weighted goals. Ordinal preferences are not considered since none of the presented solutions in which they are applicable is convenient for testing. The performance of the decision version of the Pareto-optimality problem is strongly depended on the input allocation which is checked for Pareto-optimality. If many allocation Pareto dominate the input one (which is almost certain for a random allocation in a large instance) an answer will be returned fast without a real measure of the performance and in the scope of this work there are no means for generating at least a 'decent' allocation. Same is the case with the welfare improvement problem, while, the solutions for the envy free problem make sense only when combined with other problems.

Tests are run on three answer set solvers: Smodels, DLV, and Clasp. Since the goal is not to compare answer set solvers, the performances of a program are presented only in the solver which performs the best.

The tests are run on Intel Centrino Dual core processor at 1.66GHz with 1GB RAM. There are five parameters that are considered in each test scenario:

- the **number of agents**
- the **number of resources**
- the **number of bundles** (coefficients/prioritized goals/bids) for which each an agent has specified a utility
- the **maximum utility** (coefficient/priority/bid) value which an agent can assign to a bundle
- the **maximum size** of a bundle for which an agent has a utility (coefficient/prioritized goal/bid)

The bundles for which an agent has a utility (coefficient/weighted goal/prioritized goal/bid) are generated randomly, firstly by choosing a random size of a bundle (in the range 1…maximum size), and then randomly choosing the resources in that bundle. Note that if a bundle is randomly generated for the second time, the generation step is repeated. (i.e. an agent can not have two utilities for the same bundle). Afterwards, the generated bundles are assigned a random utility value (coefficient/priority/bid). In the case

preferences are presented in the bundle form and XOR bids in combinatorial auctions, bigger bundles are more probable to be assigned a higher utility value compared to the smaller ones. Conversely, in the k-additive representation and OR bids in combinatorial auctions bigger bundles are more probable to be assigned a lower coefficient (priority) compared to the smaller ones. This distribution is devised in order the generated instances to better resemble real case scenarios.

In general the distribution of utilities has an effect on the results. For programs without optimization statements, the grater number of valid answer sets the less time is needed for the first one to be returned. For example, for the decision version of welfare optimization of utilitarian social welfare results are better for a distribution which assigns high utilities to small bundles compared to the converse one since in the former there are more allocations with high social welfare. The case is the opposite for the optimization version of the same problem. This is because in presence of optimization statements solvers perform worse when there are many solutions which are close to optimal. Accordingly, when maximizing utilitarian social welfare the above distribution leads to better results compared to a completely random one for preferences in the bundle form, while worse for preferences in the k-additive form.

5.1 Wellfare Optimization

For the decision version of the WO problem the results are closely depended on the value given as input for checking whether there exist an allocation with a greater social welfare. If the input value is small, then there will be many such allocation (and thus answer sets), and a solution will be returned fast. The following table sums up the results for the WO problem (the translation Π_{WO}) of utilitarian social welfare with preferences in the bundle form. The input value used is set to half of the maximum social welfare value, i.e. (*number of agents * maximum preference value*)/2. The results are listed for CLASP, which is the solver in which this problem is solved the fastest. The timings of DLV and Smodels are much worse. For example, the instances for which CLASP finds the solution within one second can take several minutes in either of the other two solvers. This is general for most of the problems.

Agents	resources	pref. per agent	Max pref. value	Max pref. size	Time for the first answer set in CLASP
10	30	30	15	10	<1s
10	30	30	15	30	<1s
15	30	30	15	30	2-3s
15	30	30	30	30	20-40s
20	30	30	15	30	>2m
30	50	30	15	30	>2m
30	50	100	15	30	>100s

Table 5.1.a. welfare optimization, utilitarian social welfare, bundle form.

In table above '<1s' is written if the answer is returned within one second, and '2m>' if the answer is not returned within 2 minutes. The timings cover the most common cases. For example if 20-40s is written in the table means that from around 8 out of 10 instances

of the problem a solution will be returned within this time frame, however, it might be the case that for some instance a result is not returned even in 2 minutes or, oppositely, returned within a second. In the tests which follow, such extreme cases are much more rear, and the timings are more precise.

Although the results do vary a lot from instance to instance, still important information can be drawn from this table. In essence, an increase in the number of agents or the maximum utility value of an agent, results in a drastic decrease of the overall performance. In fact, the range of possible social welfare values used for grounding the aggregate rule for calculating the social welfare enlarges with the increase of these parameters, resulting in many ground instantiations of the aggregate rule. This is much worse in the case of the Nash product, which has a greater range of social welfare values compared to the utilitarian social welfare. Similar is the case with k-additive preferences, which require additional aggregation. In addition, an increase of number of agent preferences or a decrease of the maximum preference size results in a better performance since the overall number of solution increases.

The results are significantly better when the explicit aggregation is not used (the translation Π_{WOBAut}, Π_{WOKAut}, Π_{WOLCut}, and Π_{WOWGut}). The following table sums up the results for the WO problem of utalitarian social welfare with preferences in the bundle form and the input value (the translation Π_{WOBAut}) same as before, half of the maximum social welfare value, i.e. (*number of agents* * *maximum preference value*)/2.

agents	resources	pref. per agent	max pref. value	max pref. size	Time for the first answer set in CLASP
10	30	30	15	10	<1s
10	30	30	15	30	<1s
20	30	30	15	30	1-2s
30	30	10	15	30	<1s
30	30	30	15	30	>2m
30	50	30	15	30	2s-30s
30	50	100	15	30	3s-10s
30	100	30	15	30	2s-20s
30	100	30	15	100	10s-20s
70	100	30	15	30	>2m

Table 5.1.b welfare optimization (no aggregates), utilitarian social welfare, bundle form.

The above table, if compared to the previous, clearly emphasizes the problems ASP encounters with aggregation, especially with the sum and the count aggregates. Thus, as long as there is no answer set solver which works with aggregates in more efficient way compared to the solvers used here, ASP solutions of problems for which such aggregation is necessary are only applicable to small instances. On the other hand, in presence of no aggregates, ASP performs relatively well.

The results are also satisfactory for the egalitarian and elitist social welfare. The following table sums up the results for the WO problem of egalitarian social (the translation Π_{WOeg}) welfare with preferences in the bundle form and the input value half of the maximum social welfare value, in this case *maximum preference value*/2.

agents	resources	pref. per agent	max pref. value	Max pref. size	Time for the first answer set in CLASP
10	30	30	5	10	<1s
10	30	30	15	10	<1s
10	50	30	15	20	<1s
20	50	30	15	20	<1s
30	70	40	15	20	1-3s
40	80	40	15	20	7-50s
50	50	50	15	10	>2m
50	100	50	15	20	40s-60s
70	150	50	15	20	>2m

Table 5.2. welfare optimization, egalitarian social welfare, bundle form.

For the elitist social welfare, the results are even better compared to the egalitarian social welfare with the input value equal to the maximum social welfare, i.e. the *maximum preference value*.

Next, the optimization version of the WO problem (WOO) is considered. The following table sums up the results for the WOO problem of egalitarian social (the translation Π_{WOO}) welfare with preferences in the bundle form. The same problem requires the sum aggregate in case of k-additive preferences and the results are significantly worse.

agents	resources	pref. per agent	max pref. value	max pref. size	Time for the first answer set in CLASP
10	30	30	15	10	<1s
20	30	30	15	10	10-40s
20	30	30	15	30	<1s
20	50	30	15	30	2-3s
20	70	30	15	30	3-4s
30	70	30	15	30	5-7s
50	70	40	15	30	40-60s
50	100	40	15	30	>2m

Table 5.3. welfare optimization (optimization version), egalitarian social welfare, bundle form.

For the WOO problem of utilitarian social welfare, aggregation is not necessary for preferences in the bundle form or in the k-additive form. The following tables sums up the results for the solution of the WOO problem, firstly for the bundle form (the translation Π_{WOOut}) and then for the k-additive form (the translation $\Pi_{WOOKAut}$).

agents	resources	pref. per agent	max pref. value	max pref. size	Time for the first answer set in CLASP
10	30	10	15	10	<1s
10	30	30	15	10	>2m
10	30	30	15	30	1-3s
20	30	30	15	30	20s-100s
30	30	10	15	30	3-5s

30	50	10	15	30	4-10s
30	100	10	15	30	20-90s
30	100	10	15	100	1-3s
70	100	10	15	100	20-40s

Table 5.4.a welfare optimization (optimization version), utilitarian social welfare, bundle form.

agents	resources	coef. per agent	max coef. value	max coef. size	Time for the first answer set in CLASP
10	30	10	15	10	1-2s
10	30	30	15	10	>2m
10	30	30	15	30	1-2s
20	30	30	15	30	15-90s
30	30	10	15	30	3-5s
30	50	10	15	30	4-6s
30	100	10	15	30	20-90s
30	100	10	15	100	1-3s
70	100	10	15	100	10-20s

Table 5.4.b welfare optimization (optimization version), utilitarian social welfare, k-additive form.

From the above tables it can be concluded that the overall number of preferences/coefficients (*agents* * *preferences per agent*) has a great influence on the performance, together with the amount of mutually acceptable bundles for which agent's have a defined preference/coefficient i.e. the ratio between the number of resources and the size of the preferences/coefficients. The latter is evident by the fact that a decrease in the maximal preferences/coefficients size (which results in an increase in the ratio) drastically decreases the performance, as shown by the first two columns in the tables above. This is completely opposite in the WO problem presented before, thus when the ratio is big the solution of the WO problem can be used for finding a reasonable good allocation, while, if the ratio small the solution of the WOO problem can be used for finding the optimal solution.

Finally, the following table sums up the results for returning the leximin optimal solutions in a resource allocation setting with preferences in the bundle form (the k-additive form will require aggregation and thus would be less efficient). Recall that the formalization of this problem Π_{WOOlo} used priorities on weights and thus it is only applicable in DLV.

Agents	resources	pref. per agent	max pref. value	max pref. size	Time for the first answer set in DLV
5	10	10	15	10	<1s
5	10	30	15	10	<1s
10	10	30	15	10	7-11s
10	10	30	15	5	7-11s
10	20	30	15	10	>2m
20	10	30	15	10	>2m

Table 5.5 welfare optimization (optimization version), leximin, bundle form.

5.2 Pareto Optimality

Regarding the PO problem, only its optimization version is considered since the performance of the solution for the decision version are closely dependent on the allocation which is given as input to the problem. Same as the leximin optimality problem, the solution of the POO problem, with preferences in the bundle form (the translation Π_{POOBF}), is only applicable in DLV. The following table sums up the results.

Agents	resources	pref. per agent	max pref. value	max pref. size	Time for the first answer set in DLV
10	30	10	15	10	1-2s
10	30	10	15	30	5s-20s
10	70	10	15	10	1-5s
10	70	10	15	30	10s-20s
10	30	30	15	10	30s-120s
10	30	30	15	30	>2m
30	30	10	15	10	15s-100s
50	30	10	15	10	>60s

Table 5.6 Pareto-optimality(optimization version), bundle form.

As shown by the above table, the performance of the POO problem decrease with the increase of the size of preferences due to the computation necessary for ensuring that the considered allocations are valid. This is completely opposite to the results for the WOO problem discussed in the preceding section. Another important difference is that the performance of the POO problem is more sensitive to an increase of the number of preferences per agent compared to an increase of the number of agents. In the WOO problem this is not the case as the overall number of preferences is what matters, regardless of whether there are many agents with not many preferences or vice versa.

5.3 Envy Free

Since it does not make sense to run the solution for the EF problem alone (often, the returned allocation will assign bundles for which the agents have a 0 utility value), this problem it is combined with other problems. Thus, the table which follows sums up the results for finding the envy free allocation which maximize utilitarian social welfare in a resource allocation setting with preference in the bundle form (the translation $\Pi_{WOOut}(A,R,U) \cup \Pi_{EFq}(A,R,U)$).

agents	resources	pref. per agent	max pref. value	max pref. size	Time for the first answer set in CLASP
10	30	10	15	10	1-2s
10	30	30	15	10	>2m
10	30	30	15	30	2-5s
20	30	30	15	30	>100s
30	30	10	15	30	15-25s
30	50	10	15	30	30-60s
30	100	10	15	30	>2

30	100	10	15	100	15-25s
50	100	10	15	100	>100s

Table 5.7. envy free welfare optimization (optimization version), utilitarian social welfare, bundle form.

The results above are slightly worse than the ones for the WOO problem alone in table 5.4.a. This means that the envy free check does not add a great computational cost. However, when preferences are in the k-additve form the performance decrease drastically compared to the ones in table 5.4.b due to the introduction of aggregates for the envy free check. In this case, problems involving more than 10 agents and 10 resources can not be solved within two minutes.

The results for the solution of the POO and the EF problems combined with preference in the bundle form, i.e. finding an efficient envy free solution in a resource allocation setting (the translation $\Pi_{POOEFut}(A,R,U)$), are more or less identical to the results for the POO problem alone presented in table 5.6. This is because the computational cost for generating envy free allocations is compensated by the reduction in the number of allocations checked for Pareto-optimality. Even, in some cases, the solution for the POO and EF combined performs better than the solution to the POO alone.

Next, the results for the solutions of the WOO problem and the optimization versions of the EF problem combined are presented. The following table sums up the results for returning the allocations with the least number of envious agents among the optimal allocations in respect to utilitarian social welfare, with preferences in the bundle form (the translation $\Pi_{WOOut}(A,R,U,K) \cup \Pi_{EFO2q}(A,R,U)$).

Agents	**resources**	**pref. per agent**	**max pref. value**	**max pref. size**	**Time for the first answer set in DLV**
5	10	10	15	10	<1s
5	10	30	15	10	1-2s
10	10	10	15	10	5-20s
10	10	30	15	10	2m>
10	20	10	15	10	2m>
20	10	10	15	10	2m>

Table 5.8 welfare optimization (optimization version), utilitarian social welfare, with least number of envious agents, bundle form.

These results are also relevant for the reverse problem, i.e. returning the allocation optimal allocation among the allocations with the smallest number of envious agents, as well as, the other optimization version of the EF problem, minimizing the degree of envy (the translation $\Pi_{WOOut}(A,R,U,K) \cup \Pi_{EFO1q}(A,R,U)$).

5.4 Winner determination in combinatorial auctions

The following tables sum up the results for the OR and the XOR problems (the translations $\Pi_{Axor}(A,R,U)$ and $\Pi_{Aor}(A,R,U)$) respectively.

Agents	resources	Bids per agent	max bid value	max bid size	Time for the first answer set in CLASP
10	30	10	15	10	<1s
10	30	30	15	10	>60s-2m
10	30	30	15	30	4-5s
20	30	30	15	30	35s-90s
30	30	10	15	30	4-6s
30	50	10	15	30	4-7s
30	100	10	15	30	10-30s
30	100	10	15	100	22s-25s
70	100	10	15	100	>2m

Table 5.9a winner determination problem, XOR bids

Agents	resources	Bids per agent	max bid value	max bid size	Time for the first answer set in CLASP
10	30	10	15	10	<1s
10	30	30	15	10	>60s
10	30	30	15	30	5-6s
20	30	30	15	30	35s-90s
30	30	10	15	30	4-6s
30	50	10	15	30	4-7s
30	100	10	15	30	10-30s
30	100	10	15	100	22s-25s
70	100	10	15	100	>2m

Table 5.9b winner determination problem, OR bids

The above results can be directly compared with the ones in table 5.4.a and 5.4.b for the WOO problem of utilitarian social welfare with preferences in the bundle and the k-additive form, since this is essentially the same problem as the winner determination problem. This provides a good reference for comparison of the performance of the to alternative problem formalizations, the one which generates allocations on the base of allocating resources to agents (used in WOO), and the one which generates allocations on the base of accepting or not accepting a preference (bid) of a specific agent (used in the winner determination and the POO problem). Apparently, the two perform very similar. The latter is a bid faster for smaller instances, however when the number of resources and number of agents is large, it performs worse due to the time needed for parsing (as shown by the last two columns of the tables).

5.5 Zeuthen Strategy

This problem is inconvenient for testing since it is dependent on the initial allocation as well as the two last proposals, all of which are given as input. In general, when the agent should not concede, the program is stratified and solutions are returned fast. The actual complexity arises when the agent which behavior is modeled needs to concede. Since running a full negotiation process (the two programs run in each negotiation step, and the output of the programs is used as input in the next negotiation step) is impractical for

large data, the testing is done in a different manner. All three input allocation are generated random (although it is often the case they do not represent a realistic negotiation step), and the agent is forced to concede no matter if it should or not. This way, the program will try to find a rational allocation (better than the initial one) which changes the risk balance, and thus going through all the computation necessary for generating a valid proposal. Accordingly, the results would be also relevant for a realistic negotiation step, and are presented in the table below, for preference in the bundle form. For k-additive preferences aggregation is necessary and results will be significantly worse. For this problem Smodels have the best performance, though CLASP has similar timings.

resources	pref. per agent	max pref. value	max pref. size	Time for the first answer set in Smodels
50	500	15	50	2-3s
200	500	15	200	2-3s
200	1000	15	200	10s-20s
100	1500	15	100	10s-20s
100	2000	15	100	20-30s
200	1500	15	200	>2m

Table 5.10 Determining the next proposal under the Zeuthen strategy in the monotonic concession protocol, with preferences in the bundle form.

The results are expectably much better than for any of the problems discussed previously. This is because the solution of this problem does not contain optimization statements.

5.6 Additional constraints

Most of the additional constraints do not have a major effect on the performance, thus the results presented in the preceding sections are also relevant in presence of additional constraints. Although constraints in most of the cases decrease the number of solution, they also decrease the search space. Of course there can be exceptions when the constraint decreases significantly the number of solutions and not so much the search space (or vice versa).

6. Conclusions and Further work

The goal which initiated this work was to find flexible, declarative and efficient solutions to common MARA problems. This was motivated by many real world applications in which this attributes are either necessary or strongly favored. It was argued that a fully declarative logic programming paradigm, Answer Set Programming, can be used for providing solutions with such characteristics given its declarative, intuitive and highly expressive semantics. This was also aided by the fact that ASP had been already successfully applied for solving the winner determination problem in combinatorial auctions.

Indeed, ASP proved to be adequate for solving many complex problems in MARA. In this respect, solutions were presented for the following problems:

- the **welfare improvement problem** for: the Nash product, utilitarian, elitist, and egalitarian.
- the **welfare optimization problem**, both the decision and the optimization version, for: the Nash product, utilitarian, elitist, egalitarian and the leximin social welfare(only applies to the optimization version).
- the **Pareto-optimality** problem, both the decision and the optimization version.
- the **envy free** problem, both the decision version and the optimization versions (i.e. minimizing the envy degree and minimizing the number of envious agents).
- the **winner determination** problem in combinatorial auctions for bids in the XOR and OR languages,
- determining the next proposal under the **Zeuthen strategy** in the Monotonic Concession protocol.

It was also presented how some of these problems can be combined, particularly the different variations of the envy free problem with the Pareto-optimality and the welfare optimization problem (for example, finding an efficient envy free solutions), as well as, how some common constraints can be incorporated. The solutions capture the most common preference representations including, for quantitative preferences: the bundle form, the k-additive form, and the weighted goals; and for ordinal preferences: the best-out, the descrimin and the leximin orderings of prioritized goals. Exception is the optimization version of the Pareto-optimality problem which applies only to preferences in the bundle form.

The ASP approach is general and flexible, thus the solutions can be easily adapted for solving real world problems in dynamic domains. This is supported by the ease of which additional constraints can be added and by the fact that the solutions are universal (with small exceptions) for different preference representations used and for different metrics measuring the social welfare. Regarding efficiency, the only reference for comparison is combinatorial auctions, since alternative algorithms are very rare for the other problems. In this respect, the performance of the ASP solutions, as stated in [39], can not be compared to the specialized algorithms for winner determination. However, the

specialized algorithms do not offer the flexibility of the ASP approach, particularly, in terms of different preference representations, constraints and generalizations.

The presented work can also aid the development of efficient answer set solvers, since the solutions can be used as benchmarks for testing. It provides an example of how ASP can be used for solving problems of different complexity classes in a declarative and flexible manner and it is likely to encourage researches to apply ASP for problems in other domains which share similar requirements with MARA.

The focus in the dissertation was on the 'hottest' problems in centralized MARA while there are also other complex MARA problems, especially in negotiations, which were not addressed here. If the need arises, such problems can be solved with ASP in the future, either as a completely separate work or as a continuation of this dissertation. However, of greater importance for the completeness of the ASP application in MARA is to apply the presented solutions in a practical application of MAS systems. In essence, in this dissertation results were presented only for completely artificial data, and would be interesting to see how the solutions scale for realistic problem instantiations, as well as how the solutions of several problems can be used for developing a single agent capable of solving different MARA problems depending on the needs at a specific moment. Accordingly, it is strongly suggested that this work is complemented with a case study in which most of the problems solved here are relevant.

References

[1] Y. Chevaleyre, P. E. Dunne, Ulle Endriss, Jerome Lang, Michel Lemaitre Nicolas Maudet , Julian Padget, Steve Phelps, Juan A. Rodriguez-Aguilar and Paulo Sousa. Issues in Multiagent Resource Allocation. *Informatica*, 30:3–31, 2006.

[2] Y. Chevaleyre, U. Endriss, S. Estivie, and N. Maudet. Multiagent resource allocation with k-additive utility functions. In Proc. *DIMACS-LAMSADE Workshop on Computer Science and Decision Theory*, Annales du LAMSADE 3, 83–100, 2004.

[3] P. E. Dunne, M. Wooldridge, and M. Laurence.The complexity of contract negotiation. *Artificial Intelligence*, 164(1–2):23–46, 2005.

[4] J. Lang. Logical preference representation and combinatorial vote. *Annals of Mathematics and Artificial Intelligence*, 42(1– 3):37–71, 2004.

[5] R. I. Brafman and C. Domshlak. Introducing variable importance tradeoffs into CPnets. In Proc. *18th Conference in Uncertainty in Artificial Intelligence (UAI-2002)*, 69–76. Morgan Kaufmann, 2002.

[6] C. Boutilier, F. Bacchus, and R. I. Brafman. UCP-networks: A directed graphical representation of conditional utilities. In Proc. *17th Annual Conference on Uncertainty in Artificial Intelligence (UAI-2001)*, 56–64. Morgan Kaufmann, 2001.

[7] T. W. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135(1–2):1–54, 2002.

[8] N. Nisan. Bidding languages for combinatorial auctions. In P. Cramton et al., editors, *Combinatorial Auctions*. MIT Press, 2006.

[9] Y. Fujishima, K. Leyton-Brown, and Y. Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In Proc. *16th International Joint Conference on Artificial Intelligence (IJCAI-1999)*, 548–553. Morgan Kaufmann, 1999.

[10] K. J. Arrow, A. K. Sen, and K. Suzumura, editors. *Handbook of Social Choice and Welfare*. North-Holland, 2002.

[11] H. Moulin. *Axioms of Cooperative Decision Making*. Cambridge University Press, 1988.

[12] H. Moulin. *Fair Division and Collective Welfare*. MIT Press, 2003.

[13] P. Cramton, Y. Shoham, and R. Steinberg, editors. *Combinatorial Auctions*. MIT Press, 2006.

[14] J. Kalagnanam and D. C. Parkes. Auctions, bidding and exchange design. In D. Simchi- Levi et al., editors, *Handbook of Quantitative Supply Chain Analysis: Modeling in the E-Business Era*. Kluwer, 2004.

[15] V. Krishna. *Auction Theory*. Academic Press, 2002.

[16] W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16(1):8–37, 1961.

[17] P. R. Wurman, M. P. Wellman, and W. E. Walsh. A parametrization of the auction design space. *Games and Economic Behaviour*, 35(1–2):304–338, 2001.

[18] T. W. Sandholm. Contract types for satisficing task allocation: I Theoretical results. In Proc. *AAAI Spring Symposium: Satisficing Models*, 1998.

[19] U. Endriss, N. Maudet, F. Sadri, and F. Toni. On optimal outcomes of negotiations over resources. In Proc. *2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS- 2003)*, 177–184. ACM Press, 2003.

[20] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, 1980.

[21] T. W. Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In Proc. *11th National Conference on Artificial Intelligence (AAAI-1993)*, 256–262. AAAI Press, 1993.

[22] A. Giovannucci, J. A. Rodr´ıguez-Aguilar, A. Reyes, F. X. Noria, and J. Cerquides. Towards automated procurement via agentaware negotiation support. In Proc. *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2004)*, 244–253. ACM Press, 2004.

[23] P. Sousa, C. Ramos, and J. Neves. The Fabricare scheduling prototype suite: Agent interaction and knowledge base. *Journal of Intelligent Manufacturing*, 14(5):441-455, 2003.

[24] S. Aknine, S. Pinson, and M. F. Shakun. An extended multi-agent negotiation protocol. J*ournal of Autonomous Agents and Multi- Agent Systems*, 8(1):5–45, 2004.

[25] Ulle Endriss. Monotonic concession protocols for multilateral negotiation. In Proc. *The fifth international joint conference on Autonomous agents and multiagent systems*, 392 - 399. 2006.

[26] F. Zeuthen. *Problems of Monopoly and EconomicWarfare*. Routledge, 1930.

[27] P. E. Dunne. Multiagent resource allocation in the presence of externalities. In *Proc. 4th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS-2005)*, 408–417. Springer- Verlag, 2005.

[28] S. Bouveret and J. Lang. Efficiency and envy-freeness in fair division of indivisible goods: Logical representation and complexity. In Proc. *19th International Joint Conference on Artificial Intelligence (IJCAI- 2005)*, 935–940. Morgan Kaufmann, 2005.

[29] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In Proc. *Fifth International Conference on Logic Programming*, 1070-1080. MIT Press, Cambridge, 1988.

[30] Henry Kautz and Bart Selman. Planning as Satisfiability. In Proc. *Tenth European Conference on Artificial Intelligence (ECAI'92)*, 369-363. John Wiley & Sons, Inc, 1992.

[31] I. Niemelä, P. Simons. Smodels - An Implementation of the Stable Model and Well-Founded Semantics for Normal Logic Programs. In Proc. *4th International Conference on Logic Programming and Nonmonotonic Reasoning*, 421–430. Springer- Verlag, 1997.

[32] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. A deductive system for nonmonotonicreasoning. In *Proc. of KR 98*, 406–417, 1998.

[33] V. Lifschitz. Action Languages, Answer sets and planning. In *The Logic Programming Paradigm, a 25-years perspective*, 357-373. Springer Verlag 1999.

[34] I. Niemelä. Logic programs with stable models semantics as constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence,* 25:241-273, 1990.

[35] V. Matek and M. Truszczynski. Stable models and an alternative logic paradigm. In *The Logic Programming Paradigm, a 25-years perspective*, 375-398. Springer Verlag 1999.

[36] C. Baral. *Knowledge Representation, Reasoning and Declarative problem solving*. Cambridge University Press, 2003.

[37] A. V. Gelder, K. A. Ross and J. S. Schilpf. The Well-Founded Semantics for General Logic Programs. In *Jurnal of ACM (JACM)*, 619-649. ACM 1991.

[38] P. Simons. Extending the stable model semantics with more expressive rules. In *Proc. of InternationalConference on Logic Programming and Nonmonotonic Reasoning, LPNMR'99*,1999.

[39] C. Baral and C. Uyan. Declarative Specification and Solution of Combinatorial Auctions Using Logic Programming. In *Proc. of International Conference on Logic Programming and Nonmonotonic Reasoning LPNMR-01*, 186–199, 2001.

[40] J. W. Lloyd. Practical advantages of declarative programming. In *Joint Conference on Declarative Programming, GULP-PRODE'94*, 94.

[41]T. Sandholm. An algorithm for optimal winner determination in combinatorial auctions. In Proc. of *16th International Joint Conference in Artificial Intelligence IJCAI 99*, 542–547, 1999.

[42] Thomas Eiter, Axel Polleres. Towards automated integration of guess and check programs in answer set programming: a meta-interpreter and applications. In *Theory and Practice of Logic Programming*, 23 - 60, Cambridge University Press, 2006.

[43] Martin Gebser, Benjamin Kaufmann, André Neumann and Torsten Schaub. Clasp: A conflict-driven answer set solver (2007). *In Proc.* of *InternationalConference on Logic Programming and Nonmonotonic Reasoning*, *LPNMR'07*, 2007.

[44] T. Eiter, G. Gottlob, and H. Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems(TODS)*, 22(3):364-418, 1997.

[45] Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Declarative Problem-Solving Using the DLV System. In Jack Minker, editor, *Logic-Based Artificial Intelligence*, 79–103. Kluwer Academic Publishers, 2000.