INSTITUT FÜR INFORMATIK
Lehr- und Forschungseinheit für
Programmier- und Modellierungssprachen
Oettingenstraße 67, D–80538 München

Set Description Languages and Reasoning about Numerical Features of Sets

Hans Jürgen Ohlbach

http://www.pms.informatik.uni-muenchen.de/publikationen
Forschungsbericht/Research Report PMS-FB-2001-1, April 2001

Set Description Languages and Reasoning about Numerical Features of Sets

Hans Jürgen Ohlbach
Institut für Informatik, Universität München
email: ohlbach@informatik.uni-muenchen.de

March 27, 2001

Abstract

Set description languages, for example description logics, can be used to specify sets and set-theoretic relationships between them. Mathematical programming, on the other hand, can be used to find optimal solutions for arithmetical equation and in-equation systems. In this paper a combination methodology is presented, which allows one to use numerical algorithms for reasoning about numerical features of sets specified with a set description language. The method is applied to description logics as an examples for a set description language. In this context new properties of logical languages become interesting which have not been considered so far.

Contents

1 Introduction

If somebody says that his grandma had two sons and one daughter, you know immediately that she had three children. What is so easy for humans, however, is not easy at all for computers because it requires a combination of logical reasoning (sons are children, daughters are children etc.) and arithmetical reasoning ($2 + 1 = 3$). Both have been well investigated separately, but how to combine strong logical systems with strong arithmetical systems is still not well understood.

There is, however, not *the* combination of logic and arithmetic. Moreover, there are very different ways to combine them, and it depends on the application which combination is appropriate. In this paper we consider a combination where a logic is used to specify sets and set-theoretic relationships between them, and the arithmetic is used to specify arithmetical relations between numerical features of the sets. The above example illustrates what this means. The sets under consideration are the sets of sons, daughters and children. The set theoretic relationships are in particular the subset relationships between sons and children and daughters and children. The numerical features of these sets are their cardinalities. Finally, the arithmetic deals with the numbers involved.

This problem has already been investigated by Ohlbach & Koehler in [7] for the case that the logic is just plain Propositional Logic. In this paper the approach from [7] is generalized to almost any logic which is able to specify sets. Description logics [4] are prominent examples for such 'set description languages'.

It turns out that the proposed combination methodology puts some requirements on the set description language, which may trigger new theoretical investigations of logics. For example, how to characterize multi-modal logics with the property that a consistent formula can be made true in exactly n worlds? Alternatively, is there a decision procedure for checking whether a given formula can be made true in exactly n worlds?

1.1 Related Work

As already mentioned, this paper carries the ideas forward which have been presented in [7] and also in [8].

A combination of description logic and arithmetic, or more general, concrete domains, has been presented by Baader and Hanschke in [2] and [3]. In these papers 'functional roles', i.e. functions mapping objects of the set domain to other concrete domains, for example numbers, have been used to increase the expressivity of the description logic. Functional roles characterize individuals, whereas in the approach presented in this paper, 'bridging functions' characterizing a *whole set* of objects are the focus of the attention. The set description language is more or less used as a black box for the combination methodology of this paper. It could very well be a description logic with functional roles in the style of [2]. There can be correlations between functional roles and bridging functions, but this has not yet been investigated (cf. Sec. 7).

There is a closer relationship between this paper and the paper 'Cardinality Restrictions on Concepts' [1]. In [1] cardinality restrictions of the type $|c| \leq n$ or $|c| \geq n$ can become part of a TBox in the particular description logic $\mathcal{ALCQ}$. The consistency algorithm for this language is a modified tableaux calculus, which treats cardinality restrictions similar to qualified number restrictions.

The difference to the method presented in this paper is that in this paper the TBox is kept separate to the specification of the numerical constraints of the sets. Therefore we can us consistency algorithms of description logic which do not need to treat cardinality restrictions of concepts. Moreover, we can deal with a lot more general relations between numerical features of sets, not only cardinalities. Again, the description logic presented in [1] could be the SDL-part for our combination methodology.

1.2 A Road Map

The introduction gives a first overview on the languages and techniques involved. A motivating example from Operations Research is presented in Section 1.7. The logical and arithmetical languages involved are characterized in Section 2. Since both are parameters to the combination methodology, no concrete languages are defined, but the constraints on the languages are explained which are necessary to make the combination methodology work. The combination methodology, 'atomic decomposition' is presented in Section 3. This method allows one to get rid of the logical part of a combined logical-arithmetical problem and to translated it into a pure arithmetical problem.

Unfortunately, it turns out that the arithmetical solutions are not always correct. This can happen if the logic itself is expressive enough to state constraints on the cardinality of sets. To account for this effect, one has to filter out wrong solutions. How to do this is explained in Section 4.

There are quite a number of steps in the method where optimizations are possible. Some of them are independent of the concrete languages, some depend on the languages involved. This is discussed in Section 5.

In Section 6 the whole method is applied to description logics as set description languages. Further research problems emerging from the investigations in this paper are summarized in Section 7.

1.3 What are Set Description Languages?

As the name indicates, Set Description Languages (SDL) are logical languages for describing sets, but what kind of sets? General sets of elements, sets of tuples (binary relations), sets of triples (ternary relations) or what? In fact, so far special set description languages for general sets have been developed (Description Logics (DL) [4]) and set description languages for binary relations (representable relation algebras [5]). First Order Predicate Logic is a set description language for any kind of sets, but satisfiability for this language is undecidable. Therefore it is not well suited for getting a reasonably efficient system.

The simplest set description language for general sets is Propositional Logic (PL) with its set-theoretic interpretation. Propositional variables are interpreted as sets, conjunction ($\wedge$) as set intersection ($\cap$), disjunction ($\vee$) as set union ($\cup$), negation ($\neg$) as set complement ($'$), implication ($\Rightarrow$) as subset relationship ($\subseteq$) and equivalence ($\Leftrightarrow$ or just $=$) as set identity.

Description logics augment Propositional Logic with an existential quantifier ($\exists$) like the modal $\Diamond$-operator and a universal quantifier ($\forall$) like the modal $\Box$-operator, together with several other operators and constructors. A typical DL expression could be

$$\mathit{parent} = \mathit{person} \wedge \exists\, \mathit{has\text{-}child.person}$$

defining the 'concept' *parent* as a person having a child which is also a person. *parent* and *person* denote sets and *has-child* denotes a binary relation.

Propositional Logic is also a language for specifying binary relations. For example

$$\begin{aligned} \mathit{has\text{-}son} &\Rightarrow \mathit{has\text{-}child} && (1)\\ \mathit{has\text{-}daughter} &\Rightarrow \mathit{has\text{-}child} && (2)\\ \mathit{has\text{-}child} &\Rightarrow \mathit{has\text{-}son} \vee \mathit{has\text{-}daughter} && (3)\\ \mathit{has\text{-}son} \wedge \mathit{has\text{-}daughter} &\Rightarrow \bot && (4) \end{aligned}$$

(1) expresses that the *has-son*-relation is a sub-relation of the *has-child*-relation. (3) expresses that children are either sons or daughters, and therefore the

has-child-relation can be decomposed into the disjoint sub-relations *has-son* and *has-daughter*.

In this context falsity ($\bot$) denotes the empty relation and truth ($\top$) denotes the universal relation.

The language of relation algebras introduces two more operators besides the Boolean operators. They make only sense for binary relations, and not for general sets. The composition operator (;) denotes composition of relations and the inverse operator ($^{-1}$) denotes the inverse relation. In this language one can for example express with

$$r; r \Rightarrow r \tag{5}$$

that the relation r is transitive, or one can define a new relation

$$\textit{has-grand-child} = \textit{has-child}; \textit{has-child} \tag{6}$$

in terms of other relations.

Relational terms are in particular used together with the quantifiers of more expressive Description Logics. For example 'cars all whose parts are manufactured in European countries' could be defined as

$$\textit{car} \wedge \forall\, (\textit{has-part}; \textit{manufactored-by}; \textit{located-in}).\textit{European-country}$$

using the composition of the three relations *has-part* (mapping to parts of the cars) *manufactured-by* (mapping parts to companies), and *located-in* (mapping companies to countries).

The language of relational terms is another example of a set description language, and the combination methodology should work for it as well. The details, however, are not presented in this paper.

1.4 Set Description Languages and Arithmetics

Set description languages can specify sets and set theoretic relationships between sets, but they usually cannot specify numerical features of sets. For example, in a SDL you can specify

$$\textit{silk-ties} \Rightarrow \textit{ties} \tag{7}$$

(all silk ties are ties), but you cannot express 'there are at least 100 silk ties'. In order to do this, you need numbers, arithmetic relations and a 'bridging function' mapping sets to numbers, in this case the cardinality function $|.|$.

$$|\textit{silk-ties}| \geq 100 \tag{8}$$

would be a natural way to express this. In a suitable combination of logical reasoning and arithmetical reasoning one could then prove from (7) and (8) for example $|\textit{ties}| \geq 50$. One could also prove from (7) and a statement $\textit{max-cost}(\textit{ties}) \leq 5$ the formula $\textit{max-cost}(\textit{silk-ties}) \leq 7$. (*max-cost* is another 'bridging function' which maps sets to numbers).

In [7] it was shown how a combination of logical and arithmetical reasoning can solve such mixed problems if the set description language is just Propositional Logic. In this paper we extend the method for very general Set Description Languages.

1.5 Atomic Decomposition

Atomic Decomposition was introduced in [7] as a method for eliminating the logical part of a mixed SDL-arithmetical problem where SDL is just Propositional Logic. With this technique mixed problems can be reduced to pure arithmetical problems. The key ideas of this technique are also applicable when the SDL is more expressive than PL. Therefore we give a brief overview on the atomic decomposition technique.

Atomic decomposition exploits the possibility to decompose finite sets of sets into mutually disjoint *atomic* components. These are the atoms of the Boolean algebra consisting of the closure of the sets under union, intersection and complement. To illustrate this idea, suppose the two sets *sons* and *daughters* are specified as subsets of *children.* From

$$|sons| \geq 2 \wedge |daughters| \geq 3 \tag{9}$$

one can deduce $|children| \geq 5$. The three sets can overlap in the most general way as depicted in Figure 1. There are seven different areas (together with the

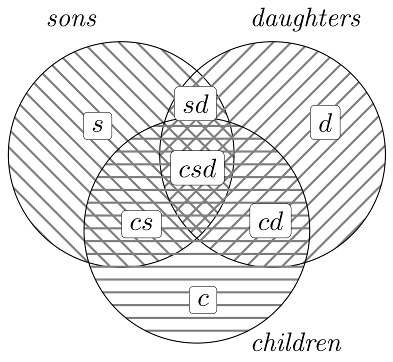


Figure 1: A general set structure

complement of the hatched areas there are in fact $2^3 = 8$ different areas) named c, s, d, cs, cd, sd, and csd. The informal meaning is

c = children, not sons, not daughters.
s = sons, not children, not daughters.
d = daughters, not children, not sons.
cs = children, which are sons, not daughters.
cd = children, which are daughters, not sons.
sd = sons, which are daughters, not children.
csd = children, which are both sons and daughters.

The original sets can now be obtained from their 'atomic' components:

$$\begin{aligned} children &= c \cup cs \cup cd \cup csd \\ sons &= s \cup cs \cup sd \cup csd \\ daughters &= d \cup cd \cup sd \cup csd. \end{aligned}$$

Moreover, since this decomposition is mutually disjoint and exhaustive, the cardinalities of the sets just add up:

$$\begin{aligned}|children| &= |c| + |cs| + |cd| + |csd| \\ |sons| &= |s| + |cs| + |sd| + |csd| \\ |daughters| &= |d| + |cd| + |sd| + |csd|.\end{aligned}$$

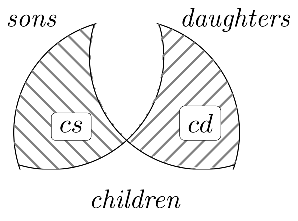
The relationships between *children*, *sons* and *daughters* can actually be specified in propositional logic:

$sons \Rightarrow children$	sons are children	
$daughters \Rightarrow children$	daughters are children	
$sons \wedge daughters \Rightarrow \bot$	there are no hermaphrodites	(10)
$children \Rightarrow sons \vee has\text{-}daughter$	children consist only of sons and daughters	

These formulae have three propositional models:

$$\begin{array}{l} children, sons, \neg daughters \\ children, daughters, \neg sons \\ \neg children, \neg sons, \neg daughters. \end{array}$$

They correspond to the two non-empty sets *cs* and *cd* in the figure below, together with the surrounding area.



The fact that these are the only models means $|c| = 0$, $|s| = 0$, $|d| = 0$, $|sd| = 0$, $|cd| = 0$ and $|csd| = 0$. The problem of determining whether there are at least 5 children can now be reformulated

$$|cs| \geq 2 \wedge |cd| \geq 3 \Rightarrow |cs| + |cd| \geq 5. \tag{11}$$

Since the sets are mutually disjoint, the internal structure of 'cardinality terms' like $|cs|$ is no longer relevant, and $|cs|$ can be replaced with a non-negative integer-valued variable x_{cs}. We obtain

$$x_{cs} \geq 2 \wedge x_{cd} \geq 3 \Rightarrow x_{cs} + x_{cd} \geq 5. \tag{12}$$

which is trivial to check.

Decomposition for Boolean Algebra Terms

In [7] this idea was developed into a general methodology for augmenting formal systems with a Boolean algebra component. The general methodology works for formal systems whose language has a notion of (existentially quantified) variables. A typical example is a mathematical programming system for solving equations and in-equations. On the syntactic side, this formal system can be extended with set terms embedded in *bridging functions* at variable positions. The bridging functions map objects of one logic to objects of another logic. For example, if the basic system allows for equations like

$$2 \cdot x + 3 \cdot y = 5$$

then the extended system would allow for equations like

$$2 \cdot |\mathit{sons} \cup \mathit{friends}| + 3 \cdot \mathit{max\text{-}age}(\mathit{friends} \setminus \mathit{sons}) = 5.$$

$\mathit{sons} \cup \mathit{friends}$ and $\mathit{friends} \setminus \mathit{sons}$ are Boolean set terms. The cardinality function $|..|$ and the *max-age* function are bridging functions. Both map sets to numbers such that multiplication with 2 and 3 is defined. In the general setting, bridging functions map the sets to objects in the basic system which make sense there.

The relationships between the sets can be axiomatized in propositional logic. (10) is an example for such a propositional axiomatization. It exploits the fact that the elements and connectives of Boolean algebras can always be interpreted as sets and the corresponding set operations (Stone's representation theorem [11]). With some elementary Boolean algebra theory one can show that the models of the propositional axiomatization correspond to the atoms of the Boolean algebra generated by the closure of the sets under union, intersection and complement.[1]

This correspondence can be turned into an algorithm for eliminating the Boolean terms and the bridging functions. In the first step we compute a syntactic representation of the models of the propositional axiomatization. The Boolean terms t can now be decomposed into the atomic components

$$\{m_1, \ldots, m_n\} \stackrel{\text{def}}{=} \{m \mid m \text{ is a model for the axioms and } m \text{ satisfies } t\}.$$

This way all Boolean terms t embedded in a bridging function $f(t)$ can be rewritten into $f(\{m_1, \ldots, m_n\})$.[2]

In the next step of the decomposition method we must use an extra assumption about bridging functions. They must be *additive*. This means, if two sets x

[1] A Boolean algebra is a non-empty set equipped with the functions $\sqcap$ (meet), $\sqcup$ (join) and $'$ (inverse), a smallest element $\bot$ and a largest element $\top$. A $\leq$-relation is definable as: $x \leq y$ iff $x \sqcap y = x$. Set algebras where $\sqcap$ is intersection, $\sqcup$ is union and $'$ is complement and $\leq$ is the subset relation, is one particular kind of Boolean algebra. Every Boolean algebra, however, is equivalent to a set algebra. A Boolean algebra is *complete* iff all (finite and infinite) joins belong to it. It is *atomic* iff every element can be obtained as the join of a set of smallest elements above $\bot$, the *atoms*. The atoms in set algebras are the singleton sets. Finite Boolean algebras are always complete and atomic (cf. any textbook on Boolean algebras).

[2] The m_i are conjunctions of positive or negative Boolean variables. But for most purposes it is sufficient to take the m_i as *names* for the models.

and y are disjoint then it must be possible to compute $f(x \cup y)$ by first computing $f(x)$ and $f(y)$ and then joining the results with some *combination function*. Examples where this is fulfilled are:

$$\begin{array}{lll}
x \cap y = \emptyset & \Rightarrow & |x \cup y| = |x| + |y| \\
x \cap y = \emptyset & \Rightarrow & \textit{max-age}(x \cup y) = \max(\textit{max-age}(x), \textit{max-age}(y)) \\
x \cap y = \emptyset & \Rightarrow & \textit{average-age}(x \cup y) = \dfrac{|x|\textit{average-age}(x) + |y|\textit{average-age}(y)}{|x| + |y|}.
\end{array}$$

The additivity of the bridging functions and the fact that the atoms m_i all denote disjoint sets, allows us to rewrite terms $f(\{m_1, \ldots, m_n\})$ into $g(f(m_1), \ldots, f(m_n))$ where g is a *composition function* like $+$ or max.

In the last step we replace terms $f(m_i)$ with new variables $x_{f(m_i)}$ of the basic system. (11) $\rightarrow$ (12) is such a replacement.

The transformations are sound and complete. This means that the original problem in the mixed language has a solution if and only if the transformed problem has a solution in the basic system.

Towards The General Combination Methodology

Closer examination of the atomic decomposition technique reveals that the key point which makes it work is *not* the fact that we are dealing with an atomic Boolean algebra. The key step is the partitioning of the domain into disjoint parts whose internal structure is irrelevant for the problem at hand. Once the disjoint parts are known, one can push down the bridging functions to the 'atomic' level and then replace the 'atomic bridge terms' with numeric variables.

In the Boolean Algebra case the partitioning was obtained from the models of the given axioms. These models can be computed in a particular way, and this way can be generalized to the non-Boolean Algebra case. To illustrate this, suppose we have some axioms $\mathcal{S}$ and some bridge terms containing the two propositional variables p and q. The possible models are $\{p, q\}$,$\{p, \neg q\}$, $\{\neg p, q\}$, $\{\neg p, \neg q\}$. Which of them are the real models can actually be computed by checking which formulae among $p \wedge q$, $p \wedge \neg q$, $\neg p \wedge q$, $\neg p \wedge \neg q$ are consistent with $\mathcal{S}$. The ones which are inconsistent cannot describe models for $\mathcal{S}$. What is actually needed for doing this check is therefore a conjunction operator and a negation function in the language (with the usual semantics), and a consistency test for formulae of the kind $\mathcal{S} \wedge p \wedge \neg q$. It is not even necessary for p and q to be propositional variables. They can be any formula of the given set description language. What only counts is that $p \wedge q$, $p \wedge \neg q$, $\neg p \wedge q$, $\neg p \wedge \neg q$ determine a partitioning of the domain, and this is guaranteed, regardless what p and q is. The formulae are pairwise inconsistent, therefore they denote sets whose intersection is empty, and the disjunction of all of them is a tautology, and therefore the union of the sets they denote is the whole domain.

This suggests the following method for computing the atomic decomposition: Given is a set $\mathcal{S}$ of axioms in the set description language and a problem specification P in the mixed language. Suppose $\varphi_1, \ldots, \varphi_n$ are the non-Boolean SDL-subformulae occurring in P including the propositional variables. The atoms of the atomic decomposition are now determined by those formulae $\pm\varphi_1 \wedge \ldots \wedge \pm\varphi_n$ which are consistent with $\mathcal{S}$ ($\pm\varphi_i$ is either the unnegated or negated version of φ_i). Each φ_i is now decomposed into the disjunction (union) of the atoms containing φ_i positively. The rest is — almost — as in the Boolean Algebra case.

1.6 Correlations Between the Logical Language and the Arithmetical Language

Unfortunately, things are not so straightforward when the set description language is more expressive than PL. The problem is that in more expressive set description languages the logical language itself may be able to express constraints on the size of the set-interpretation of formulae. In a description logic, for example, the following constraint holds

$$|\varphi| = 0 \Rightarrow |\exists\, r.\varphi| \tag{13}$$

(If the set φ is empty then the set of objects having some r-successors in φ must be empty too.) That means if the arithmetic solver returns a solution of the translated problem with $x_{|\varphi|} = 0$ and $x_{|\exists\, r.\varphi|} \neq 0$, this is inconsistent!

Even worse, some description logics are able to express constraints on the cardinality of the domain. For example, if the language is able to represent the universal relation $\top$ then a formula *atmost* $5\ \top$ (denoting the set of objects having at most 5 $\top$-successors) implicitly fixes the size of the domain to $\leq\ 5$.

The most general method for dealing with this kind of correlations between the logical and arithmetical part of the language is by a kind of 'post mortem analysis' of the generated arithmetical solutions. If the arithmetic solver comes up with a solution $x_{|a_1|} = n_1, ...$ then one has to check whether there is really a model of the axioms where the 'atoms' a_i denote sets of cardinality n_i. This is possible in certain set description languages by a modified tableaux algorithm, but it is very inefficient. Unfortunately there is no better method which is independent of the set description language. For particular SDLs, however, one can either show that these kind of correlations do not occur, or one might take care of it by submitting extra constraints to the arithmetic solver. We shall investigate this further in Section 4.

1.7 A Motivating Example

We illustrate the whole procedure with an optimization example from marketing research. The example can be solved using Propositional Logic as set description language, but it indicates clearly that for more complex examples a more expressive set description language is needed.

The informal specification of the problem is:

For political polls concerning immigration laws a number of at least 2300 U.S. people have to be surveyed, at least 1000 of them have to *be 30 years or younger and 600 between 31 and 50. At least 15 % of the surveyed people have to live in a state bordering Mexico and not more than 20 % of those who are 51 or more live in a state bordering Mexico. The prices of interviewing people in each age and region category are 7.50 \$ for those younger than 30 and living in a state bordering Mexico and 6.90 \$ for those who do not live in a state bordering Mexico, and so on.*

The sets of people to be interviewed can be axiomatized as follows:

$$\begin{array}{lcl} 30^- & = & \mathit{int} \wedge \mathit{age} \leq 30 \\ 30^+ & = & \mathit{int} \wedge \neg(\mathit{age} \leq 30) \wedge \mathit{age} \leq 50 \\ 50^+ & = & \mathit{int} \wedge \neg(\mathit{age} \leq 50) \\ \mathit{bs} & = & \mathit{int} \wedge (\exists\, \mathit{resident\text{-}in.bordering\text{-}state}) \\ \mathit{nbs} & = & \mathit{int} \wedge \neg(\exists\, \mathit{resident\text{-}in.bordering\text{-}state}) \\ \mathit{int} & \neq & \mathit{bordering\text{-}state} \end{array} \tag{14}$$

where *int* denotes the set of people to be interviewed, *nb* denotes the set of people to be interviewed and living in the bordering states, *nbs* denotes the set of people to be interviewed not living in a bordering state, 30^- denotes the set of people younger than 30, 30^+ denotes the set of people between 30 and 50, and finally 50^+ denotes the set of people older than 50.

Terms like $\mathit{age} \leq 30$ or $\exists\, \mathit{resident\text{-}in.bordering\text{-}state}$ are clearly not propositional. For the purpose of this example, however, we can treat them as (funny looking) predicate names.

The arithmetic part of the specification can now be formulated using the notions introduced above:

$$\begin{array}{l} |\mathit{int}| \geq 2300, |30^-| \geq 1000, |30^+| \geq 600, \\ |\mathit{bs}| \geq 0.15 * |\mathit{int}|, \\ |50^+ \wedge \mathit{bs}| \leq 0.20 * |50^+|, \\ \mathit{costs}(\mathit{bs} \wedge 30^-) = 7.50 * |\mathit{bs} \wedge 30^-|, \\ \mathit{costs}(\mathit{bs} \wedge 30^+) = 6.80 * |\mathit{bs} \wedge 30^+|, \quad \ldots \end{array} \tag{15}$$

The problem is to find numbers for the cardinalities of the sets such that $\mathit{costs}(\mathit{int})$ is minimal.

The predicates to be decomposed are

$$\{\mathit{int}, \mathit{bordering\text{-}state}, \mathit{age} \leq 30, \mathit{age} \leq 50, \exists\, \mathit{resident\text{-}in.bordering\text{-}state}\}.$$

The set formulae to be decomposed are *int*, $\mathit{age} \leq 30$, $\mathit{age} \leq 50$ and $\mathit{bs} \overset{\text{def}}{=} \exists\, \mathit{resident\text{-}in.bordering\text{-}state}$ There are six atoms which are consistent with the above definitions of these notions:

$$\begin{array}{lcl} b30^- & = & \mathit{int} \wedge \mathit{bs} \wedge \mathit{age} \leq 30 \\ b30^+ & = & \mathit{int} \wedge \mathit{bs} \wedge \neg(\mathit{age} \leq 30) \wedge (\mathit{age} \leq 50) \\ b50^+ & = & \mathit{int} \wedge \mathit{bs} \wedge \neg(\mathit{age} \leq 50) \end{array}$$

together with $nb30^-$, $nb30^+$ and $nb50^+$, which are defined in the same way as above except with $\neg bs$ instead of bs. These six atoms correspond to the six different sets into which the set of people to be interviewed can be partitioned.

Using these six atoms one can decompose the set terms into sets of atoms, and the functions operating on Boolean terms – in this example the cardinality function and the *costs* function – into new number valued variables. For example the term *int* can be replaced by $b30^- \wedge nb30^- \wedge b30^+ \wedge nb30^+ \wedge b50^+ \wedge nb50^+$. The 'cardinality term' $|int|$ itself can therefore be replaced by a term denoting the sum of the cardinality of the atoms: $n_{b30^-} + n_{nb30^-} + n_{b30^+} + n_{nb30^+} + n_{b50^+} + n_{nb50^+}$. n_{b30^-} etc. are new non-negative integer valued variables. The *costs* terms are treated in a similar way. For example the term $costs(bs \wedge 30^-)$ is replaced with c_{b30^-} where c_{b30^-} is a positive real-valued variable. The result of all these transformations is

$$\begin{aligned}
&n_{b30^-} + n_{nb30^-} + n_{b30^+} + n_{nb30^+} + n_{b50^+} + n_{nb50^+} \geq 2300\\
&n_{b30^-} + n_{nb30^-} \geq 1000\\
&n_{b30^+} + n_{nb30^+} \geq 600\\
&n_{b30^-} + n_{b30^+} + n_{b50^+} = 0.15 * (n_{b30^-} + n_{nb30^-} + n_{b30^+} +\\
&\qquad\qquad\qquad\qquad\qquad n_{nb30^+} + n_{b50^+} + n_{nb50^+})\\
&n_{b50^+} \leq 0.2 * (n_{b50^+} + n_{nb50^+})\\
&c_{b30^-} = 7.50 * n_{b30^-} \qquad \mathit{etc.}
\end{aligned}$$

This is now a standard representation of an optimization problem with objective function $c_{b30^-} + c_{nb30^-} + c_{b30^+} + c_{nb30^+} + c_{b50^+} + c_{nb50^+}$ to be minimized. The solution is $c_{nb30^-} = 1000$, $c_{b30^+} = 600$, $c_{b50^+} = 140$, $c_{nb50^+} = 560$.

In this example it was not necessary to look into the internal structure of terms like $age \leq 30$ and $\exists\, resident\text{-}in.bordering\text{-}state$. If the examples become more complicated, however, propositional logic is not sufficient, and we must be able to deal with the full semantics of such expressions. This is the main motivation for this paper.

2 The Languages Involved

We need three components in the syntax, a language $\mathcal{L}_E$ of some basic arithmetical system E, a set description language $\mathcal{L}_S$ and a language of 'bridging functions' for combining the arithmetical and the set description language. In this part of the paper neither $\mathcal{L}_E$ nor $\mathcal{L}_S$ need be concrete languages. We only require some basic properties.

2.1 The Arithmetical Language

Almost any kind of arithmetical language $\mathcal{L}_E$ which allows for expressing constraints on number-valued variables, and for which consistency is decidable, is suitable for our purposes. Depending on the application and on the degree of automation we want to achieve, a few extra things in $\mathcal{L}_E$ are needed.

Definition 2.1 (The Arithmetical Language $\mathcal{L}_E$) *We call an arithmetical language* admissible *(for our purposes) iff*

1. *the domain of $\mathcal{L}_E$-interpretations is a fixed set $\mathbb{N}_E$ of numbers which must include the integers.*
2. *integers are valid $\mathcal{L}_E$-terms;*
3. *$\mathcal{L}_E$ has non-negative integer-valued variables;*
4. *all free variables in an $\mathcal{L}_E$-formula are implicitly existentially quantified;*
5. *$\mathcal{L}_E$ comes with a decidable notion of* consistency of a finite set of $\mathcal{L}_E$-formulae*;*
6. *a* solution, *i.e. an interpretation satisfying a consistent set of formulae can be represented as a (kind of) substitution consisting of*
   (a) *a set of variable assignments $x \mapsto n$ where n is a number and,*
   (b) *if $\mathcal{L}_E$ has free function symbols, a partial graph $f(\vec{n}_i) = m_i$ for each function symbol f.*
7. *solutions for a consistent finite set of $\mathcal{L}_E$-formulae can be enumerated.*

$\mathcal{L}_E$ is non-linear *iff multiplication between two or more variables is possible, otherwise $\mathcal{L}_E$ is linear.*

$\mathcal{L}_E$ has free functions *iff not only the standard arithmetical function symbols, but also other free function symbols are allowed.*

$\mathcal{L}_E$ has implications *iff formulae of the kind*

$$x_1 = n_1 \land \ldots \land x_m = n_m \Rightarrow x_{m+1} = n_{m+1}$$

are allowed where the x_i are variables and the n_i are numbers. ■

The non-negative integer-valued variables are needed for expressing information about the cardinality of sets. Free function symbols are needed if bridging functions with $\mathcal{L}_S$-terms as well as $\mathcal{L}_E$-terms occur in the problem formulation (cf. (16) below). Non-linear terms are generated in the automated proof procedure for partitioning consistency (Proc. 2.19).

Finally, implications are needed for two purposes. First of all they are used to state simple constraints like 'the cost of the empty set is 0' which is represented by the generated formula $x_{|a|} = 0 \Rightarrow x_{cost(a)} = 0$.

Furthermore they help filtering out certain inconsistent solutions if the set description language itself is expressive enough to state constraints on the cardinality of sets. An example might be the implication which is implicit in a description logic like $\mathcal{ALC}$: 'if there are no teacher at all then there cannot be people with children who are teacher'. It would be expressed as $x_{|teacher|} = 0 \Rightarrow x_{|\exists\, has\text{-}child.teacher|} = 0$.

Definition 2.2 ($\mathcal{L}_E$-solver) *A $\mathcal{L}_E$-solver $S_{\mathcal{L}_E}(E)$ is a sound (and, if possible, complete) procedure that takes as input a finite set E of $\mathcal{L}_E$-formulae and enumerates all solutions σ_i for E. The empty set of solutions indicates that E is inconsistent.* ∎

$\mathcal{L}_E$ may not only be a language of arithmetic equation and inequation systems. It can also have a means to state objective functions for expressing optimization problems. In this case we assume that the $\mathcal{L}_E$-solver generates the most optimal solution first, and it generates less and less optimal solutions if required to enumerate further solutions. That means, if a solution σ is rejected for an optimization problem E then the next best solution generated by $S_{\mathcal{L}_E}$ is an optimal solution for $E, \neg\sigma$.

It is sometimes helpful if the $\mathcal{L}_E$-solver can deal with disjunctions. Otherwise disjunctions must be simulated by case analysis.

In the sequel we consider only admissible arithmetic languages.

2.2 The Set Description Language

The second component of our combination methodology is the set description language $\mathcal{L}_S$.

Definition 2.3 (The Set Description Language $\mathcal{L}_S$) *We call a set description language $\mathcal{L}_S$* admissible *with respect to a arithmetical language $\mathcal{L}_E$ iff*

1. *$\mathcal{L}_S$ is an extension of Propositional Logic (with $\top$ for true and $\bot$ for false);*
2. *$\mathcal{L}_S$ and $\mathcal{L}_E$ do not share any non-logical symbols;*
3. *one can distinguish $\mathcal{L}_S$-*axioms *from ordinary $\mathcal{L}_S$-*formulae*;*
4. *$\mathcal{L}_S$ is equipped with a set theoretic semantics such that*
   (a) *for each $\mathcal{L}_S$-formula φ and $\mathcal{L}_S$-interpretation $\Im$, $\varphi^{\Im}$ is a subset of $\Im$'s domain $\mathcal{D}_{\Im}$.*
   We sometimes write $\Im, x \models \varphi$ for $x \in \varphi^{\Im}$ ($\Im$ satisfies *φ in x);*
   (b) *the propositional connectives are interpreted set theoretically ($\wedge$ as intersection, $\vee$ as union, $\neg$ as complement, $\bot$ as the empty set and $\top$ as the whole domain);*
   (c) *$\mathcal{L}_S$-axioms have a Boolean interpretation, i.e. they are either true or false in an $\mathcal{L}_S$-interpretation. An interpretation $\Im$ making a set $\mathcal{S}$ of $\mathcal{L}_S$-axioms true is called a* model *for $\mathcal{S}$. One can also say, $\Im$ satisfies $\mathcal{S}$, written $\Im \models \mathcal{S}$;*
   (d) *for a set $\mathcal{S}$ of $\mathcal{L}_S$-axioms and a $\mathcal{L}_S$-formula φ it is decidable whether $\mathcal{S}$ is* consistent *with φ. $\mathcal{S}$ is* consistent *with φ iff there is a model $\Im$ for $\mathcal{S}$ such that $\varphi^{\Im} \neq \emptyset$.* ∎

In the rest of this paper we consider only admissible set description languages $\mathcal{L}_S$.

A $\mathcal{L}_S$-formula φ is *closed* iff it has no free variables. Most set description languages have no variables at all. Their formulae are all closed.

Remark: $\mathcal{L}_S$-formulae will sometimes also be called $\mathcal{L}_S$-terms. This is because an $\mathcal{L}_S$-formula may appear as argument of a bridging function symbol. Arguments of function symbols are usually called *terms.* If an $\mathcal{L}_S$-formula is called *term* we therefore emphasize its role as an argument of a bridging function symbol.

Description Logics as SDL

The prototypic description logic is $\mathcal{ALC}$ [10]:

The syntax of $\mathcal{ALC}$-formulae is:

$$\mathcal{ALC} = \mathbb{C}|\mathcal{ALC} \wedge \mathcal{ALC}|\mathcal{ALC} \vee \mathcal{ALC}|\neg\mathcal{ALC}|\forall\,\mathbb{R}.\mathcal{ALC}|\exists\,\mathbb{R}.\mathcal{ALC}$$

where $\mathbb{C}$ is a set of *concept names* (propositional variables) and $\mathbb{R}$ is a set of *role names* (modal parameters).

An $\mathcal{ALC}$*-axiom* is an expression $\varphi = \psi$ or $\varphi \Rightarrow \psi$ where φ and ψ are $\mathcal{ALC}$-formulae.

A set of $\mathcal{ALC}$-axioms is usually called a *TBox.*[3]

An $\mathcal{ALC}$-interpretation $\Im = (\mathcal{D}_\Im, \mathcal{C}, \mathcal{R})$ consists of a non-empty domain $\mathcal{D}_\Im$, for each concept name $c \in \mathbb{C}$ a set $c^\Im = \mathcal{C}(c) \subseteq \mathcal{D}_\Im$, and for each role name $r \in \mathbb{R}$ an (accessibility) relation $r^\Im = \mathcal{R}(r) \subseteq \mathcal{D}_\Im \times \mathcal{D}_\Im$. $\wedge$ is interpreted as set union, $\vee$ as set intersection, $\neg$ as set complement. In addition we have

$$\begin{array}{rcl}(\forall\,r.\varphi)^\Im & = & \{x \mid \text{for all } (x,y) \in \mathcal{R}(r) : y \in \varphi^\Im\} \\ (\exists\,r.\varphi)^\Im & = & \{x \mid \text{there is } (x,y) \in \mathcal{R}(r) \text{ and } y \in \varphi^\Im\}.\end{array}$$

An $\mathcal{ALC}$-interpretation $\Im$ satisfies an $\mathcal{ALC}$-axiom $\varphi = \psi$ iff $\varphi^\Im = \psi^\Im$. It satisfies $\varphi \Rightarrow \psi$ iff $\varphi^\Im \subseteq \psi^\Im$.

It is well known that consistency of $\mathcal{ALC}$-formulae with respect to a TBox is decidable [10].

Many other operators have been used for description logics. Among them are the *number restriction* operators[4] with the semantics

$$\begin{array}{rcl}(\mathit{atleast}\,n\,r)^\Im & = & \{x \mid |\{y \mid xr^\Im y\}| \geq n\} \\ (\mathit{atmost}\,n\,r)^\Im & = & \{x \mid |\{y \mid xr^\Im y\}| \leq n\}\end{array}$$

[3]There is a one-to-one correspondence between $\mathcal{ALC}$-formulae and formulae in a multi-modal logic [9]. A TBox, however, consists of *ground* axioms which, from a modal point of view, are true in all worlds. TBox axioms are not to be mixed up with characteristic axioms for modal logics. TBox axioms are ground. They do not contain quantifications over predicates. This is usually not considered for modal logics.

[4]Description logics with qualified number restrictions correspond to modal logics with graded modalities [12]

(set of objects with at least / at most n 'role fillers').

A variant are the *qualified number restriction* operators with the semantics

$$\begin{array}{lcl}(atleast\, n\, r.\varphi)^{\Im} & = & \{x \mid |\{y \mid xr^{\Im}y\} \cap \varphi^{\Im}| \geq n\} \\ (atmost\, n\, r.\varphi)^{\Im} & = & \{x \mid |\{y \mid xr^{\Im}y\} \cap \varphi^{\Im}| \leq n\}\end{array}$$

(set of objects with at least / at most n 'role fillers' in φ).

Another extension concerns the representation of the accessibility relations. Instead of just using role names, one can use complex role terms with Boolean operators, operators from relation algebras (; for composition and $^{-1}$ for the converse), and operators from dynamic logic (* for iteration, ? for test) etc. Moreover, the relationships between roles could be axiomatized (e.g. $r \Rightarrow s$ for subrole relationship, $r \wedge s = \bot$ for disjointness etc.) in a separate *RBox*.

Boolean axioms for roles are relatively easy to handle. Non-boolean axioms, however, may change the logic. $r; r \Rightarrow r$, for example, imposes transitivity on the relation r, which means that the r-part of the description logic is of K4 type, and requires special treatment in the decision procedures.

2.3 Bridging Functions

As a bridge between the two languages $\mathcal{L}_E$ and $\mathcal{L}_S$ we need a distinguished set $\mathcal{F}_B$ of *bridging functions*, different to all other symbols involved. One particular bridging function is the cardinality function $|.|$ mapping sets to integers.[5]

In the examples we have seen so far only one-place bridging functions appearded. *cost(x)*, *max-cost(x)*, *average-cost(x)* are one-place bridging functions. But nothing stops us from using n-place bridging functions. An example for using a 2-place bridging function might be

$$consumption(girls,chocolate) < 100$$

(the girls consume less than 100 units of chocolate).

In these cases all arguments are set terms. As a further generalization we can even have bridging functions where some arguments are set terms and some arguments are arithmetic terms.

$$salary(professors,2000) > 100000 \tag{16}$$

(the salary of the professors in the year 2000 is $>$ 100000), for example, requires a bridging function *salary* where the first argument is a set term and the second argument is a number or an arithmetic term. Since the atomic decomposition technique only eliminates the logical terms from mixed specifications, a formula like (16) will be translated into something like

$$salary'_{professors}(2000) > 10000$$

[5]We do not distinguish the cardinality function $|.|$ from the cardinality function symbol as a syntactic entity. This is possible because the interpretation of this symbol is fixed. It just denotes the caridinality function.

where *salary'$_{professors}$* is a free function symbol.

We can use these mixed-type bridging functions only if the arithmetic language $\mathcal{L}_E$ has free function symbols (Def. 2.1). For the general theory we assume this is possible. In concrete implementations one of course discard this possibility.

Definition 2.4 (Bridging Functions) *Let $\mathcal{F}_B$ be a set of* bridging function symbols, *different to all other symbols in the languages $\mathcal{L}_E$ and $\mathcal{L}_S$.*

$\mathcal{F}_B$ contains the distinguished cardinality function symbol $|.|$. This is the only function with a fixed interpretation. All the others are free function symbols.

A bridging function symbol may have any finite arity. Each argument position, however, can take either only $\mathcal{L}_E$-terms or only $\mathcal{L}_S$-terms as arguments. For convenience, we assume that a bridging function of arity $n+k$ reserves the first n arguments for $\mathcal{L}_S$-terms and the remaining k arguments for $\mathcal{L}_E$-terms.

A bridging function f with n $\mathcal{L}_S$ arguments and k $\mathcal{L}_E$-arguments is of argument type *$n+k$.* ■

Bridging functions can — and need be — axiomatized in a certain way. There are two groups of axioms, the *additivity axioms $\mathcal{A}_B$*, and the *general constraint axioms $\mathcal{C}_B$*.

2.4 The Additivity Axioms

The first group of axioms, the *additivity axioms*, are necessary for making the decomposition technique work. If a bridging function is applied to the union of two *disjoint* sets x and y, then it must be possible to apply the bridging function to x and y separately and then combine the result. For the set cardinality function for example, this is a very natural property: $|x \cup y| = |x| + |y|$ holds, provided x and y are disjoint. For other bridging functions a corresponding property must be explicitly stated.

Definition 2.5 (Additivity Axioms)
The additivity axioms $\mathcal{A}_B$ for a bridging function $f \in \mathcal{F}_B$ with arity $n+k$ are:

$$\forall x, y, \vec{z}\ x \wedge y = \bot \Rightarrow f(\ldots, z_{i-1}, x \vee y, z_{i+1}, \ldots) \\ = g_i(f(\ldots, z_{i-1}, x, z_{i+1}, \ldots), f(\ldots, z_{i-1}, y, z_{i+1}, \ldots))$$

for each $i \in \{1, \ldots, n\}$ of the set argument positions. $g_i(x, y)$ is some term in the arithmetic language $\mathcal{L}_E$ with the following constraints:

1. *$g_i(x, y)$ is commutative and*
2. *the consequence parts $\varphi = \psi$ of the additivity axioms must form a* confluent *system of rewrite rules $\varphi \rightarrow \psi$.*[6]

[6]Confluence is necessary because we want to use the additivity axioms as rewrite rules, and it must not matter which argument position is rewritten first. Confluence is automatically guaranteed if there is only one single $\mathcal{L}_S$-argument, or if all g_i are the same for one function f and g_i is commutative and associative.

$g_i(x,y)$ is the distribution function *for the i-th argument of f.* ■

Example 2.6 *If $x \cap y = \emptyset$ then $|x \cup y| = |x| + |y|$. Therefore the bridging function 'cardinality' is additive and the function g_1 in the definition above is the addition of numbers.*

Other examples are:

$$x \wedge y = \bot \Rightarrow$$

$$\begin{aligned}
\text{cost}(x \vee y) &= \text{cost}(x) + \text{cost}(y) \wedge \\
\text{max-cost}(x \vee y) &= max(\text{max-cost}(x), \text{max-cost}(y)) \wedge \\
\text{min-cost}(x \vee y) &= min(\text{min-cost}(x), \text{min-cost}(y)) \wedge \\
\text{average-cost}(x \vee y) &= \frac{|x|\text{average-cost}(x) + |y|\text{average-cost}(y)}{|x| + |y|} \wedge \\
\text{consumption}(x \vee y, z) &= \text{consumption}(x, z) + \text{consumption}(y, z) \wedge \\
\text{consumption}(z, x \vee y) &= \text{consumption}(z, x) + \text{consumption}(z, y)
\end{aligned}$$

■

The distribution functions $+, \max, \min$ and the construction for 'average' seem to be very natural choices. Therefore it is worth giving them special names.

Definition 2.7 (Distribution Types) *A bridging function f for which the distribution function g_i (Def. 2.5) for the i-th argument is $+$ (*max*,* min*) is called of* distribution type $+$ (max, min) *in the i-th argument.*

If $g_i(x,y) = \frac{|x|f(\ldots x \ldots) + |y|f(\ldots y \ldots)}{|x|+|y|}$ then it is called of distribution type 'average' in the i-th argument. ■

Consequences of the Additivity Axioms

Two sets x and y are disjoint if one of them is the empty set. Therefore if either x or y is the empty set then the implications of the additivity axioms must hold. This way we get some information about the behaviour of the bridging functions on empty sets. For example, if f's distribution type is $+$ then

$$f^{\Im}(x) = f^{\Im}(x \cup \emptyset) = f^{\Im}(x) + f^{\Im}(\emptyset)$$

which means

$$f^{\Im}(\emptyset) = 0.$$

If, on the other hand, f's distribution type is max then

$$f^{\Im}(x) = f^{\Im}(x \cup \emptyset) = \max(f^{\Im}(x), f^{\Im}(\emptyset))$$

which is equivalent to

$$f^{\Im}(x) \geq f^{\Im}(\emptyset)$$

for all x. If f's distribution type is min then $f^{\Im}(x) \leq f^{\Im}(\emptyset)$ is a consequence. Finally, if f's distribution type is 'average' then

$$f^{\Im}(x) = f^{\Im}(x \cup \emptyset) = \frac{|x|f^{\Im}(x) + |\emptyset|f^{\Im}(\emptyset)}{|x| + |\emptyset|} = f^{\Im}(x),$$

i.e. the distribution type 'average' has no consequences on the value of $f^{\Im}(\emptyset)$.

Definition 2.8 (Emptyness Axioms for Bridging Functions) *Let $\mathcal{A}_B$ be the additivity axioms for the bridging functions $\mathcal{F}_B$. We obtain the* emptyness axioms *$\mathcal{E}_B$ for $\mathcal{F}_B$ by replacing the y in the implication part in each additivity axioms with $\perp$, .i.e.*

$$f(\ldots, z_{i-1}, x, z_{i+1}, \ldots) = g_i(f(\ldots, z_{i-1}, x, z_{i+1}, \ldots), f(\ldots, z_{i-1}, \perp, z_{i+1}, \ldots)),$$

and simplifying the resulting formula if possible. ■

For the four distribution types $+$, max, min and average we get the emptyness axioms:

$$\begin{array}{rcll} f(\ldots, z_{i-1}, \perp, z_{i+1}, \ldots) & = & 0 & \text{if } g_i = + \\ f(\ldots, z_{i-1}, x, z_{i+1}, \ldots) & \geq & f(\ldots, z_{i-1}, \perp, z_{i+1}, \ldots) & \text{if } g_i = \max \\ f(\ldots, z_{i-1}, x, z_{i+1}, \ldots) & \leq & f(\ldots, z_{i-1}, \perp, z_{i+1}, \ldots) & \text{if } g_i = \min \\ & \top & & \text{if } g_i = \mathit{average} \end{array} \quad (17)$$

The emptyness axioms must be part of the constraint axioms (Def. 2.13) for the bridging functions.

Remark: The composition function min imposes $f^{\Im}(x) \leq f^{\Im}(\emptyset)$. Therefore one should better leave $f^{\Im}(\emptyset)$ undefined.

2.5 Partitionability

Bridging functions f of argument type $n+k$ are interpreted as functions mapping n subsets of a domain $\mathcal{D}_{\Im}$ of an $\mathcal{L}_S$-interpretation $\Im$ and k numbers to a number, i.e.

$$f^{\Im} \in ({\mathcal{D}_{\Im}}^n \times {\mathbb{N}_E}^k) \mapsto \mathbb{N}_E.$$

For technical reasons we need to require a property of the bridging functions we call *partitionability*. It is actually a property of the additivity axioms which seems to hold in general, but which seems not to be provable in general. The problem is that the proofs require a construction which depends on the distribution function g in the additivity axioms.

The motivation for this property comes of course from the atomic decomposition method we want to employ.

With atomic decomposition we partition the domain of discourse, translate the problem at hand into a pure $\mathcal{L}_E$-problem and generate solutions for the $\mathcal{L}_E$-problem. These solutions represent values for terms like $f(a_i)$ where for example f is a bridging function of type $1 + 0$ and the a_i are the atoms of the decomposition. In order to prove that the generated solution is consistent with the additivity axioms, one needs not only values of f for the finitely many atoms, but one needs to construct an interpretation $\Im$ for the bridging functions where $f^{\Im}$ has values for all subsets of $\Im$'s domain. That means, from a few values of $f^{\Im}$'s graph one needs to to be able to reconstruct the full graph of $f^{\Im}$ in such a way that the additivity axioms are satisfied. This construction depends on the distribution function g in the additivity axioms.

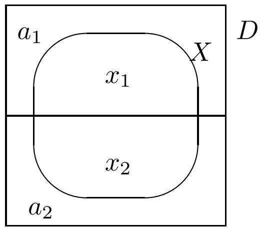
Definition 2.9 (Partitionability) *The additivity axioms $\mathcal{A}_B$ for a set $\mathcal{F}_B$ of bridging functions are called* partitionable *if for every* finite *non-empty set D and every partitioning P_D of D one can construct for every bridging function $f \in \mathcal{F}_B$ of argument type $n+k$ from a partial graph*

$$\{f^{\Im}(\vec{a},\vec{n}) = m \mid \vec{a} \in (P_D \cup \{\emptyset\})^n, \vec{n} \in {\mathbb{N}_E}^k, m \in \mathbb{N}_E \textit{ for some } m\},$$

a full interpretation $\Im$ for the bridging functions such that $\Im$ satisfies the additivity axioms. Let us call $\Im$ the interpolated model. ■

We show that partitionability is automatically guaranteed if the bridging functions are of distribution type (Def. 2.7) $+, \max, \min$ or average.

Example 2.10 (for Interpolation Models) *We illustrate the construction of the interpolation model for a one-place function f if the partitioning of the domain D into two partitions a_1 and a_2 looks like in the picture below.*



Suppose $f^{\Im}$ is defined for a_1 and a_2 only. $f^{\Im}(X)$ has to be defined for all subsets X of D. X can be partitioned into $x_1 = a_1 \cap X$ and $x_2 = a_2 \cap X$.
Let g be f's distribution type The interpolation model is

$$\begin{array}{lcll} f^{\Im}(X) & \stackrel{\text{def}}{=} & \frac{|x_1|}{|a_1|} f^{\Im}(a_1) + \frac{|x_2|}{|a_2|} f^{\Im}(a_2) & \textit{if } g = + \\ f^{\Im}(X) & \stackrel{\text{def}}{=} & \max(f(a_1), f(a_2)) & \textit{if } g = \max \\ f^{\Im}(X) & \stackrel{\text{def}}{=} & \min(f(a_1), f(a_2)) & \textit{if } g = \min \\ f^{\Im}(X) & \stackrel{\text{def}}{=} & \frac{|x_1| f(a_1) + |x_2| f(a_2)}{|x_1| + |x_2|} & \textit{if } g = \textit{average} \end{array}$$

It is an easy exercise to verify that this definition satisfies the corresponding additivity axioms. ■

Proposition 2.11 (Partitionability) *If the bridging functions are of distribution type (Def. 2.7) $+, \max, \min$ or average in all the set argument positions then the additivity axioms $\mathcal{A}_B$ are partitionable.*

Proof: Let $f \in \mathcal{F}_B$ be of type $n+k$. Let $P = \{a_1, \ldots, a_m\}$ be a partitioning of some domain $\mathcal{D}_{\Im}$ of a (partial) interpretation for the bridging functions. Let $f^{\Im}(\vec{a}, \vec{z}) = m_{a,z}$ be given for all $\vec{a} \subseteq (P \cup \{\emptyset\})^n$ and $\vec{z} \subseteq {\mathbb{N}_E}^k$.

We start with the first argument of f and define $f^{\Im}(X, \vec{a}, \vec{z})$ for all subsets X of $\mathcal{D}_{\Im}$.

Suppose $X = x_{j_1} \cup \ldots \cup x_{j_l}$ where $x_{j_i} = (a_{j_i} \cap X) \neq \emptyset$, $a_{j_i} \in P$. This is a partitioning of X into the parts intersecting with the partitions of P.

We define

$$\begin{array}{ll}
f^{\Im}(X, \vec{a}, \vec{z}) \stackrel{\text{def}}{=} \Sigma_{i=1}^{l} \frac{|x_{j_i}|}{|a_{j_i}|} f^{\Im}(a_{j_i}, \vec{a}, \vec{z}) & \text{if } g_1 = + \\
f^{\Im}(X, \vec{a}, \vec{z}) \stackrel{\text{def}}{=} \max(f^{\Im}(a_{j_i}, \vec{a}, \vec{z}) \mid i = 1, \ldots, l) & \text{if } g_1 = \max \\
f^{\Im}(X, \vec{a}, \vec{z}) \stackrel{\text{def}}{=} \min(f^{\Im}(a_{j_i}, \vec{a}, \vec{z}) \mid i = 1, \ldots, l) & \text{if } g_1 = \min \\
f^{\Im}(X, \vec{a}, \vec{z}) \stackrel{\text{def}}{=} \frac{\Sigma_{i=1}^{l} |x_{j_i}| f^{\Im}(a_{j_i}, \vec{a}, \vec{z})}{|X|} & \text{if } g_1 \ = \ \text{`average'}.
\end{array} \tag{18}$$

Notice that this definition is compatible with the given partial function graph if $X = a$ for one of the partitions a. For $X = \emptyset$, $f^{\Im}(\emptyset, \vec{a}, \vec{z})$ is known as well.

Once $f^{\Im}(X, \vec{a}, \vec{z})$ is defined for all sets X, one can define $f^{\Im}(X, Y, \vec{a}', \vec{z})$ in the same way for all sets Y in the second argument and so on. This way the full function graph for $f^{\Im}(\vec{X}, \vec{z})$ is built up for all $\mathcal{L}_S$-arguments.

We have to show that this construction satisfies the additivity axioms. To simplify notation we show it for a bridging function with one argument only. The proof for a function with n $\mathcal{L}_S$-arguments is just an n-fold repetition of this proof.

Let X and Y be two disjoint sets. Let $\{x_i \cup y_i,\ i = 1, \ldots l\}$ be the P-partitioning[7] of $X \cup Y$. $\{x_i,\ i = 1, \ldots l\}$ is a P-partitioning of X (where some of the x_i may be empty) and y_i, $\{i = 1, \ldots l\}$ is a P-partitioning of Y (where some of the y_i may also be empty).

Case $g = +$:

$$\begin{array}{rcl}
f^{\Im}(X \cup Y) & = & \Sigma_i \frac{|x_i \cup y_i|}{|a_i|} f^{\Im}(a_i) \\
& = & \Sigma_i \frac{|x_i| + |y_i|}{|a_i|} f^{\Im}(a_i) \\
& = & \Sigma_i \frac{|x_i|}{|a_i|} f(a_i) + \frac{|y_i|}{|a_i|} f^{\Im}(a_i) \\
& = & f^{\Im}(X) + f^{\Im}(Y).
\end{array}$$

Case $g = \max$:

$$\begin{array}{rcl}
f^{\Im}(X \cup Y) & = & \max(f^{\Im}(a_i) \mid i = 1, \ldots, n) \\
& = & \max(\max(f^{\Im}(a_i) \mid i = 1, \ldots, n, X \cap a_i \neq \emptyset), \\
& & \qquad \max(f^{\Im}(a_i) \mid i = 1, \ldots, n, Y \cap a_i \neq \emptyset)) \\
& = & \max(f^{\Im}(X), f^{\Im}(Y)).
\end{array}$$

Case $g = \min$: this case is similar to the max-case.

Case g = 'average':

[7]For a partitioning $P = \{a_1, \ldots, a_m\}$ we define the P-partitioning of a set X to be $\{X \cap a_1, \ldots X \cap a_m\}$.

$$\begin{aligned} f^{\Im}(X \cup Y) &= \frac{\Sigma_i |x_i \cup y_i| f^{\Im}(a_i)}{|X \cup Y|} \\ &= \frac{\Sigma_i (|x_i| + |y_i|) f^{\Im}(a_i)}{|X| + |Y|} \\ &= \frac{\Sigma_i |x_i| f^{\Im}(a_i) + \Sigma_i |y_i| f^{\Im}(a_i)}{|X| + |Y|} \\ &= \frac{|X| \frac{\Sigma_i |x_i| f^{\Im}(a_i)}{|X|} + |Y| \frac{\Sigma_i |y_i| f^{\Im}(a_i)}{|Y|}}{|X| + |Y|} \\ &= \frac{|X| f^{\Im}(X) + |Y| f^{\Im}(Y)}{|X| + |Y|}. \end{aligned}$$

Thus, in all four cases the corresponding additivity axioms are satisfied. ■

The construction of the interpolated model for the four distribution types $+, \max, \min$ and 'average' will be needed in examples below, but only for sets x which are subsets of a single partition. Therefore it is worth a separate definition.

Definition 2.12 (Special Interpolated Model) *If a one-place bridging function has one of the four distribution types* $+, \max, \min$ *and 'average' and* x *is a subset of a single partition* a *then*

$$\begin{aligned} f^{\Im}(x) &= \frac{|x|}{|a|} f^{\Im}(a) \quad && \textit{if } g = + \\ f^{\Im}(x) &= f^{\Im}(a) && \textit{if } g = \max \textit{ or } \min \textit{ or 'average'} \end{aligned}$$

holds for the interpolated model $\Im$.

The term $I_f(x, a)$ *which defines* $f^{\Im}(x)$ *in terms of a single partition* a *is called the* singleton interpolation term. *For example in the* $+$*-case above* $I_f(x, a) = \frac{|x|}{|a|} f(a)$. ■

2.6 Constraint Axioms for the Bridging Functions

Besides the additivity axioms it may be necessary to state further properties of the bridging functions. For example, one might want to make sure that salaries (cf. (16)) never decrease, i.e.

$$x < y \Rightarrow \mathit{salary}(z, x) < \mathit{salary}(z, y) \tag{19}$$

should hold for all years x, y, and sets z.

Another example is

$$\mathit{min\text{-}cost}(x) \leq \mathit{average\text{-}cost}(x) \leq \mathit{max\text{-}cost}(x) \tag{20}$$

which ensures at least a basic correlation between these three functions (although the prefixes 'min', 'average', 'max' of the function names suggest a particular meaning, on the atomic level they are just free function symbols without built in semantics). Properties like these can be stated as *bridging function constraint axioms.*

Definition 2.13 (Bridging Function Constraint Axioms)
The bridging function constraint axioms are $\mathcal{L}_E$ formulae where the $\mathcal{L}_E$-variables might be replaced with bridging function terms $f(\vec{x}, \vec{e})$ whose $\mathcal{L}_S$ argument positions are occupied with variables or $\perp$. All variables are universally quantified. ■

(19) and (20) satisfy this definition.

Definition 2.14 (Bridging Function Theory) *For a set $\mathcal{F}_B$ of bridging functions we define a* bridging function theory $\mathcal{T}_B \overset{\text{def}}{=} (\mathcal{A}_B, \mathcal{C}_B)$ *to consist of the additivity axioms (Def. 2.4) $\mathcal{A}_B$ and some* bridging function constraint axioms $\mathcal{C}_B$ *which must include the emptyness axioms $\mathcal{E}_B$ (Def. 2.8).* ■

The next definition formalizes an intermediate result in the process of solving a mixed problem via atomic decomposition: a solution of the decomposed problem has been obtained which yields values for the bridging functions applied to atoms (the solution 'satisfies the partition instances of the constraint axioms') We need to argue that this partial solution is sufficient.

Definition 2.15 (Satisfying Partition Instances) *Let $\Im$ be a (partial) interpretation for bridging function constraint axioms $\mathcal{C}_B$ and P be a partitioning of $\Im$'s domain $\mathcal{D}_\Im$. Let $Ax[\vec{x}] \in \mathcal{C}_B$ be a constraint axioms with n free set variables $\vec{x}$.*

$\Im$ satisfies the partition instances *of $Ax[\vec{x}]$ iff $\Im[\vec{x} \mapsto \vec{a}]$ satisfies $Ax[\vec{x}]$ for all $\vec{a} \subseteq P^n$. ($\Im[\vec{x} \mapsto \vec{a}]$ is like $\Im$, but maps all $\vec{x}$ to $\vec{a}$).* ■

In every automated reasoning application it is necessary that the axioms used to prove a theorem are consistent, otherwise the theorem is pointless. In the case of the bridging function theory, it is of course therefore also necessary that its axioms are consistent. Unfortunately, again by technical reasons, we need a stronger consistency property for the atomic decomposition technique to work. We call this property *partitioning consistency.* It means that every interpolated model (Def. 2.9) satisfies the constraint axioms $\mathcal{C}_B$.

Definition 2.16 (Partitioning Consistency)
A bridging function theory $\mathcal{T}_B \overset{\text{def}}{=} (\mathcal{A}_B, \mathcal{C}_B)$ *for partitionable additivity axioms $\mathcal{A}_B$ is* partitioning consistent *if every interpolated model (Def. 2.9) which satisfies the partition instances of the constraint axioms $\mathcal{C}_B$ also satisfies the general constraint axioms $\mathcal{C}_B$.* ■

We illustrate this property by two examples, one with a partitioning consistent constraint axiom and one with a partitioning inconsistent constraint axiom. The two examples illustrate that partitioning consistency is no general property, but has to be proved case by case.

Example 2.17 (for Partitioning Consistency) *Let I (for Income) and S (for Spending) be two one-place bridging functions. Their additivity axioms are*

$$\begin{array}{rcl} I(x \vee y) & = & I(x) + I(y) \\ S(x \vee y) & = & S(x) + S(y) \end{array}$$

for disjoint x and y.

Let $P = \{a_1, \ldots\}$ be a partitioning of some domain. Since the distribution function is + in both cases, the interpolated model is defined (Prop. 2.11):

$$\begin{aligned} I^{\Im}(X) &= \Sigma_i \tfrac{|x_i|}{|a_i|} I^{\Im}(a_i) \\ S^{\Im}(X) &= \Sigma_i \tfrac{|x_i|}{|a_i|} S^{\Im}(a_i) \end{aligned}$$

where $x_i = a_i \cap X$ for each a_i.

Example 1*: Consider the constraint axiom*

$$I(x) \geq S(x)$$

(the income is always greater than the spending). Suppose the interpretation $\Im$ satisfies all its partition instances, i.e. $I^{\Im}(a_i) \geq S^{\Im}(a_i)$ for all a_i.

This implies

$$\begin{aligned} & I^{\Im}(X) \geq S^{\Im}(X) \\ \textit{iff} \quad & \Sigma_i \tfrac{|x_i|}{|a_i|} I^{\Im}(a_i) \geq \Sigma_i \tfrac{|x_i|}{|a_i|} S^{\Im}(a_i) \\ \textit{iff} \quad & I^{\Im}(a_i) \geq S^{\Im}(a_i) \textit{ for all } a_i. \end{aligned}$$

which was assumed.

Example 2*: Now consider the constraint axiom*

$$I(x) \cdot S(x) = 100$$

(income times spending is always 100). Satisfaction of the partition instances $I^{\Im}(a_i) \cdot S^{\Im}(a_i) = 100$ for all a_i does by no means entail

$$I^{\Im}(X) \cdot S^{\Im}(X) = \Sigma_i \frac{|x_i|}{|a_i|} I^{\Im}(a_i) \cdot \Sigma_i \frac{|x_i|}{|a_i|} S^{\Im}(a_i) = 100.$$

Therefore this constraint axiom is not *partitioning consistent.* ■

Partitioning consistency must be ensured. The question is therefore whether it can be proved or disproved automatically. In fact, it is possible to prove partitioning consistency by induction and to generate the proof obligations fully automatically. The drawback is that the generated proof obligations contain non-linear arithmetic terms. Therefore proving partitioning consistency can be fully automated only if the language $\mathcal{L}_E$ is non-linear (Def. 2.1).

The idea for the induction proof is explained first for an axiom $Ax[f(x), g(x)]$ which has just one free variable x and, say, two bridging functions f and g, both having x as their only argument (remember the $\mathcal{L}_S$-arguments of the bridging functions in the constraint axioms are only variables). Assume f and g are of distribution type $+$.

The idea for the induction proof is now the following: We assume an arbitrary but fixed partioning $P = \{a_1, \ldots\}$. The assumption is that $Ax[f(a_i), g(a_i)]$

holds for all a_i. We want to prove that $Ax[f(x), g(x)]$ holds for all sets x. The base case of the induction is that x lies in just one part a of the partition. In this case we can use the interpolation definition of f and g (Def. 2.12):

$$f(x) = \tfrac{|x|}{|a|} f(a) \qquad g(x) = \tfrac{|x|}{|a|} g(a)$$

and substitute this into $Ax[f(x), g(x)]$:

$$Ax[\frac{|x|}{|a|} f(a), \frac{|x|}{|a|} g(a)]$$

Next we replace the non-arithmetic terms by new arithmetic variables, i.e. $|x|$ by $n_{|x|}$, $|a|$ by $n_{|a|}$, $f(a)$ by $n_{f(a)}$ and $g(a)$ by $n_{g(a)}$.

The proof obligation now becomes:

$$Ax[\frac{n_{|x|}}{n_{|a|}} n_{f(a)}, \frac{n_{|x|}}{n_{|a|}} n_{g(a)}]$$

has to be proved from the assumptions

$$Ax[n_{f(a)}, n_{g(a_i)}] \qquad \text{and} \qquad n_{|x|} \leq n_{|a|}.$$

All of these are (almost) $\mathcal{L}_E$-formulae and can therefore be submitted to the $\mathcal{L}_E$-solver. (If $\mathcal{L}_E$ does not allow divisions, one needs to multiply the denominator $n_{|a|}$ away).

With the base case we have proved that $Ax[f(x), g(x)]$ holds for all sets x which are subsets of one partition only.

As induction hypothesis, one can now assume that $Ax[f(x), g(x)]$ holds for all sets x which are subsets of $n-1$ partitions. For the induction step we split a set X into $X_{n-1} \cup x$ where X_{n-1} is a subset of the $n-1$ partitions and x is a subset of the n-th partition a.

Thus, the induction hypothesis is

$$Ax[f(X_{n-1}), g_(X_{n-1})] \quad \text{together with} \quad |x| \leq |a|$$

The formula to prove is

$$Ax[f(X_{n-1} \cup x), g(X_{n-1} \cup x)]$$

Since X_{n-1} and x are disjoint, we can apply the additivity axioms for f and g and get

$$Ax[f(X_{n-1}) + f(x), g(X_{n-1}) + f(x)].$$

Using the interpolation definition of f and g in the same way as in the base case, we get.

$$Ax[f(X_{n-1}) + \frac{|x|}{|a|} f(a), g(X_{n-1}) + \frac{|x|}{|a|} g(a)].$$

In the same way as in the base case we replace the bridging terms with arithmetic variables and get a pure $\mathcal{L}_E$-problem to be proved.

Example 2.18 (for Automated Partitioning Consistency Checking)
Let us illustrate the technique with the constraint axiom

$$I(x) \geq S(x) \tag{21}$$

we considered already in Example 2.17.

Base Case of the Induction*:*
Assumptions: $I(a) \geq S(a)$ *and* $|x| \leq |a|$
Theorem: $\frac{|x|}{|a|}I(a) \geq \frac{|x|}{|a|}S(a)$

After Replacements:
Assumptions: $n_{I(a)} \geq n_{S(a)}$ *and* $n_{|x|} \leq n_{|a|}$.
Theorem: $\frac{n_{|x|}}{n_{|a|}}n_{I(a)} \geq \frac{n_{|x|}}{n_{|a|}}n_{S(a)}$.

Simplified Theorem: $n_{I(a)} \geq n_{S(a)}$ *which is just the assumption.*

Induction step*:*
Assumptions: $I(X) \geq S(X)$, $I(a) \geq S(a)$ *and* $|x| \leq |a|$
Theorem: $I(X) + \frac{|x|}{|a|}I(a) \geq S(X) + \frac{|x|}{|a|}S(a)$

After Replacements:
Assumptions: $n_{I(X)} \geq n_{S(X)}$, $n_{I(a)} \geq n_{S(a)}$ *and* $n_{|x|} \leq n_{|a|}$.
Theorem: $n_{I(X)} + \frac{n_{|x|}}{n_{|a|}}n_{I(a)} \geq n_{S(X)} + \frac{n_{|x|}}{n_{|a|}}n_{S(a)}$.

The theorem follows from the assumptions, but it is a non-linear arithmetic problem. ■

Constraint axioms for the bridging functions may of course contain more than one $\mathcal{L}_S$-variable. If there are m such variables in an axiom $Ax[x_1, \ldots, x_m]$, one has to prove one base case (all m variables denote a subset of a single partition), and m induction steps, one for each variable. With the first induction one proves that $Ax[X_1, x_2, \ldots, x_n]$ holds for arbitrary sets X_1 and other sets $x_2, \ldots, x_n$, which are subsets of one partition only. This becomes an assumption for the second induction step, where one proves that $Ax[X_1, X_2, x_3 \ldots, x_n]$ holds for arbitrary sets X_1, X_2 and so on.

Notice that the $\mathcal{L}_E$-terms which might occur as arguments of bridging functions of type $n + k$ are not touched during the preparation of the proof obligations. They are submitted to the arithmetic solver as they are.

We formalize the method now for the general case.

Procedure 2.19 (for Proving Partitioning Consistency)
Let $Ax[f_1(\vec{y}_1), \ldots, f_l(\vec{y}_l)]$ *be a bridging function constraint axiom where the* f_i *are of argument type* $n_i + k_i$. *(Only the* $\mathcal{L}_S$*-variables are listed in the* $Ax[...]$ *notation.)*
Let $x_1, \ldots, x_m$ *be an enumeration of the* $\mathcal{L}_S$*-variables occurring in* Ax.

Generation of the Base Case Proof Obligation*:*

Step 1: *generate new constant symbols* $a_1, \ldots a_m$ *(for the partitions corresponding to the variables* $x_1, \ldots$*).*

Let $\sigma \overset{\text{def}}{=} \{x_1 \mapsto a_1, \ldots\}$ be a substitution replacing all $\mathcal{L}_S$-variables with the corresponding new constant symbols.
Let $\rho \overset{\text{def}}{=} \{f_1(\vec{y}_1) \mapsto I_{f_1}(\vec{y}_1, \vec{y}_1\sigma), \ldots\}$ be the rewrite rule which replaces all the bridging functions by their singleton interpolation terms (Def. 2.12).

Step 2: *prepare the assumptions:*
$A \overset{\text{def}}{=} Ax\sigma$ *and* $|x_1| \leq |a_1|, \ldots |x_m| \leq |a_m|$

Step 3: *prepare the theorem:*
$T \overset{\text{def}}{=} Ax\rho$

Step 4: *replace all bridging terms $f(\vec{s}, \vec{t})$ in the assumptions A and the theorem T with newly generated variables $x_{f(\vec{s})}$ (if the $\mathcal{L}_E$-part $\vec{t}$ is empty), or terms $f'_{f(\vec{s})}(\vec{t})$ with a fresh function symbol $f'_{f(\vec{s})}$.*
Step 5: *submit this problem to a suitable $\mathcal{L}_E$-solver.*

Generation of the Induction Step Proof Obligations*:*
For $i = 1, \ldots, m$ do:

Let $\sigma \overset{\text{def}}{=} \{x_i \mapsto a_i, \ldots, x_m \mapsto a_m\}$.

Step 1: *prepare the assumptions:*
$A \overset{\text{def}}{=} Ax\sigma \wedge |x_i| \leq |a_i|, \ldots |x_m| \leq |a_m|$

Step 2: *prepare the theorem:*
Let $T = A \overset{\text{def}}{=} Ax\{x_i \mapsto X \vee x, x_{i+1} \mapsto a_{i+1}, \ldots x_m \mapsto a_m\}$
Let T' be the result of applying the bridging axioms to T.
Let $T'' \overset{\text{def}}{=} T'\rho$.

Step 3: *replace all bridging terms in the assumptions and the theorem T'' by fresh variables in the same way as in the base step.*

Step 4: *submit this problem to an $\mathcal{L}_E$-solver.*

Return 'partitioning consistent' if all proof obligations could be proved. ■

This procedure is sound, i.e. Ax is partitioning consistent if the $\mathcal{L}_E$-solver proves all the generated proof obligations. But what about completeness? It turns out that completeness is not really an issue because the very notion of partitioning consistency is more or less an auxiliary notion which is sufficient but not necessary that the decomposition technique works.

To understand this, consider the constraint axiom $\mathit{cost}(x) \leq 100$ which is *not* partitioning consistent. (One may have $\mathit{cost}(a) = 100$ and $\mathit{cost}(b) = 100$ for two atoms a and b, thus satisfying the constraint axiom, but $\mathit{cost}(a \vee b) = 200$ violates it.)

Nevertheless, $\mathit{cost}(x) \leq 100$ is a meaningful statement. It expresses that the sum of the cost of *all* partitions is less than 100, which may very well be what one wants. The point is that this is actually not an intrinsic property of the bridging function *cost*, but a feature of the given problem. Such a statement should therefore not be formulated as a constraint axiom for the bridging function *cost*, but as part of the problem formulation: $\mathit{cost}(\top) \leq 100$. There it has exactly the desired effect.

That means the syntax of the bridging function constraint axioms allows one to formulate properties which actually don't belong there, but should be part of the problem formulation. Our bridging consistency test filters the constraint axioms out which definitely belong there, but this is essentially a heuristic filter. It might in theory be possible that one wants constraint axioms which get rejected by this test, and for which the decomposition method still works. Since this can't be guaranteed in general, we do not allow this.

It would be nice if the syntax of the bridging function constraint axioms could be restricted, such that only partitioning consistent axioms can be formulated in the first place, but this seems to be extremely difficult, or even impossible. For example $\mathit{cost}(x) \leq 100$ should be forbidden, but the syntactically more complex formula $\mathit{cost}(x) \leq 100 \cdot \mathit{spending}(x)$ should be allowed.

Since the bridging function constraint axioms are a kind of background theory which is to be used for many different concrete problem formulations, this is not really an important issue. In real applications one would build up a library of verified partitioning consistent constraint axioms and use only axioms from this library.

2.7 The Combined Language $\mathcal{L}_{ES}$

The combined language $\mathcal{L}_{ES}$ is the language for expressing arithmetical constraints about the cardinality and other numerical features of sets. $\mathcal{L}_{ES}$ is essentially like the arithmetic language $\mathcal{L}_E$, but $\mathcal{L}_E$-term positions can be occupied by $\mathcal{L}_S$-formulae embedded in a bridging function. The $\mathcal{L}_S$-part in the combined language is ground with respect to $\mathcal{L}_E$. That means there are no shared variables in different occurrences or even different arguments of the bridging functions.

Definition 2.20 (The Combined Language $\mathcal{L}_{ES}$)
If $s[\vec{x}] \in \mathcal{L}_E$ where $\vec{x}$ are variables in s, $f_i \in \mathcal{F}_B$ with arity n_i+k_i, $\vec{t_i}$ are n_i-tuples of closed $\mathcal{L}_S$-formulae, $\vec{s}_i$ are k_i-tuples of $\mathcal{L}_E$-terms then $s[x_1/f_1(\vec{t_1}, \vec{s}_1), x_2/...] \in \mathcal{L}_{ES}$. No other formulae are in $\mathcal{L}_{ES}$. ■

Since the language $\mathcal{L}_S$ and $\mathcal{L}_E$ and the bridging function symbols $\mathcal{F}_B$ do not share any non-logical symbols, we can define a combined interpretation $\Im$ as the union of an $\mathcal{L}_E$-interpretation $\Im_E$ and an $\mathcal{L}_S$-interpretation $\Im_S$ together with the interpretation of the bridging function symbols.[8]

The semantics of the combined language $\mathcal{L}_{ES}$ might look complicated, but it follows in fact very naturally from its components. Consider the formula

$$\mathit{cost}(\mathit{cars} \vee \mathit{bikes}) \geq 10000 \cdot |\mathit{cars} \vee \mathit{bikes}|$$

[8]There is a small technical problem. The Boolean connectives have a two-valued Boolean interpretation in $\mathcal{L}_E$ and for the bridging function axioms, but a set theoretic interpretation in $\mathcal{L}_S$. To be on the safe side, one could use different symbols in these three languages, but it seems that this confuses more than it helps. Since the three languages are separated, it is always clear which interpretation is meant.

(the total cost of the cars and bikes is greater than 10000 times the numbers of cars and bikes.)

cars and *bikes* is interpreted by the $\mathcal{L}_S$-part of the interpretation. They denote sets. $cars \vee bikes$ is an $\mathcal{L}_S$-formula. Therefore the $\vee$ is interpreted as set union. $cars \vee bikes$ now denotes a set S. *cost* is a bridging function and maps S to a number, say n. $|.|$ is the cardinality function and maps S also to a number, say m. We are left with interpreting $n \geq 1000 \cdot m$, which is done by the $\mathcal{L}_E$-part of the interpretation.

Definition 2.21 (Problem Specification)
A problem specification $P = (\mathcal{P}, \mathcal{S}, \mathcal{T}_B)$ *consists of the three parts:*

1. *a finite set $\mathcal{P}$ of $\mathcal{L}_{ES}$-formulae, the* problem formulation*,*
2. *a finite set $\mathcal{S}$ of $\mathcal{L}_S$-axioms,*
3. *a bridging function theory $\mathcal{T}_B$ (Def. 2.14) with partitionable additivity axioms (Def. 2.9) and partitioning consistent constraint axioms (Def. 2.16).* ■

(14) is an example for the $\mathcal{L}_S$-axioms $\mathcal{S}$. (15) is an example for the problem formulation $\mathcal{P}$.

Definition 2.22 ($\mathcal{L}_{ES}$-Models) *A $\mathcal{L}_{ES}$-interpretation $\Im$* satisfies, (is a model for) *a problem specification $P = (\mathcal{P}, \mathcal{S}, \mathcal{T}_B)$ iff*

1. *$\Im$ satisfies the bridging function theory $\mathcal{T}_B$ (remember that all free variables in $\mathcal{T}_B$ are universally quantified);*
2. *the $\mathcal{L}_S$-part of $\Im$ satisfies the $\mathcal{L}_S$-axioms $\mathcal{S}$ (Def. 2.3);*
3. *$\Im$ satisfies the problem formulation $\mathcal{P}$.* ■

3 Atomic Decomposition

In a sequence of transformations we eliminate the set part from the problem specification and turn the main problem to be solved into a pure problem in the arithmetic system E.

3.1 Atoms

In the first step we decompose the $\mathcal{L}_S$-formulae occurring as arguments of bridging functions in the problem formulation $\mathcal{P}$ into disjoint disjunctions of conjunctions (a special kind of disjunctive normal form which satisfies the premises of the additivity axioms.) Each part of the disjunctive normal form corresponds to an atom of a Boolean set algebra.

Definition 3.1 (Atoms) *Given a problem specification $P = (\mathcal{P}, \mathcal{S}, \mathcal{T}_B)$, let F_P be the set of all toplevel*[9] *propositional variables and non-Boolean subterms occurring in $\mathcal{P}$.*

[9]'toplevel' means that the terms are not in the scope of a non-Boolean operator.

The set A_{F_P} of syntactic atoms *over F_P is defined recursively:*

$$\begin{array}{rcl} A_\emptyset & = & \emptyset. \\ A_{X\cup\{\psi\}} & = & \{\varphi\wedge\psi \mid \varphi\in X, \varphi\wedge\psi \textit{ is consistent with } \mathcal{S}\} \cup \\ & & \{\varphi\wedge\neg\psi \mid \varphi\in X, \varphi\wedge\neg\psi \textit{ is consistent with } \mathcal{S}\}. \end{array}$$

■

Syntactic atoms are $\mathcal{L}_S$-formulae such that their interpretation yields a partitioning of the domain into subsets whose internal structure is supposed to be irrelevant for the given problem.

Lemma 3.2 (A_{F_P} is a Partitioning) *For each model $\Im$ of a problem specification $P = (\mathcal{P}, \mathcal{S}, \mathcal{T}_B)$, the set $\{a^\Im \mid a \in A_{F_P}\}$ is a partitioning of $\Im$'s domain $\mathcal{D}_\Im$.*

Proof: By induction on the construction of A_{F_P} one can easily prove that all $a^\Im$ are disjoint. The fact that the union of all the $a^\Im$ is $\Im$'s domain can be proved by induction on the number of elements in F_P. The base case is that F_P contains just one element ψ. If ψ is inconsistent with $\mathcal{S}$, i.e. $\psi^\Im = \emptyset$ then $(\neg\psi)^\Im = \mathcal{D}_\Im$. If $\neg\psi$ is inconsistent with $\mathcal{S}$, i.e. $(\neg\psi)^\Im = \emptyset$ then $\psi^\Im = \mathcal{D}_\Im$. If both are consistent then $A_{F_P} = \{\psi, \neg\psi\}$ and therefore $\psi^\Im \cup (\neg\psi)^\Im = \mathcal{D}_\Im$.
For the induction step we argue

$$(X\wedge\psi)^\Im \cup (X\wedge\neg\psi)^\Im = ((X\wedge\psi)\vee(X\wedge\neg\psi))^\Im = X^\Im = \mathcal{D}_\Im.$$ ■

Now we are ready to define a decomposition function that transforms an $\mathcal{L}_S$-formula into an equivalent representation formulated in terms of the syntactic atoms. The decomposition function is formulated in two steps. First we need a function which yields a set of syntactic atoms. The second decomposition function just turns this set into a disjunction.

Definition 3.3 (Decomposition Function $\mathcal{DC}$) *For a problem specification $P = (\mathcal{P}, \mathcal{S}, \mathcal{T}_B)$ and a $\mathcal{L}_S$-formula φ we define a decomposition function $\mathcal{DC}_{P_s}$:*

$$\begin{array}{rcll} \mathcal{DC}_{P_s}(\varphi) & \stackrel{\text{def}}{=} & \{a\in A_{F_P} \mid a = \ldots\wedge\varphi\wedge\ldots\} & \textit{if } \varphi\in F_P \\ \mathcal{DC}_{P_s}(\neg\varphi) & \stackrel{\text{def}}{=} & A_{F_P} \setminus \mathcal{DC}_{P_s}(\varphi) & \\ \mathcal{DC}_{P_s}(\varphi\wedge\psi) & \stackrel{\text{def}}{=} & \mathcal{DC}_{P_s}(\varphi) \cap \mathcal{DC}_{P_s}(\psi) & \\ \mathcal{DC}_{P_s}(\varphi\vee\psi) & \stackrel{\text{def}}{=} & \mathcal{DC}_{P_s}(\varphi) \cup \mathcal{DC}_{P_s}(\psi) & \end{array}$$

$\mathcal{DC}_{P_s}$ is defined for all $\mathcal{L}_S$-terms occurring as arguments of bridging functions in P because all the toplevel non-Boolean terms are in F_P. They are treated by the first part of the definition. The Boolean connectives are treated by the remaining parts.
The main decomposition function is now

$$\mathcal{DC}_P(\varphi) \stackrel{\text{def}}{=} \bigvee_{a\in\mathcal{DC}_{P_s}(\varphi)} a$$

Notice that $\mathcal{DC}_P(\varphi) = \bot$ if $\mathcal{DC}_{P_s}(\varphi) = \emptyset$. ■

The next lemma confirms that atomic decomposition of a $\mathcal{L}_S$-formula is an equivalence preserving transformation.

Lemma 3.4 (Equivalence Preserving) *For each $\mathcal{L}_S$-formula φ occurring in a problem specification $P = (\mathcal{P}, \mathcal{S}, \mathcal{T}_B)$ and for each model $\Im$ for P: $\varphi^\Im = \mathcal{DC}_P(\varphi)^\Im$.*

Proof: by structural induction.
Base Case: φ is a top-level non-Boolean $\mathcal{L}_S$-subformula of $\mathcal{P}$.

$$\begin{array}{rcll}
\varphi^\Im & = & \bigcup_{a \in A_{F_P}} (a^\Im \cap \varphi^\Im) & A_{F_P} \text{ defines a partitioning (Lemma 3.2)} \\
& = & \bigcup_{a \in A_{F_P}, a^\Im \cap \varphi^\Im \neq \emptyset} (a^\Im \cap \varphi^\Im) & \\
& = & (\bigvee_{a \in A_{F_P}, a \wedge \varphi \text{ is consistent with } \mathcal{S}} a \wedge \varphi)^\Im & \\
& = & (\bigvee_{a \in A_{F_P}, a = \ldots \wedge \varphi \wedge \ldots} a)^\Im & \text{(Def.3.1)} \\
& = & \mathcal{DC}_P(\varphi)^\Im & \text{(Def.3.3)}
\end{array}$$

The induction step is obvious. ■

Now we can apply the decomposition to $\mathcal{L}_{ES}$-formulae.

Definition 3.5 ($\mathcal{DC}_P$ applied to $\mathcal{L}_{ES}$-formulae) *We extend the definition of $\mathcal{DC}_P$ to $\mathcal{L}_{ES}$-formulae ψ as follows:*

If $f(\varphi_1, \ldots, \varphi_n, \vec{e})$ is a bridging terms for a bridging function f of argument type $n + k$ then

$$\mathcal{DC}_P(f(\psi_1, \ldots, \psi_n, \vec{e})) \stackrel{\text{def}}{=} f(\mathcal{DC}_P(\psi_1), \ldots, \mathcal{DC}_P(\psi_n), \vec{e})$$

$\mathcal{DC}_P(\psi)$ is the homomorphic extension of $\mathcal{DC}_P$ to the full $\mathcal{L}_{ES}$-language. ■

A corollary to Lemma 3.4 is the statement that $\mathcal{DC}_P(\psi)$ is an equivalence preserving transformation.

Corollary 3.6 (Equivalence Preserving) *$\mathcal{DC}_P(\psi)$ is an equivalence preserving transformation for the $\mathcal{L}_{ES}$-language* ■

3.2 Elimination of the Bridging Functions

Since the extensions of the atoms are disjoint, the disjointness conditions of the additivity axioms are fulfilled. In a second step we can therefore apply the additivity axioms as equivalence preserving rewrite rules to push the bridging functions down to the atomic level. We end up with a modified problem formulation where the arguments of the bridging functions are only atoms $a = \varphi_1 \wedge \ldots \wedge \varphi_n$. Let us call this the *normalized problem formulation.* In the theory given by the $\mathcal{S}$ and $\mathcal{T}_B$-part of a problem specification, the atomic decomposition and normalization is an equivalence preserving transformation.

Definition 3.7 (Pushing Down the Bridging Functions)
Let $P = (\mathcal{P}, \mathcal{S}, \mathcal{T}_B)$ be a problem specification. Let ψ be an $\mathcal{L}_{ES}$-formula and

$\psi' \overset{\text{def}}{=} \mathcal{DC}_P(\psi)$ *(Def. 3.5) We define a* push down *operation* $\mathcal{PD}_{\mathcal{A}_B}(\psi')$*. It uses the consequence parts of the additivity axioms* $\mathcal{A}_B$ *in* $\mathcal{T}_B$ *exhaustively as rewrite rules to push the bridging functions occurring in* ψ' *down to the atomic level.*

A 'pushed down' bridging function term has only atoms as $\mathcal{L}_S$*-arguments. We call these terms* atomic *bridging function terms.* ∎

The push down operation is well defined because the additivity axioms are required to represent a *confluent* rewrite rule system.

Example 3.8 (for the $\mathcal{PD}_{\mathcal{A}_B}$-Operation) *As an example, consider the formula*

$$\psi = f(a \vee b, c \wedge d) \geq 0.$$

Suppose $\mathcal{DC}_P(a \vee b) = a_1 \vee a_2$ *and* $\mathcal{DC}_P(c \wedge d) = b_1 \vee b_2 \vee b_3$*. Then* $\mathcal{DC}_P(f(a \vee b, c \wedge d)) = f(a_1 \vee a_2, b_1 \vee b_2 \vee b_3)$*. Suppose further,* f *is of distribution type* $+$ *in both arguments. Then*

$$\begin{array}{lll}
\psi & = & f(a \vee b, c \wedge d) \geq 0 \\
& \to & f(a_1 \vee a_2, b_1 \vee b_2 \vee b_3) \geq 0 \\
& \to & f(a_1 \vee a_2, b_1 \vee b_2 \vee b_3) \geq 0 \\
& \to & f(a_1, b_1 \vee b_2 \vee b_3) + f(a_2, b_1 \vee b_2 \vee b_3) \geq 0 \\
& \to & \ldots \to f(a_1, b_1) + f(a_1, b_2) + f(a_1, b_3) + f(a_2, b_1) + f(a_2, b_2) + f(a_2, b_3) \\
& = & \mathcal{PD}_{\mathcal{A}_B}(\mathcal{DC}_P(\psi))
\end{array}$$

∎

It is easy to prove that $\mathcal{PD}_{\mathcal{A}_B}$ is also an equivalence preserving transformation in the theory given by the $\mathcal{L}_S$-axioms $\mathcal{S}$ and the additivity axioms of the bridging functions.

Corollary 3.9 ($\mathcal{PD}_{\mathcal{A}_B}$ is equivalence preserving) *Let* $P = (\mathcal{P}, \mathcal{S}, \mathcal{T}_B)$ *be a problem specification. Then* $\psi^{\Im} = \mathcal{PD}_{\mathcal{A}_B}(\mathcal{DC}_P(\psi))^{\Im}$ *for all* $\mathcal{L}_{ES}$*-formulae* ψ *and and all models* $\Im$ *for* $\mathcal{S}$ *and* $\mathcal{T}_B$*.* ∎

This is a direct consequence of Lemma 3.4 and Corollary 3.6 and the fact that the atoms of the decomposition denote mutually disjoint sets.

3.3 Getting Rid of the Logic

The decomposition function $\mathcal{DC}_P$ and the push down function $\mathcal{PD}_{\mathcal{A}_B}$ pave the way for the final transformation which allows us to get rid of the set-theoretical parts of the problem specification and to generate pure $\mathcal{L}_E$ problems.

In this final transformation, bridging terms $f(\vec{a})$ are replaced with (existentially quantified) $\mathcal{L}_E$-variables $x_{f(\vec{a})}$. If they have $\mathcal{L}_E$-arguments as well a term $f(\vec{a}, \vec{e})$ is replaced with new free function terms $f'_{f(\vec{a})}(\vec{e})$ (here we need free function symbols in the $\mathcal{L}_E$-syntax).

More precisely:

Definition 3.10 (Elimination of Bridging Fcts.)
We define a replacement operation $\mathcal{RP}_P$ which replaces for a problem specification $P = (\mathcal{DC}_P(\mathcal{P}), \mathcal{S}, \mathcal{B})$ all atomic bridging function terms (Def. 3.7) with new $\mathcal{L}_E$-variables and terms:
Let $\vec{a}$ be a vector of atoms and $\perp$.
If f has argument type $n + 0$ then

$$\mathcal{RP}_P(f(\vec{a})) \stackrel{\text{def}}{=} x_{f(\vec{a})}$$

where $x_{f(\vec{a})}$ is new $\mathcal{L}_E$-variable.
If f has argument type $n + k$ then

$$\mathcal{RP}_P(f(\vec{a}, \vec{e})) \stackrel{\text{def}}{=} f'_{f(\vec{a})}(\vec{e})$$

where $f'_{f(\vec{a})}$ is a new free $\mathcal{L}_E$-function symbol.
$\mathcal{RP}_P(\varphi)$ is the homomorphic extension to normalized $\mathcal{L}_E$-formulae. ■

The problem formulation $\mathcal{P}$ is not the only part in a problem specification which has to be prepared for submission to the $\mathcal{L}_E$-solver. We also have to generate all *atomic instances* of the bridging function constraint axioms.

Definition 3.11 (Instantiation of Bridging Fct. Constraint Axioms)
Let A_{F_P} be the syntactic atoms (Def. 3.1) for a problem specification $P = (\mathcal{P}, \mathcal{S}, \mathcal{T}_B)$.

We define an instantiation function $\mathcal{I}_P(Ax[x_1, \ldots, x_n])$. It takes a bridging function constraint axiom $Ax[x_1, \ldots, x_n] \in \mathcal{C}_B$ with n free $\mathcal{L}_S$-variables as input and generates all its 'atomic instances', i.e.

$$\mathcal{I}_P(Ax[x_1, \ldots, x_n]) \stackrel{\text{def}}{=} \{Ax\{x_1 \mapsto a_1, \ldots x_n \mapsto a_n\} \mid (a_1, \ldots a_n) \in A^n_{F_P}\}$$

■

For example if the constraint axioms is (21)

$$I(x) \leq S(x)$$

and we have three atoms a, b, c then

$$\mathcal{I}_P(I(x) \leq S(x)) = \{I(a) \leq S(a), I(b) \leq S(b), I(c) \leq S(c)\}$$

With all these preparations we are now in the position to comprise everything into a procedure which prepares a problem specification for the $\mathcal{L}_E$-solver.

Definition 3.12 (Translation into $\mathcal{L}_E$-problems) *Let $P = (\mathcal{P}, \mathcal{S}, \mathcal{T}_B)$ be a problem specification where at least $\mathcal{S}$ and $\mathcal{T}_B = (\mathcal{A}_B, \mathcal{C}_B)$ are consistent.*

The function $\mathcal{T}r(P)$ translates the problem specification P into an $\mathcal{L}_E$-formula:

$$\mathcal{T}r(P) \stackrel{\text{def}}{=} \mathcal{RP}_P(\mathcal{PD}_{\mathcal{A}_B}(\mathcal{DC}_P(\mathcal{P})) \cup \mathcal{I}_P(\mathcal{C}_B))$$

■

The definition of the translation function $\mathcal{T}r$ is very compact. Therefore it might be instructive to split it into the different steps which have to be implemented:

Procedure 3.13 (Procedural Description of $\mathcal{T}r$)
Let $P = (\mathcal{P}, \mathcal{S}, \mathcal{T}_B)$ be the problem specification.

Step 1: *extract all propositional variables and toplevel non-Boolean subterms F_P from $\mathcal{P}$.*

Step 2: *compute the syntactic atoms A_{F_P} (Def. 3.1) (this may require exponentially many $\mathcal{L}_S$-consistency checks).*

Step 3: *compute the decomposition $\mathcal{P}' \stackrel{\text{def}}{=} \mathcal{DC}_P(\mathcal{P})$. (Def. 3.3)*

Step 4: *push the bridging functions down to the atomic level, i.e. $\mathcal{P}'' \stackrel{\text{def}}{=} \mathcal{PD}_{\mathcal{A}_B}(\mathcal{P}')$.*

Step 5: *instantiate the bridging function constraint axioms $\mathcal{C}_B$: ${\mathcal{C}_B}' \stackrel{\text{def}}{=} = \mathcal{I}_P(\mathcal{C}_B)$ (Def. 3.11).*

Step 6: *replace atomic bridging terms with $\mathcal{L}_E$-terms: $\mathcal{T}r(P) = \mathcal{RP}_P(\mathcal{P}'' \cup {\mathcal{C}_B}')$.* ■

At this state we can show that the translation function $\mathcal{T}r$ is *complete*. That means, if the problem specification P is satisfiable then $\mathcal{T}r(P)$ has a solution.

Theorem 3.14 (Completeness)
If a problem specification $P(\mathcal{P}, \mathcal{S}, \mathcal{B})$ is satisfiable then $\mathcal{T}r(P)$ has a solution.

Proof: Let $\Im$ be an $\mathcal{L}_{ES}$-interpretation satisfying P. $\mathcal{T}r$ is defined by

$$\mathcal{T}r(P) \stackrel{\text{def}}{=} \mathcal{RP}_P(\mathcal{PD}_{\mathcal{A}_B}(\mathcal{DC}_P(\mathcal{P})) \cup \mathcal{I}_P(\mathcal{C}_B))$$

(Def. 3.12). $\mathcal{DC}_P(\mathcal{P})$ is an equivalence transformation (Cor. 3.6). $\mathcal{PD}_{\mathcal{A}_B}(\mathcal{DC}_P(\mathcal{P}))$ is also an equivalence transformation (Cor. 3.9). Therefore $\mathcal{PD}_{\mathcal{A}_B}(\mathcal{DC}_P(\mathcal{P}))$ is satisfied by $\Im$. $\mathcal{I}_P(\mathcal{C}_B)$ is an instantiation of universally quantified variables and therefore also satisfied by $\Im$.

This means, each atomic bridging function term $f(\vec{a}, \vec{e})$ which occurs in $\mathcal{PD}_{\mathcal{A}_B}(\mathcal{DC}_P(\mathcal{P})) \cup \mathcal{I}_P(\mathcal{C}_B)$ is mapped by $\Im$ to a numeric value $(f(\vec{a}, \vec{e}))^{\Im}$ such that $\mathcal{PD}_{\mathcal{A}_B}(\mathcal{DC}_P(\mathcal{P})) \cup \mathcal{I}_P(\mathcal{C}_B)$ is satisfied.

We extend the interpretation $\Im$ by defining $(f'_{f(\vec{a})}(\vec{e}))^{\Im} \stackrel{\text{def}}{=} (f(\vec{a}, \vec{e}))^{\Im}$ for the new functions and variables generated by $\mathcal{RP}_P$. This way the new $\mathcal{L}_E$-terms in $\mathcal{T}r(P)$ are interpreted in exactly the same way as the atomic bridging terms in $\mathcal{PD}_{\mathcal{A}_B}(\mathcal{DC}_P(\mathcal{P})) \cup \mathcal{I}_P(\mathcal{C}_B)$. Therefore the truth value for each formula $\varphi \in \mathcal{PD}_{\mathcal{A}_B}(\mathcal{DC}_P(\mathcal{P})) \cup \mathcal{I}_P(\mathcal{C}_B)$ is the same as for $\mathcal{RP}_P(\varphi)$ (shown by structural induction). Since $\Im$ makes all these formulae φ true, the extended interpretation makes all $\mathcal{L}_E$-formulae $\mathcal{RP}_P(\varphi)$ true. Thus, $\mathcal{T}r(P)$ has a solution. ■

4 Post Mortem Analysis

Completeness of the translation $\mathcal{T}r$ means, we do not loose solutions by translating a mixed problem specification into a pure $\mathcal{L}_E$-problem. But it does not guarantee that all solutions for the $\mathcal{L}_E$-problem are also solutions for the initial problem (soundness). It turns out that in general this is indeed not the case. The $\mathcal{L}_E$-solver may in fact return wrong solutions. Therefore we propose a *post mortem* analysis to filter out the wrong solutions. This is quite expensive, but in Section 6 we shall see that for special set description languages the extent of the post mortem analysis can be considerably reduced.

The post mortem analysis consists of two different parts. The first part deals with the special role empty sets play in a set description language. If the $\mathcal{L}_E$-solver yields $x_{|a|} = 0$ this means the atom a is supposed to be false. It is like adding $\neg a$ to the problem formulation. This extra logical information may have implications for the consistency of other atoms, and this has to be checked.

The second part of the post mortem analysis deals with situations where the set description language itself can express constraints on the cardinality of sets. It may happen that for example the $\mathcal{L}_E$-solver proposes $x_{|a|} = n$, but there is no model for P where a is true for exactly n objects. We shall see that for a certain class of description logics, this is not the case, and therefore need not be checked. But in general it can happen, and needs to be checked.

4.1 Empty Sets

For most set description languages which are more expressive than Propositional Logic, the generated atoms are in fact not as independent as we would like them to be. For example in a description logic with an existential quantifier we have the constraint

$$|\varphi| = 0 \Rightarrow |\exists\, r.\varphi| = 0$$

(if a set φ is empty then there cannot be anything which has at least one r-role filler in φ). This formula needs to be turned into an arithmetic constraint of the kind

$$x_{|\varphi|} = 0 \Rightarrow x_{|\exists\, r.\varphi|} = 0$$

which restricts the solution space for the arithmetic solver.

In order to make sure that no constraints of this type are forgotten, we propose the following procedure:

1. In a first step we generate the arithmetic problem without such an 'emptyness constraint'.

2. When the arithmetic solver comes up with a solution where the variables $x_{|a_1|} = \ldots = x_{|a_j|} = 0$ and the other cardinality variables $x_{|a_{j+1}|}, \ldots, x_{|a_m|}$ have values different from 0, we add $\neg a_1 \wedge \ldots \wedge \neg a_j$ to the set axioms and test $a_{j+1}, \ldots, a_m$ for consistency with the extended set axioms.

3. For each atom a which is inconsistent with the extended set axioms we add a constraint $x_{|a_1|} = 0 \wedge \ldots \wedge x_{|a_j|} = 0 \Rightarrow x_{|a|} = 0$ to the arithmetic formulation of the problem.

4. The arithmetic solver is restarted with the new arithmetic formulation.

5. The procedure is iterated until no new emptyness constraints are discovered.

Since there are only finitely many arithmetic variables, there are only finitely many emptyness constraints. Therefore the procedure terminates eventually.

More formally:

Procedure 4.1 (*$EC(P)$* for 'Emptyness Constraints')
Let $P = (\mathcal{P}, \mathcal{S}, \mathcal{A}_B)$ be a problem specification

The procedure $EC(P)$ for dealing with the emptyness constraints works as follows:

Let $A_0 = \mathcal{T}r(P)$ be the generated $\mathcal{L}_E$-problem formulation.

Let $k = 0$, $E = \emptyset$.

Until there are no new empty atoms E:

1. *Let σ_k be the solution of A_k (as computed by the $\mathcal{L}_E$-solver). If there is no such solution, return 'unsolvable'.*

2. *Let $x_{|a_1|} = 0 \wedge \ldots \wedge x_{|a_j|} = 0$ be the part of σ_i with zero solutions of the cardinality variables. Let $x_{|a_{j+1}|}, \ldots, x_{|a_m|}$ be the other variables.*

3. *Let $E \overset{\text{def}}{=} \{a_i \mid j < i \leq m, \{\mathcal{S}, \neg a_1, \ldots, \neg a_j, a_i\}$ is inconsistent$\}$ be the set of new empty atoms.*

4. *Let $A_{k+1} = A_k \cup \{x_{|a_1|} = 0 \wedge \ldots \wedge x_{|a_j|} = 0 \Rightarrow x_{|a|} = 0 \mid a \in E\}$*

5. *$k = k + 1$*

Return σ_k.

$EC(P)$ just returns a single solution candidate for P. It is trivial to change the implementation of $EC(P)$ such that it enumerates all (candidate) solutions for P, provided the underlying $\mathcal{L}_E$-solver can enumerate all solutions.

Therefore let ECM be a variant of EC which enumerates all candidate solutions for P. ■

4.2 General Cardinality Constraints

Even without explicit cardinality operators some logics can impose restrictions on the size of a set. A relatively simple example is the formula

$$\Diamond p \wedge \Diamond \neg p$$

from modal logic KTB. KTB has a reflexive and symmetric accessibility relation. (In Description Logic syntax this formula would be $\exists\, r.p \wedge \exists\, r.\neg p$). Because of the p and $\neg p$ a formula for this model must have at least two worlds. A little model looks like this:

$$\begin{tikzcd} p \arrow[loop below] \arrow[rr, leftrightarrow] & & \neg p \arrow[loop below] \end{tikzcd}$$

In both worlds $\Diamond p \wedge \Diamond\neg p$ holds. It turns out that, due to reflexivity and symmetry of the accessibility relation, there is no possibility to construct a model where $\Diamond p \wedge \Diamond\neg p$ holds in only one world. Thus, $|\Diamond p \wedge \Diamond\neg p| \geq 2$ is a constraint coming from the logic itself.

Not all set description languages can enforce constraints like this. Therefore we introduce some new notions which characterize useful features of set description languages and help reducing the extent of the post mortem analysis.

The first notion, the 'singleton set model' property is a variant of the finite model property. A logic has the finite model property iff for each consistent formula φ there is a model $\Im$ for φ *with finite domain.* It has *the singleton set model* property iff $|\varphi^{\Im}| = 1$ (the domain need not be finite).

Definition 4.2 ('singleton set model' Property)
A Set Description Language has the 'singleton set model' *property iff whenever a set description language formula φ is consistent with axioms $\mathcal{S}$ then there is also an interpretation $\Im$ satisfying $\mathcal{S}$ where $\varphi^{\Im}$ is a singleton set.* ■

From solutions $x_{|\varphi|} = n$ produced by the $\mathcal{L}_E$-solver we want to build up models $\Im$ with $|\varphi^{\Im}| = n$. If the set description language has the singleton set model property, we get at least a model $\Im'$ with $|\varphi^{\Im'}| = 1$. If it is possible to join n disjoint copies of $\Im'$, we could achieve $|\varphi^{\Im}| = n$. Unfortunately it is not always possible to join disjoint models without changing the truth value of formulae. In particular in description logics with role complement, even unrelated objects in the model affect the truth value of a formula. Therefore we define the 'joinable models' property.

Definition 4.3 ('joinable models' Property)
A set description language $\mathcal{L}_S$ has the 'joinable models' *property iff for two interpretations $\Im_1$ and $\Im_2$* with disjoint domains *both satisfying $\mathcal{S}$ the disjoint union of $\Im_1$ and $\Im_2$ also satisfies $\mathcal{S}$ and the same formulae as $\Im_1$ and $\Im_2$ separately.*[10] ■

Set description languages with singleton set model property and joinable models property are harmless for our purposes. They don't require any further post mortem analysis.

Set description languages which have the joinable models property, but not the singleton set model property require an extra test.

[10]The disjoint union of two disjoint interpretations $\Im_1$ and $\Im_2$ is obtained by taking the union of the two domains as the new domain, and by taking the union $a^{\Im_1} \cup a^{\Im_2}$ as the new interpretation of a signature element a.

Definition 4.4 (Single Cardinality Test *SCT*) *A* single cardinality test *$SCT(\mathcal{S}, \varphi)$ for a set description language is a decision procedure that takes as input a set $\mathcal{S}$ of set axioms, a formula φ which is consistent with $\mathcal{S}$. It returns 'true' if there is a model $\Im$ for $\mathcal{S}$ and φ such that $|\varphi^{\Im}| = 1$.* ■

A set description language without singleton set model property and joinable models property requires a full post mortem analysis of the generated $\mathcal{L}_E$-solution. For this purpose we need a *multiple cardinality test*

Definition 4.5 (Multiple Cardinality Test *MCT*) *A* multiple cardinality test *$MCT(S, \varphi_1, n_1, \ldots, \varphi_k, n_k)$ for a set description language is a decision procedure that takes as input a set $\mathcal{S}$ of set axioms, formulae $\varphi_1, \ldots, \varphi_k$ each of which are consistent with $\mathcal{S}$, but pairwise inconsistent and k positive integers n_i. It returns 'true' if there is a model $\Im$ for $\mathcal{S}$ and $\varphi_1, \ldots, \varphi_k$ such that $|\varphi_i^{\Im}| = n_i$ for $1 \leq i \leq k$.* ■

The *mixed set problem solver* defined below is a very general method for solving problem specifications.

Procedure 4.6 (Mixed Set Problem Solver) *The* mixed set problem solver *$MSP(P)$ for a problem specification $P = (\mathcal{P}, \mathcal{S}, \mathcal{B})$ now works as follows:*

1. *If the set description language has the singleton set model property (Def. 4.2) and the joinable models property (Def. 4.3) then return $EC(P)$ (Def. 4.1).*

2. *If the set description language has the joinable models property (Def. 4.3) and a single cardinality test SCT (Def. 4.4), and if $SCT(\mathcal{S}, a)$ holds for all atoms a then return $EC(P)$.*

3. *If the set description language has the joinable models property and a multiple cardinality test MCT (Def. 4.5) then enumerate the solutions with $ECM(P)$ (Def. 4.1). For each solution σ in the enumeration:*

   (a) *Let $x_{|a_1|} = 0, \ldots, x_{|a_k|} = 0$ be the part in σ requiring the a_i to be empty.*

   (b) *For each assignment $x_{|a|} = n$ in σ with $n > 0$ and where the single cardinality test $SCT(\mathcal{S}, a)$ failed check $MCT(\mathcal{S} \wedge \neg a_1 \wedge \ldots \wedge \neg a_k, a, n)$.*

   If this test always returns 'true' then return σ, otherwise reject σ and get the next solution in the enumeration.

4. *If the set description language has a multiple cardinality test MCT (Def. 4.5) then enumerate the solutions with $ECM(P)$. For each solution σ in the enumeration:*

   (a) *Let $x_{|a_1|} = 0, \ldots, x_{|a_k|} = 0$ be the part in σ requiring the a_i to be empty. Let $x_{|a_{j+1}|} = n_{j+1}, \ldots, x_{|a_m|} = n_m$ be the assignments for the other 'cardinality variables'.*

(b) *Check* $MCT(\mathcal{S} \wedge \neg a_1 \wedge \ldots \wedge \neg a_k, a_{j+1}, n_{j+1}, \ldots, a_m, n_m)$.

If this test returns 'true' then return σ*, otherwise reject* σ *and get the next solution in the enumeration.*

5. *otherwise* P *cannot be solved with the decomposition method.*

Theorem 4.7 (Soundness of the Mixed Problem Solver)
If the mixed problem solver $MSP(P)$ *(Def. 4.6) returns a solution* σ *for the problem specification* $P = (\mathcal{P}, \mathcal{S}, \mathcal{A}_B)$ *then there is a model* $\Im$ *for* P *such that* $(f(\vec{a}, \vec{e}))^{\Im} = f'_{f(\vec{a})}(\vec{e})\sigma$ *for all atomic bridging functions, provided the additivity axioms for the bridging functions are partitionable (Def. 2.9) and the constraint axioms are partitioning consistent (Def. 2.16).*

Proof: In a first step we construct the set part of the interpretation $\Im$. In the second step we treat the other bridging functions.

Let $x_{|a_1|} = 0, \ldots, x_{|a_k|} = 0$ be the part in σ requiring the a_i to be empty. Let $x_{|a_{j+1}|} = n_{j+1}, \ldots, x_{|a_m|} = n_m$ be the assignments for the other 'cardinality variables'.

The emptyness constraint procedure EC (Def. 4.1) guarantees that $\mathcal{S}$, $\neg a_1$, …, $\neg a_k$ is consistent with each of the a_i for $j > i \leq m$. Therefore there are models $\Im_i$ satisfying $\mathcal{S}, \neg a_1, \ldots, \neg a_k, a_i$ for $j > i \leq m$.

The Set-Part of the Interpretation

The construction of the set part of the interpretation $\Im$ is different for each of the three cases in MSP.

Case 1: The set description language has the singleton set model property and the joinable models property.
Because of the singleton set model property, we can assume that $|a_i^{\Im_i}| = 1$ for $j > i \leq m$. Since the a_i are pairwise inconsistent, we can further assume that the domains of the $\Im_i$ are pairwise disjoint. Because of the joinable models property, we can take n_i disjoint copies of $\Im_i$ and get models $\Im'_i$ satisfying $\mathcal{S}, \neg a_1, \ldots, \neg a_k$ such that $|a_i^{\Im_i}| = n_i$ for $j > i \leq m$. The domains of the $\Im'_i$ are also pairwise disjoint. Therefore we can exploit the joinable models property again and join all these models together to get a single interpretation $\Im$ satisfying $\mathcal{S}, \neg a_1, \ldots, \neg a_k$ such that $|a_i^{\Im}| = n_i$ for $j > i \leq m$.

Case 2: The set description language has the joinable models property and a single cardinality test SCT (Def. 4.4). and $SCT(\varphi, a)$ succeeded for all atoms a. This is the same situation as in Case 1.

Case 3: The set description language has the joinable models property and a multiple cardinality test MCT (Def. 4.5). The multiple cardinality test guarantees that there are models $\Im'_a$ satisfying $\mathcal{S}$, $\neg a_1$, …, $\neg a_k$ such that $|a^{\Im_a}| = n$ for the atoms a for which the single cardinality test failed. For the atoms for which it succeeds, we can get a joined model with $|a^{\Im_a}| = n$ anyway. Since the a_i are pairwise inconsistent, we can further assume that the domains of the $\Im'_i$ are pairwise disjoint. In the same way as before we can exploit the joinable models property to join all these models together to get a single interpretation $\Im$ satisfying $\mathcal{S}, \neg a_1, \ldots, \neg a_k$ such that $|a_i^{\Im}| = n_i$ for $j > i \leq m$.

Case 4: The set description language has multiple cardinality test MCT (Def. 4.5)
The multiple cardinality test guarantees that there is a model $\Im$ satisfying $\mathcal{S}, \neg a_1, \ldots, \neg a_k$ such that $|a_i^{\Im}| = n_i$ for $j > i \leq m$.

The Bridging Function Part of the Interpretation

The interpretation $\Im$ constructed in the previous step satisfies the set part of the problem specification and it guarantees that the cardinality constraints for the atoms which are required by the solution σ are met.

Now we have to extend $\Im$ by suitable meanings for the other bridging functions such that the other parts $f'_{f(\vec{a},\vec{e})}(\vec{e}) = n_{f,a,e}$ in σ are met.

First we define $f^{\Im}(\vec{a}^{\Im}, \vec{e}^{\Im}) \overset{\text{def}}{=} n_{f,a,e}$ for each $f'_{f(\vec{a})}(\vec{e}) = n_{f,a,e}$ in σ. For combinations of the atoms $\vec{a}$, for which no constraint is required by σ, we choose an arbitrary $n_{f,a,e}$. Since the additivity axioms are partitionable (Def. 2.9), and since the emptyness axioms (Def. 2.8) are in the bridging function constraint axioms, there is an interpolation model for $\Im$ satisfying all the additivity axioms. Furthermore, since the bridging function constraint axioms are partitioning consistent, (Def. 2.16), we know that the interpolation model satisfies these axioms as well.

Thus, the so constructed interpretation satisfies the bridging function theory $\mathcal{T}_B$ and the decomposed problem formulation $\mathcal{PD}_{\mathcal{A}_B}(\mathcal{DC}_P(\mathcal{P}))$. Since $\mathcal{DC}_P$ and $\mathcal{PD}_{\mathcal{A}_B}$ are equivalence transformations (Cor. 3.6 and 3.9), it satisfies P itself. Moreover, it agrees with the solution σ on the interpretation of the atomic bridging terms. ■

This finally proves that the decomposition method is a sound and complete way for solving mixed logical-arithmetical problems. If the $\mathcal{L}_S$-consistency test and the $\mathcal{L}_E$-solver always terminate and the set description language has the singleton set model property and the joinable models property then the mixed problem solver terminates as well. Unfortunately this need not be the case if the single cardinality test or the multiple cardinality test has to be called. In this case the $\mathcal{L}_E$-solver may keep producing solutions which again and again get rejected. This need not terminate.

Since there may be exponentially many atoms, there is an exponential step in the mixed problem solver which is inherent in the decomposition technique. Therefore the time complexity cannot be better than EXPTIME, even if the $\mathcal{L}_S$ and $\mathcal{L}_E$-problems can be solved faster.

5 Optimizations

The decomposition method has three main bottlenecks:

1. the number of atoms may be exponential in the number of non-Boolean subfomulae occurring in the problem formulation;

2. the computation of the atoms involves a consistency check. Therefore there may be exponentially many consistency checks;
3. the post mortem analysis may generate a possibly nonterminating generate and test loop.

In principle, one cannot get around these problems. There may be problem specifications where all this cannot be avoided. But with a careful design of the axiomatization and a careful implementation of the whole procedure, one can do a lot to improve its behaviour. We sketch some possibilities, but further research is necessary to find really good optimizations.

The Exponential Number of Atoms

Each atom corresponds to a possibly nonempty intersection of sets which are relevant for the problem. If it is known right from the beginning that some intersections must be empty, then one can avoid generating the 'intersection atoms' at all. Suppose, for example, the set axioms specify an ontology about, say, cars and people. If there are n subconcepts for cars and m subconcepts for people then you may get the order of $2^n \cdot 2^m$ 'intersection atoms', all denoting objects which are cars and people at the same time, which is nonsense. If, however, you give the system the information that cars and people are disjoint then you get at most $2^n + 2^m$ atoms. This is an exponential improvement. Thus, the more information the system gets, the less atoms are generated.

The Consistency Test for the Atoms

The recursive definition of the procedure for computing the atoms (Def. 3.1) offers the possibility for employing heuristics to improve the computation. This procedure depends on the ordering of the non-Boolean subformulae F_P to be integrated into the atoms. If the algorithm first checks ψ_1 for consistency of $X \wedge \psi_1$, and finds that this is inconsistent, then for a second formula ψ_2, consistency of $X \wedge \psi \wedge \psi_2$ will never be checked. Therefore it is important to choose those formulae first, which are most likely to produce inconsistencies.

Information from *classified* concept hierarchies and information about disjointness of concepts may help choosing a good ordering. But it may also help avoiding consistency tests altogether. Suppose $\psi \wedge \varphi$ is to be tested for consistency. If a classified concept hierarchy contains the information $\psi \Rightarrow a$ and $\varphi \Rightarrow b$, and a and b are disjoint, then $\psi \wedge \varphi$ must be inconsistent. No further check is needed. Another possibility for avoiding consistency checks is: if the classified concept hierarchy says $\varphi \Rightarrow \psi$, then $\varphi \wedge \neg\psi$ are inconsistent and need not be checked.

A good algorithm for computing the atoms would make optimal use of the information about concept hierarchies and disjointness of concepts. Such an algorithm is certainly worth an extra paper.

The Post Mortem Analysis

The post mortem analysis is not necessary if the set description language is just Propositional Logic. Therefore the best way to avoid it is to use Propositional Logic whenever it is possible (cf. the example in Section 1.7). If it cannot be avoided in general, one can try to reduce it.

The first phase of the post mortem analysis deals with empty sets. It takes care of correlations in the set description language of the form 'if these sets are empty then those sets must be empty as well'. The constraint that a set φ is empty can be expressed logically by adding $\neg\varphi$ to the set axioms. Therefore empty sets need a special treatment. The optimal way for dealing with empty sets would be by submitting sufficient constraints of the form $x_{|a_1|} = 0 \wedge \ldots \wedge x_{|a_n|} = 0 \Rightarrow x_{|a|} = 0$ to the $\mathcal{L}_E$-solver. Whether this is possible depends on the set description language. It requires a careful analysis of the impact of false subformulae to the truth value of other formulae. In most description logics, for example you have the correlation $|\varphi| = 0 \Rightarrow |\exists\, r.\varphi| = 0$ which can be turned into such a constraint. But whether this is sufficient or not is not clear.

The general procedure $EC(P)$ (Def. 4.1) that deals with such 'emptyness constraints' is independent of the set description language, and therefore very likely not optimal. One possibility to improve it is to integrate it into the $\mathcal{L}_E$-solver (if the implementation of the solver is flexible enough to enable this): as soon as the solver generates $x_{|a_l|} = 0$ as partial solution for some variables $x_{|a_l|}$, one can check whether this entails $x_{|a|} = 0$ for some other variable $x_{|a|}$ and then keep this as an extra constraint. This prevents the solver from generating unnecessary solutions. On the other hand, in a later state of the search, the solver itself might reject the $x_{|a_l|} = 0$ part of the solution, and therefore the expensive consistency test might never be necessary. Some heuristic guidance is needed at this point.

The second part of the post mortem analysis deals with more complicated constraints on the cardinality of sets imposed by the set description language itself. It is a generate and test method. The $\mathcal{L}_E$-solver generates a solution candidate and suitable $\mathcal{L}_S$-tests check whether this solution is acceptable. This test is not necessary if the set description language has the singleton set model property and the joinable model property. Therefore assuring this is the best way to avoid the expensive generate and test loop. If the set description language has the joinable model property, but lacks the singleton set model property, one might possibly be able to extract constraints on $|a^{\Im}|$, which may help the $\mathcal{L}_E$-solver to avoid generating incompatible solutions. This is an area where further research on the ability of set description languages to express constraints on the size of interpretations is necessary.

6 Description Logics as SDLs

A general theory is not worth much if you cannot apply it to concrete examples. Therefore we investigate in this section how one can make use of the general method for combining Description Logic with arithmetic reasoning. Description logics themselves have been briefly introduced in Section 2.2. Since they come in so many variations, we can use them for illustrating all aspects of the general combination methodology, and for illustrating the kind of theoretical investigations necessary to apply the general methodology.

One of the most inefficient parts in the decomposition method is the post mortem analysis (Section 4). This has to be avoided at all costs. Therefore it is extremely important to check the singleton set model property and the joinable models property. It turns out that there are in fact description logics without these properties, but there are also 'harmless' description logics where these properties hold. The main contribution of this section is therefore a characterization of the description logics with or without these properties.

Both properties are closely related with well known properties of modal logics, the generated submodel property and the tree model property. A description logic with the generated submodel property has the joinable models property. A description logic which also has the *finite* tree model property has the singleton set model property.

Before starting characterizing description logics with the generated submodel property we need two definitions. The first definition just states what is assumed anyway for reasonable description logics.

Definition 6.1 (Kripke Style Semantics)
A description logic admits a Kripke style semantics iff

1. *each interpretation can be separated into a* frame component *and a* valuation component*;*

2. *the frame component consists of a set of objects (worlds, the* domain*) and a set of binary relations (accessibility relations) between the objects. Each 'role name' is mapped to a binary relation;*

3. *the valuation maps concept names to subsets of the domain.* ■

Only description logics with admitting Kripke style semantics are considered here.

The next definition introduces a useful notion, the *accessible part* of a frame.

Definition 6.2 (Accessible Part of Relations) *Let $R = r_1, \ldots, r_n$ be a finite set of binary relations on some set D. For each element $x \in D$ let*

$$ap_R(x) \overset{\text{def}}{=} \{y \mid xs_1; \ldots; s_k y, (s_1, \ldots, s_k) \subseteq R^n \text{ for some } n\}$$

be the subset of D which is accessible from x via finitely many (including 0) R-transitions.

$ap_R(x)$ is the R-accessible part from x in the set D. ■

Description logics are sometimes distinguished by the kind of TBoxes they allow. A restricted version of TBoxes, the *definitional TBoxes*, consists only of cycle free equations $c = \varphi$ where c is a concept name. *General TBoxes* allow arbitrary axioms $\psi \Rightarrow \varphi$. The advantage of definitional TBoxes is that one can use them as rewrite rules once and then forget them completely. This can make a big difference for algorithms as well as for theoretical properties of the logic. As we shall see, general TBoxes can destroy the singleton set model property.

6.1 The Generated Submodel Property

A description logic has the generated submodel property if a formula which is true for a particular object (world) w in a frame F is also true in the subframe consisting of the accessible part from w. We show easy ways of recognizing whether a description logic has this property.

Definition 6.3 (Generated Submodel Property (GSM))
A description logic which admits Kripke style semantics has the generated submodel property *iff each satisfiable formula φ which is satisfiable at some domain element w of a frame F, is also satisfiable in a subframe F' of F consisting of the accessible part $ap_R(w)$ (Def. 6.2) of w. R is the set of binary relations consisting of the interpretations of all the role names occurring in φ.* ■

Description logics which do not have this property usually have operators allowing them to 'jump around' arbitrarily in a frame. Crucial for this is the negation of roles.

Proposition 6.4 (Counter Indication for GSM) *If the description logic has the top role or role negation in its syntax and a way to express existential quantification over role fillers then it does* not *have the generated submodel property.*

Proof: Consider the formulae $p \wedge \exists\,(\neg r).\neg p$ or $p \wedge \exists\,\top.\neg p$. Both are satisfied in w_1 of the model

$$w_1 : p \qquad\qquad w_2 : \neg p$$

The generated submodel, however, consists of w_1 only, but this fails to satisfy the second part of the conjunctions $\exists\,\neg r.\neg p$ and $\exists\,\top.\neg p$. ■

Proposition 6.5 (Another Counter Indication for GSM) *If the description logic has the converse role operator $^{-1}$ in its syntax and a way to express existential quantification over role fillers then it does* not *have the generated submodel property.*

Proof: Consider the formula $p \wedge \exists\, r^{-1}.\neg p$. It is satisfied in w_2 of the model

$$\begin{tikzcd} w_1 : \neg p \arrow[r, "r"] & w_2 : p \end{tikzcd}$$

The generated submodel, however, consists of w_2 only, but this fails to satisfy the second part of the conjunction. ■

One could change the definition of accessible part of the frame by allowing to move 'backwards' along a relation. Then the converse role operator would not destroy the generated submodel property. But this notion of 'accessible part' would not allow us to prove the singleton set model property.

The generated submodel property has something to do with the fact that the operators can only 'look forward' along the accessibility relations, and not jump to other parts of the frame. This can in fact be characterized precisely. Moreover there are easy checks which depend only on the semantics of the role operators and the logical operators. That means one can check each operator once and for all, and then use them in any combination.

First of all we define *directed role operators*. A role operator is directed if the new relation defined by the operator only 'moves forward', i.e. it is a subrelation of the accessible part of the relations it combines.

Definition 6.6 (Directed Role Operator)
An n-place role operator $op(r_1, \ldots, r_n)$ is directed *iff for all interpretations $\Im$ and all domain elements x: $(op(r_1, \ldots, r_n))^{\Im_x} \subseteq ap_{\{r_1^{\Im}, \ldots, r_n^{\Im}\}}(x)$. (The r_i are just role names, no role terms).* ∎

Notice that this definition only depends on the role operator itself, and not on the logic which uses this operator.

Proposition 6.7 (Directed Role Operators) *The role operators $\wedge, \vee, ;, *, 1$ are directed whereas $^{-1}, \neg, \top$ are not directed.*

Proof:

- $(r_1 \wedge r_2)^{\Im_x} = r_1^{\Im_x} \cap r_2^{\Im_x} \subseteq ap_{r_1^{\Im}, r_2^{\Im}}(x)$.
- $(r_1 \vee r_2)^{\Im_x} = r_1^{\Im_x} \cup r_2^{\Im_x} \subseteq ap_{r_1^{\Im}, r_2^{\Im}}(x)$.
- $(r_1; r_2)^{\Im_x} \subseteq ap_{r_1^{\Im}, r_2^{\Im}}(x)$.
- $(r^*)^{\Im_x} = (r; \ldots; r)^{\Im_x} \subseteq ap_{r^{\Im}}(x)$.
- $(1)^{\Im_x} = \{x\} \subseteq ap_{1^{\Im}}(x)$.
- Counter example for $^{-1}$:
  Let $r = \{(a, b)\}$. Then $(r^{-1})^{\Im_b} = \{a\}$, whereas $ap_{r^{\Im}}(b) = \emptyset$
- Counter example for $\neg$: Let $r = \emptyset$ and the domain $D = \{a, b\}$.
  Then $(\neg r)^{\Im_a} = \{b\}$, whereas $ap_{r^{\Im}}(a) = \emptyset$.

∎

Directed role operators create only new relations which 'stay within' the R-accessible parts, i.e.

Lemma 6.8 *Let R be a set of relations and r a relation created by applying directed role operators to the relations in R. Then*

$$ap_R(w) = ap_{R \cup r}(w)$$

for all w. ∎

The proof is by induction on the construction of r.

A similar property can be defined for concept operators.

Definition 6.9 (Directed Concept Operators)
A concept operator $O(\vec{r}, \vec{c})$ is directed *with respect to some set R of role names including $\vec{r}$ iff*

1. *it does not bind variables (like in hybrid logics) and*
2. *its semantics is such that*
   for every tuple $\vec{r}$ of role names *and every tuple $\vec{c}$ of* concept names *every interpretation $\Im$ and every domain element w: if $\Im, w \models (O(\vec{r}, \vec{c}))$ then $\Im', w \models (O(\vec{r}, \vec{c}))$ where $\Im'$ is $\Im$ restricted to the accessible part $ap_R(w)$ of w.* ■

It is important that these tests can be done with concept names and role names only. This makes them independent of other operators.

Proposition 6.10 (Directed Concept Operators)
The concept operators $\wedge, \vee, \neg, \forall, \exists, \mathrm{atleast}, \mathrm{atmost}, \mathrm{more}$ *are directed, whereas the S5-operator* somewhere *is not directed.*

Proof: Let p, q be concept names and r and s role names. Let $\{r, s\} \subseteq R$

- $\Im, w \models p \wedge q$ iff $\Im', w \models p \wedge q$ where the domain of $\Im'$ contains at least w. The same holds for the other Boolean operators.
- $\Im, w \models \forall\, r.p$ iff $\Im, v \models p$ for all r-accessible objects from w iff $\Im', w \models \forall\, r.p$ where $\Im$'s domain consists of w and its r-accessible objects, and this is a subset of $ap_R(w)$. A similar argument holds for the $\exists$-operator.
- $\Im, w \models \mathit{atleast}\, n\, r.p$ iff $\Im, v \models p$ for at least n of the r-accessible objects from w iff $\Im', w \models \mathit{atleast}\, n\, r.p$ where $\Im$'s domain consists of w and its r-accessible objects, and this is a subset of $ap_R(w)$. A similar argument holds for the *atmost*-operator.
- $\Im, w \models \mathit{more}\, r\ p\ s\ q$ iff $|r^{\Im_w} \cap p^{\Im}| \geq |s^{\Im_w} \cap s^{\Im}|$ iff $\Im', w \models \mathit{more}\, r\ p\ s\ q$ where $\Im$'s domain consists of w and its $r \cup s$-accessible objects, and this is a subset of $ap_R(w)$.

The *somewhere* operator is not directed. Suppose $\Im, w_1 \models \mathit{somewhere}\, p$ where $\Im$ looks like

$$w_1 : \neg p \qquad\qquad w_2 : p$$

i.e. w_2 is not accessible from w_1. Deleting w_2 makes $\mathit{somewhere}\, p$ false at w_1. ■

Theorem 6.11 (Generated Submodel Property) *A description logic whose role and concept operators are directed has the generated submodel property.*

Proof: Let R be the set of role names. Suppose an interpretation $\Im$ satisfies a formula φ in a domain element w, i.e. $\Im, w \models \varphi$. Let $\Im'$ be the generated submodel with domain $ap_R(w)$. We show by structural induction $\psi^{\Im'} \subseteq \psi^{\Im}$ for all formulae ψ. This implies $\Im', w \models \varphi$.

The cases where ψ is a concept name (base case) or a formula with a Boolean toplevel connective (some of the cases in the induction step) are straightforward.

Now suppose $\psi = O(s_1, \ldots, s_n, \psi_1, \ldots, \psi_m)$ for some role terms s_i and formulae ψ_j. The induction hypothesis is $\psi_j^{\Im'} \subseteq \psi_j^{\Im}$ for $1 \leq j \leq m$.

We introduce new role names $r_1 \ldots r_n$ and concept names $c_1, \ldots, c_m$. The interpretation $\Im$ is extended to $\Im^+$ by defining $r_i^{\Im^+} \stackrel{\text{def}}{=} s_i^{\Im}$ for $1 \leq i \leq n$ and $c_j^{\Im^+} \stackrel{\text{def}}{=} \psi_j^{\Im}$ for $1 \leq j \leq m$.

Obviously we have $\psi^{\Im} = O(r_1, \ldots, r_n, c_1, \ldots, c_m)^{\Im^+}$.

Since all role operators are directed, we further have $ap_R(w) = ap_{R'}(w)$ where $R' = R \cup \{r_1, \ldots, r_n\}$. (Lem. 6.8). Therefore we can define $\Im^{+'}$ as the generated submodel of $\Im^+$ with domain $ap_R(w)$. $\Im^{+'}$ agrees with $\Im'$ on $ap_R(w)$. Furthermore we have $r_i^{\Im^{+'}} = s_i^{\Im'}$ for $1 \leq i \leq n$ and $c_j^{\Im^{+'}} = \psi_j^{\Im'}$ for $1 \leq j \leq m$.

Since all concept operators are directed, we have $O(r_1, \ldots, r_n, c_1, \ldots, c_m)^{\Im^{+'}} \subseteq O(r_1, \ldots, r_n, c_1, \ldots, c_m)^{\Im^+} = \psi^{\Im}$. Thus, $O(r_1, \ldots, r_n, c_1, \ldots, c_m)^{\Im^{+'}} \subseteq \psi^{\Im}$, and by the definition of $\Im^{+'}$, $\psi^{\Im'} \subseteq \psi^{\Im}$. ■

This theorem gives us an easy and intuitive check for the generated submodel property: test whether all role and concept operators are directed.

It is now easy to prove that logics with the generated submodel property also have the joinable models property.

Proposition 6.12 (JMP and GSMP) *Every description logic with the generated submodels property (Def. 6.3) has the joinable models property (Def. 4.3).*

Proof: Suppose $\Im_1$ and $\Im_2$ are two disjoint interpretations and $\Im$ is the disjoint union of $\Im_1$ and $\Im_2$. suppose $\Im_1, w \models \varphi$ and $\Im, w \models \neg\varphi$. Let $\Im_1'$ be the submodel of $\Im_1$ generated from w and $\Im'$ be the submodel of $\Im$ generated from w and Because of the generated submodel property $\Im_1', w \models \varphi$ and $\Im', w \models \neg\varphi$, but this is impossible because both submodels are identical. Therefore $\Im_1$ and $\Im$ interpret all formulae the same way. ■

The generated submodel property is sufficient, but probably not necessary for the joinable models property. It is, for example, very likely that the role converse operator $^{-1}$ does not harm the joinable models property. Therefore it might be necessary to check this property for particular logics which don't have the generated submodel property as defined here.

6.2 The Finite Tree Model Property

The generated submodel property is not sufficient for ensuring the single set model property. We also need the finite tree model property.

Definition 6.13 (Finite Tree Model Property (FTMP)) *Let T be a TBox.*

1. *A description logic has the* finite model property *iff each formula φ which is consistent with T is satisfiable in a model $\Im$ of T with finite domain.*
2. *It has the* finite tree model property *iff in addition $\Im$'s frame is a finite tree with possibly reflexive nodes*[11] *and φ is satisfied in the root of the tree.*

■

Proposition 6.14 (FTMP and General TBoxes) *A description logic with general TBoxes and an existential quantifier does* not *have the finite tree model property.*

Proof: As a counter example, consider the TBox

$$\begin{aligned} \mathit{male} &\Rightarrow \exists\, \mathit{has\text{-}spouse}.\neg \mathit{male} \\ \neg \mathit{male} &\Rightarrow \exists\, \mathit{has\text{-}spouse}.\mathit{male} \end{aligned}$$

The smallest model satisfying this TBox is

$$\begin{tikzcd} \mathit{male} \arrow[r, leftrightarrow, "\mathit{has\text{-}spouse}"] & \neg \mathit{male} \end{tikzcd}$$

'Unrolling' this model either yields an infinite chain, or a copy of this symmetric part of the *has-spouse* relation. Neither of it is a finite tree model. ■

The finite tree model property depends on the structure of the TBox and even more on the properties of the accessibility relations. Therefore a characterization of this property in terms of features of the operators is not possible. Instead we get an easy and very useful characterization by reducing it to properties of algorithms which generate tree models for formulae. These are the tableaux decision algorithms. They are well known for man description logic, and it is easy to check whether they generate finite tree models or not.

Proposition 6.15 (FTMP) *A description logic without general TBoxes which*

1. *has a sound, complete and terminating tableaux procedure*
2. *which does not need to reuse previously introduced worlds and*
3. *whose canonical model construction does not need to generate infinite chains of worlds and cycles in the accessibility relations, except possibly reflexive cycles*

has the finite tree model property.

[11] With 'reflexive node' we mean a node w for which $r(w, w)$ holds for one of the accessibility relations.

The standard algorithm for the description logic $\mathcal{ALC}$, for example, generates a finite tree model. If, however, one of the accessibility relations is transitive then a loop check is necessary which may cause a loop in the frame, thus destroying the finite tree model property.

Now we can combine the generated submodel property and the finite tree model property and get a sufficient condition for the singleton set model property.

Theorem 6.16 (Finite Tree Models and Singleton Set Models) *Each description logic which has the finite tree model property and the generated submodel property also has the singleton set model property.*

Proof: Let φ be a satisfiable formula. Since the description logic has the finite tree model property, φ is satisfiable at the root node of a finite tree T *with minimal depth.* If φ would also be satisfiable at another node N of the tree, it would be satisfiable in the frame consisting of the subtree of T with N as its root node (because of the generated submodel property). But this violates the assumption that T was of minimal depth. Therefore φ is satisfied *only* at the root node of T and nowhere else. ∎

Summarizing what is known from the literature and the above results, we can say the following: 'Harmless' description logics, i.e. those with singleton set model property and joinable models property are description logics

1. with only definitional TBoxes,
2. with concept operators $\forall, \exists$, *atleast, atmost* besides the Boolean connectives,
3. with the role operators $\wedge$, $\vee$, ; and *,
4. with serial or reflexive accessibility relations at most and
5. with just Boolean role axioms (role hierarchies).

'Dangerous' constructs are general TBoxes, role negation the role inverse operator and properties of the accessibility relations like transitivity, euclideanness.

6.3 Algorithmic Checks

A description logic with the joinable models property, but lacking the singleton set models property needs a single cardinality test (Def. 4.4) in order to have the decomposition method working. The proof of Theorem 6.16 shows a possibility to have a very cheap single cardinality test: when the consistency of a formula φ is checked with a tableaux algorithm, just inspect the generated model, and see whether it is a finite tree or not. This can be done with almost no overhead in the algorithm. If the frame is a finite tree then this formula at least has the the singleton set models property.

Unfortunately this test is not sufficient. Even if the frame is not a tree model, there may be another model with a tree shaped frame, but the tableaux algorithm has not found it.

There are two possibilities for developing a complete single cardinality test. If the language is sufficiently expressive, one can *internalize* the problem,i.e. reduce it to a plain consistency problem. If this is not possible, one must change a consistency checker to look explicitly for a model where the formula holds only for one single object.

Proposition 6.17 (Internalizing the SCT) *The Single Cardinality Test for a description logic with transitive roles, role hierarchies and the generated submodel property can be internalized.*

Proof: Let $\mathcal{S}$ be the TBox and φ the formula to be tested. The SCT procedure works as follows:

1. Introduce a new *transitive* (non-reflexive) role r.
2. For each role term s occurring in $\mathcal{S}$ and φ add $s \Rightarrow r$.
3. check the consistency of $\varphi \wedge \forall\, r.\neg\varphi$ with $\mathcal{S}$

If $\mathcal{S} \wedge \forall\, r.\neg\varphi \wedge \varphi$ is in fact consistent, i.e. there is an entity $w \in \varphi^{\Im}$, one can assume that $\Im$ is the w-generated submodel. Since $ap_R(w) \subseteq ap_{R,r}(w)$, and r is transitive, $\forall\, r.\neg\varphi$ makes sure that φ is false in all other entities. Thus, w is the only entity satisfying φ. ■

The last resort is a special procedure for checking the single cardinality property.

Procedure 6.18 (Tableaux Procedure for SCT) *One can modify a tableaux procedure for a description logic which can deal with general TBoxes to become a single cardinality test:*

1. *start the tableaux with* $a : \varphi$
2. *for each new entity* b *which is generated by the tableaux rules add* $b : \neg\varphi$. ■

This is actually the same kind of modification of a tableaux system which is needed for dealing with general TBoxes. It has been shown that this can be implemented very efficiently [6].

Description logics lacking the joinable models property need the full algorithmic treatment in the post mortem analysis, the multiple cardinality test (Def. 4.5). This can in fact be implemented with the same kind of modification of a tableaux procedure as for the single cardinality test.

Procedure 6.19 (Tableaux Procedure for MCT) *One can modify a tableaux procedure for a description logic which can deal with general TBoxes to become a Multiple Cardinality Test:*
Input $\mathcal{S}$ and $\varphi_1, n_1, \ldots \varphi_k, n_k$

1. *Start the tableaux with $a_{11} : \varphi_1, \ldots, a_{1n_1} : \varphi_1, \ldots, a_{k1} : \varphi_k, \ldots, a_{kn_k} : \varphi_k$*
2. *For each new entity b which is generated by the tableaux rules add $b : \neg\varphi_1 \wedge \ldots \wedge \neg\varphi_k$.*
3. *Obey the unique name assumption for the a_{ij}.* ■

7 Further Research Topics

In order to make the decomposition method presented in this paper work efficiently and for a large variety of set description languages, a number of quite interesting practical and theoretical problems have to be solved. Let us start with the practical problems.

Efficient Computation of the Atoms

The problem is the following (cf. Sec. 5): Let $\mathcal{S}$ be a fully classified concept hierarchy with information about disjoint concepts. Given a set $\{\varphi_1, \ldots, \varphi_n\}$ of $\mathcal{L}_S$-formulae, compute the set of atoms $\pm\varphi_1 \wedge \ldots \wedge \pm\varphi_n$ consistent with $\mathcal{S}$ while avoiding any unnecessary consistency checks.

This problem is largely independent of the set description language. It is about making optimal use of information about subset relationships and disjointness relationships between concepts.

Singleton Set Model Property

Decide the following property for a set description language or a class of set description languages: Given a consistent set $\mathcal{S}$ of axioms and a formula φ consistent with $\mathcal{S}$: is there always a model $\Im$ for $\mathcal{S}$ such that $\varphi^{\Im}$ is a singleton set?

We gave a sufficient criterion for description logics: a description logic with the generated submodel property and the finite tree model property has the singleton set model property. We also gave some easy to check criteria for the generated submodel property and reduced the finite tree model property to features of the tableaux consistency checkers. But these are only sufficient criteria. There may be more precise characterizations.

Related with the singleton set model property is the following question: Is there an algorithm for checking whether a given formula φ consistent with $\mathcal{S}$, can be made true for exactly one/n objects (worlds)?

In many cases a suitable modification of a tableaux prover can be used for this, but it requires considerable optimizations to get a really efficient test [6].

Joinable Models Property

Decide the following property for a set description language or a class of set description languages: Given a consistent set $\mathcal{S}$ of axioms and two models $\Im_1$ and $\Im_2$ with disjoint domains. Can one join the models without affecting the truth value of formulae?

A sufficient criterion for description logics is the generated submodel property. But this excludes for example logics with the converse role operator. Therefore a more precise characterization is still useful.

Cardinality Constraints in the SDL

We have seen that set description languages can be expressive enough to put constraints on the size of sets. A formula φ may be satisfiable only in models where φ is true for at least two (or more) objects. Using role negation, it may even restrict the size of the domain. Therefore the problem is:

Is there an algorithm that takes as input a set $\mathcal{S}$ of set axioms and a formula φ and returns for example constraints of the form '$|domain| \leq n$' or '$|\varphi| \geq n$'?

Other Bridging Functions

So far we have only assumed that set description languages may be expressive enough to specify constraints on the cardinality of sets, and we introduced mechanisms for dealing with this phenomenon. But what about other bridging functions? Here we must admit that we made one crucial assumption which simplifies things considerably, but which might be too restrictive for practical applications. The assumption is that the other bridging functions have nothing to do with the signature elements occurring in the set description language. To understand what this means, consider the formula

$$\mathit{max\text{-}salary}(\mathit{people} \wedge \mathit{salary} < 10000)$$

The formula $\mathit{people} \wedge \mathit{salary} \leq 10000$ is from a description logic with functional roles. It denotes the set of people having a salary which is less than 10000. *salary* is a functional role. *max-salary* is a bridging function. In the setting presented in this paper, *max-salary* and *salary* have nothing to do with each other. This is clearly not what one wants because $\mathit{max\text{-}salary}(\mathit{people} \wedge \mathit{salary} < 10000) > 10000$ would be a satisfiable formula. How to put in the right links between bridging functions and functional roles is still an open problem.

8 Summary

We have presented a general combination methodology for arithmetical languages and set description language like for example description logic. Both languages are parameters to the combination methodology. The arithmetical language can for example be a linear programming language. The set description language on the other hand, can be a language like description logics for specifying general sets, or a language like relation algebra terms for specifying binary relations. The two languages are combined by 'bridging functions'. These are functions like cardinality which map sets to numbers. In a typical application for the combined language one can use the set description language for specifying a general ontology, for example the products of a company. The mixed language is then used for specifying a concrete problem. This may for example be an optimization problem where cardinalities of sets, total costs, production times, etc. have to be optimized.

The combination methodology uses the technique of atomic decomposition for translating mixed problems into pure arithmetical problems. The inference engines for the arithmetical and the logical languages therefore do not need to interact. The primary inference engine is an arithmetic problem solver. The preparation of the translation operation requires a consistency checker for the set description language. In some cases it is in addition necessary to use a SDL-procedure for filtering the solutions of the arithmetic problem solver.

Before the general methodology can be applied to concrete set description languages, certain theoretical properties of the set description language have to be known, and certain modified consistency algorithms have to be developed. This leads to interesting new problems to be investigated for individual logics and classes of logics (cf. Sec. 5).

References

[1] F. Baader, M. Buchheit, and B. Hollunder. Cardinality restrictions on concepts. *Artificial Intelligence*, 88(1–2):195–213, 1996.

[2] F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, Sydney, Australia, 1991.

[3] Franz Baader and Ulrike Sattler. Description logics with concrete domains and aggregation. In *Proceedings of the 13th European Conference on Artificial Intelligence - ECAI-98*, pages 336–340, 1998.

[4] D. Nardi F. Baader, D. L. McGuinness and P. F. Patel-Schneider, editors. *Description Logics Hanbook.* Cambridge University Press, forthcoming.

[5] Steven Givant Hajnal Andreéka and István Németi. *Decision Problems for Equational Theories of Relation Algebras*, volume 126 of *Memoirs of the American Mathematical Society.* American Mathematical Society, 1997.

[6] Ian Horrocks and Peter Patel-Schneider. Optimizing description logic subsumption. *Journal of Logic and Computation*, 9(3):267–293, 1999.

[7] Hans Jürgen Ohlbach and Jana Koehler. How to extend a formal system with a boolean algebra component. In W. Bibel P.H. Schmidt, editor, *Automated Deduction. A Basis for Applications*, volume III, pages 57–75. Kluwer Academic Publishers, 1998.

[8] Hans Jürgen Ohlbach and Jana Koehler. Modal logics, description logics and arithmetic reasoning. *Journal of Aritificial Intelligence*, pages 1–31, 1999.

[9] Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of IJCAI 91*, pages 466–471. Morgan Kaufmann, 1991.

[10] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Journal of Artificial Intelligence*, 48(1):1–26, 1991.

[11] Marshall H. Stone. The theory of representations for Boolean algebras. *Transactions of American Mathematical Society*, 40:37–111, 1936.

[12] Wiebe van der Hoek. On the semantics of graded modalities. *Journal of Applied Non–Classical Logics*, 2(1):81–123, 1992.