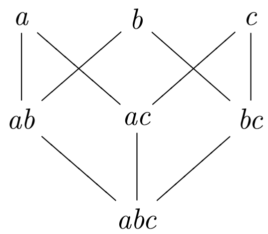
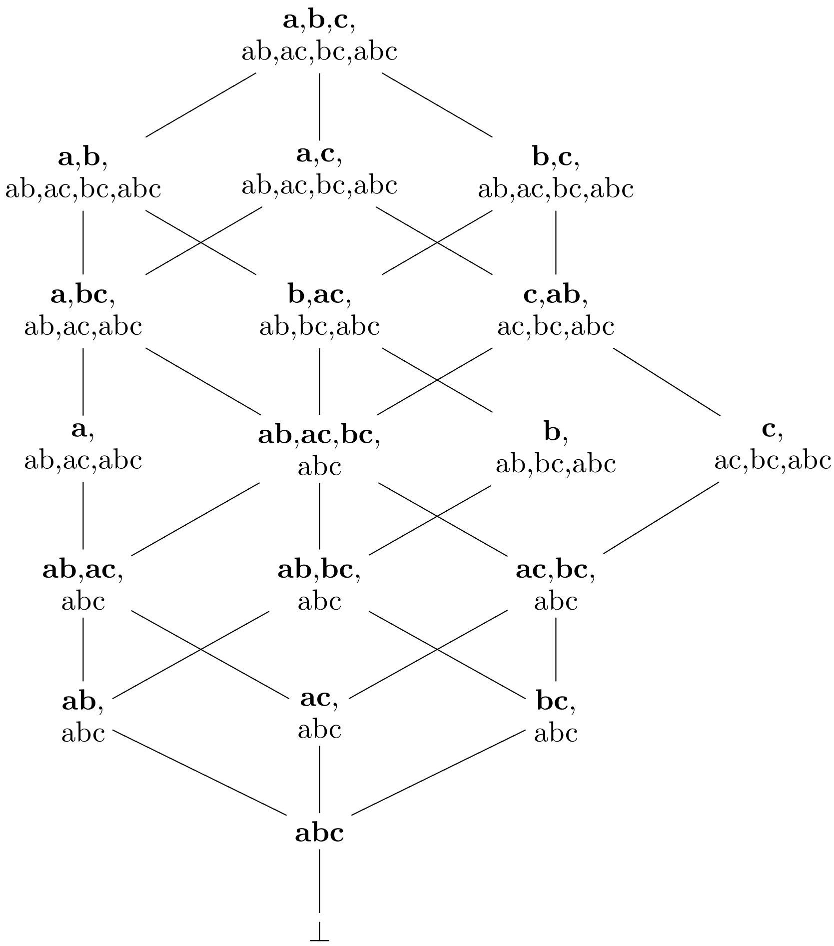
The Developement of

A Lattice Structured Database

Hans Bruun
Informatics and Mathematical Modelling,
Technical University of Denmark

2006





Abstract

In this project we have investigated the possibilities to make a system based on the concept algebra described in [3], [4] and [5]. The concept algebra is used for ontology specification and knowledge representation. It is a distributive lattice extended with attribution operations. One of the main ideas in this work is to use Birkhoff's representation theorem, so we represent distributive lattices using its dual representation: the partial order of join irreducibles. We show how to construct a concept algebra satisfying a given set of *equations*. The universal/initial algebra is usually too big to be useful even in its dual representation, so it is important to use a smaller one from the set of possible solutions. Here the most important contribution seems to be the idea of *inserting terms* in the lattice. For this to make sense we introduced the concept of the most disjoint lattice with respect to a given set of inserted terms, that is the smallest lattice where the inserted terms preserve their value compared to the value in the initial algebra/lattice. The database is the dual representation of this most disjoint lattice. We develop algorithms to construct and make queries to the database.

Contents

List of Figures

Preface

In this report we investigate a new approach to database structuring and querying. The theoretical framework for the database is based on the concept algebra described in [3], [4] and [5]. A concrete concept algebra is a distributive lattice, so the databases we are going to describe will have the contained data structured as a distributive lattice. The prototype program implemented on the basis of the theory developed in this report is called LatBase for lattice structured database.

In sections 2 - 7 we first investigate the simplified case, where attributes are removed so we are left with distributive lattices. From section 8 the full concept algebra with attributes is investigated. All functions, formulas and types/domains are specified in a small subset of the VDMSL specification language (see e.g. [2]). This report is the final result of a working document that changed and growed as the author found solutions to the problems under consideration.

1 Introduction

According to [3] and [4] a concept algebra is a distributive lattice algebra with the binary operators join (+) and meet (*), but as an essential part extended with an arbitrary number of attributes fulfilling rules for distribution of + and * and a rule for strictness. In the sequel we will also call a concept algebra a concept lattice. A user specifies a concrete concept algebra by giving a set of equations between terms in the algebra. As known from universal algebra a set of equations usually specify a set of generated algebras ranging from the most general initial algebra to the smallest so called terminal algebra.

In this report we propose a way to construct a specific concept algebra by giving a set of equations which the algebra has to fulfill, and furthermore by *inserting* terms in the lattice. A concept lattice – specified by a set of equations and a set of inserted terms – is the smallest generated concept algebra fulfilling the equations such that the inserted terms evaluate to the same value as in the initial algebra. This concept algebra is called the most disjoint concept algebra. The representation of the distributive lattice is based on Birkhoff's representation theorem for finite distributive lattices (see e.g. [1]). Birkhoff's representation theorem has been used in other knowledge representation systems (see e.g. [6]).

Before going into details we first show a few small examples of using LatBase , the developed prototype program based on the proposed algorithms.

1.1 Example, a real estate database

We consider a small database with sales information about real estates. The database is almost like a relational database table with the columns `ID`, `LOC`, `SIZE`, `PRICE` and `COND`. In the LatBase -system we may insert information about each real estate as an algebraic term:

```
insert
  flat *ID(id1)*LOC(loc3)*SIZE(large) *PRICE(medium)*COND(medium),
  flat *ID(id2)*LOC(loc1)*SIZE(small) *PRICE(large) *COND(xlarge),
  villa*ID(id3)*LOC(loc5)*SIZE(xlarge)*PRICE(xlarge)*COND(small),
  villa*ID(id4)*LOC(loc2)*SIZE(large) *PRICE(large) *COND(large),
  flat *ID(id5)*LOC(loc5)*SIZE(xlarge),
```

```
farm *ID(id6)*LOC(loc4)*AREA(medium) *PRICE(medium)
```

All the small letter names above are concept names and capital letter names are attribute names. The concepts `small`, `medium`, `large`, `xlarge` designate four different sizes and are used as a measure for sizes, prices and conditions of real estates. The concepts `loc1` – `loc5` designate five specific geographical areas and are used to indicate the physical location of a real estate. The concepts `id1` – `id6` are used as unique names. Finally the concepts `flat`, `villa` and `farm` denote real estates that are flats, villas and farms respectively.

In the example above each inserted term corresponds to a tuple in a relational database. Informally, in a relational database formulation, we may thus consider the first term as a tuple in the `flat` relation with the `ID` attribute `id1`, the `LOC` attribute equal to `loc3`, the `SIZE` attribute equal to `large`, the `PRICE` attribute equal to `medium` and the `COND` attribute equal to `medium`. When modeling a classical database relation, `flat` would be absent and we would have the same set of attributes in all tuples. As seen in the last two terms in the example above the LatBase -system does not force such a homogeneity constraint.

Given the database above we can now make a query to the system by writing a term. First we ask for all real estates having size `large`:

```
SIZE(large)
```

and the system answers with

```
{[flat, ID(id1), LOC(loc3), SIZE(large), PRICE(medium), COND(medium)]
 [villa, ID(id4), LOC(loc2), SIZE(large), PRICE(large), COND(large)]}
```

The answer from the system is also a term, but written in a special notation. The term is in disjunctive normal form, the conjunction of a set of concepts $\{c_1, c_2, \ldots, c_n\}$ is written as $[c_1, c_2, \ldots, c_n]$ and the disjunction of a set of conjunctions is written as $\{[\ldots][\ldots]\ldots[\ldots]\}$. So from the above answer we can see that there are two real estates in the answer, a flat and a villa. Each returned real estate is described by a conjunction of a set of concepts, here mainly attributes.

Next we ask for a real estate which is either a `farm` or a `villa`, which is located in either the area `loc2` or the area `loc4` and which has a price `medium` or `large`:

```
(farm + villa) * LOC(loc2 + loc4) * PRICE(medium + large)
```

Now the system answers with:

```
{[villa, ID(id4), LOC(loc2), SIZE(large), PRICE(large), COND(large)]
 [farm, ID(id6), LOC(loc4), PRICE(medium), AREA(medium)]}
```

1.2 The Concept Algebra

In the Concept Algebra terms are formed from a set of concept identifiers, and operators on concepts. The two binary operators + and * are required to obey the axioms for idempotency, commutativity, associativity, absorption, and distributivity. This means that a concrete concept algebra always is a distributive lattice with + being lattice join and * being lattice meet. From lattice theory we know that any distributive lattice is isomorphic to a lattice of sets. Consequently, in the framework of concept algebras we usually consider a concrete concept algebra as a lattice of sets, where the sets represents (the extension of) the concepts. The binary operators + and * are then set union ($\bigcup$) and set intersection ($\bigcap$). The special concept identifier *null* is the bottom element and corresponds to the empty set in the set model.

Besides the two binary operators we may introduce an arbitrary set of unary operators corresponding to attributes. All introduced attributes a must obey the following axioms

$$\begin{aligned} a(x+y) &= a(x)+a(y) \\ a(x*y) &= a(x)*a(y) \\ a(null) &= null \end{aligned}$$

These axioms ensure that * informally can be understood as a tuple constructor (see [3] and [5]).

Let us reconsider the first inserted term in the previous example

```
insert  flat *ID(id1)*LOC(loc3)*SIZE(large) *PRICE(medium)*COND(medium)
```

Here e.g. the attribute `LOC` is a function which maps the location `loc3` to the concept `LOC(loc3)` which (extensionally) designates the set of all entities located in the `loc3`-area. We can now interpret the term as the intersection of several sets:

- `flat`: the set of all real estates that are flats.
- `ID(id1)`: the set of all entities with the unique identifier `id1`. We should ensure that only one entity (in the database) has this identifier, so the set becomes a singleton set.
- `LOC(loc3)`: the set of all entities located in the area `loc3`.
- `SIZE(large)`: the set of all entities, which have the size `large`.
- etc.
  ⋮

The result is a (singleton) set containing the described real estate. But informally it is convenient to think of the result as a tuple.

According to lattice theory (see e.g. [1]) a lattice is also a partial order with the ordering relationship $\leq$ defined by

$$x \leq y \text{ iff } x = x * y$$

In the concept algebra this partial order relation is usually called the *isa*-relation.

In the LatBase -system the database is a concrete concept algebra, which is determined by the inserted terms. However, the user of the system also has the possibility to add equations which specify possible relations between concepts in the lattice. These equations are used to further constrain the lattice (or concept algebra) side by side with the general axioms for the concept algebra. The equations may specify both equalities and *isa*-relations.

1.3 Example with equations

In the previous example we used the LatBase -system almost as a traditional relational database. In this example we first give the LatBase -system a set of equations which specify a small ontology for the considered real estate domain (the numbers to the right of the equations are not a part of the input):

```
equations
  home = villa + flat,                         (1)
  mes >= small + medium + large + xlarge,      (2)
  loc >= reg1 + reg2,                          (3)
  reg1 >= loc1 + loc2 +loc3,                   (4)
  reg2 >= loc4 + loc5 + loc3,                  (5)
  home <= SIZE(mes)*PRICE(mes)*LOC(loc),       (6)
  GoodCond = COND(large+xlarge),               (7)
  Fancy >= LOC(loc3) * COND(small+medium)      (8)
```

In the equations we have introduced some new concepts many of which are generalizations of the concepts used in the first example.

1. `home` is a generalization of `villa` and `flat`, so a `home` is either a `villa` or a `flat`. In the set interpretation the set of homes is the union of the set of villas and the set of flats.

2. `mes` is a general measure including the previously used concrete measures `small`, `medium`, `large` and `xlarge`.

3. The concept `loc` designates the complete geographical area for which we have real estates in the database. According to this equation the area includes the two (overlapping) sub-regions `reg1` and `reg2`,

4. where `reg1` contains (amongst others) the areas `loc1`, `loc2` and `loc3`

5. and `reg2` contains the areas `loc4`, `loc5` and `loc3`.

6. The set of homes (for sale) is a subset/specialization of the entities having a `SIZE` and `PRICE` attribute with a value being some measure `mes` and a `LOC`ation attribute with a value being some location `loc`. Stated differently, a home (-description in the database) must have at least a `SIZE`, `PRICE` and a `LOC`ation attribute with the above mentioned values.

7. Finally is introduced some useful concepts. The `GoodCond` concept designates the set of all entities having a `COND` attribute value which is either `large` or `xlarge` and

8. the concepts `Fancy` denotes the set of all entities which are located in the area `loc3` and which is in a modest condition.

Assume we in this new database insert the same terms as in the previous example and then ask for all home's having size `large`:

```
home * SIZE(large)
```

The system responds with the same two real estates as in the first example:

```
{[home, villa, GoodCond,
        SIZE(mes), SIZE(large), PRICE(mes), PRICE(large),
        LOC(loc), LOC(reg1), LOC(loc2),
        COND(mes), COND(large), ID(id4)]
```

```
[home, flat, Fancy,
       SIZE(mes), SIZE(large), PRICE(mes), PRICE(medium),
       LOC(loc), LOC(reg1), LOC(reg2), LOC(loc3),
       COND(mes), COND(medium), ID(id1)]}
```

Notice that the description of each real estate— besides the properties originally inserted — now also contains properties which follows from the given ontology. Consider e.g. the second real estate. Besides being a `flat` as specified in the inserted term, from equation (1) it is also known to be a `home` and from equation (8) it is known to be `Fancy`. It is located in location `loc3` as specified in the inserted term, but we also know (from equations (4) and (5)) that this location is in the regions `reg1` and `reg2`.

We can of course also use the concepts introduced in the ontology to make queries, e.g. ask for home's located in region `reg1` which are in good condition:

```
home * LOC(reg1) * GoodCond:
```

The systems responds with the two `home`'s shown below:

```
{[home, villa, GoodCond,
       SIZE(mes), SIZE(large), PRICE(mes), PRICE(large),
       LOC(loc), LOC(reg1), LOC(loc2),
       COND(mes), COND(large), ID(id4)]
 [home, flat, GoodCond,
       SIZE(mes), SIZE(small), PRICE(mes), PRICE(large),
       LOC(loc), LOC(reg1), LOC(loc1),
       COND(mes), COND(xlarge), ID(id2)]}
```

As can be seen, they are both located in locations, which are in region `reg1`, and they are both in a good condition. □

In the previous examples the real estates inserted in the database are atomic, i.e. they are located just above the bottom (*null*) in the lattice. In the LatBase system the values need not be atomic as illustrated in the next example.

1.4 Example, human

In this example we have the concepts `h`,`m`,`f`, `c`, `a` (for human, male, female, child, adult). In the following input to the LatBase -system we have two equations for human, one that equals human with the union of child and adult, and one that equals human with the union of male and female:

```
equations h = c + a, h = m + f
insert h, c*a
```

We can now ask for the concept female by writing the term `f`. If we also want to see all the sub-concepts of female we precede the term with the keyword `downset`. So if we make the query "`downset f`" the system responds with

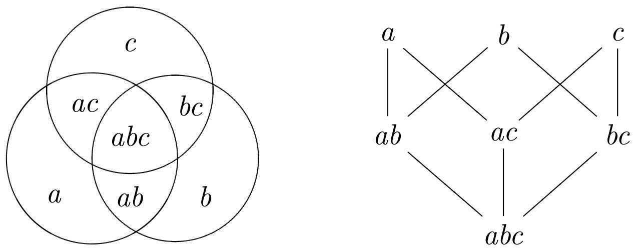
```
{[h, f, c],[h, f, a],[h, f, c, a]}
```

Here the concepts $[h, f, c]$ and $[h, f, a]$ corresponds to the concepts girl and woman. The inserted term `c*a` forces child and adult to overlap so we also get the sub-concept $[h, f, c, a]$ representing female teenagers. □

2 The Partial Order of Concept Intersections

In this and the following sections (sections 2 - 7) we first investigate the simplified case, where attributes are removed so we are left with distributive lattices. So the goal is to construct a concrete generated distributive lattice satisfying the given set of equations. So the first question we could ask is: what kind of elements should we have in the lattice?

Given a finite set of concepts we can describe the mutual relationship between these concepts by the set of intersections between the concepts. Figure 1 and 2 illustrate the idea.

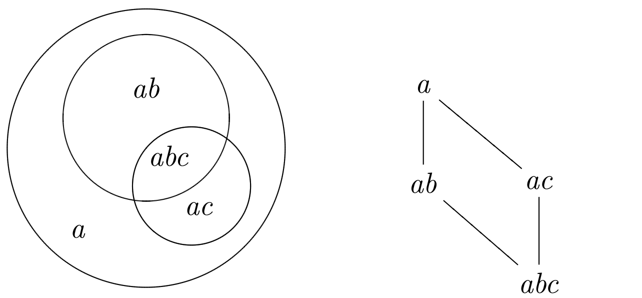


Venn Diagram for the concepts a, b and c — Hasse Diagram for the corresponding partial order

Figure 1: Concept relations described by concept-intersections

In figure 1 we have the most general situation where all possible intersections between the given concepts a, b and c exist. The given intersections are arranged in a partial order. By conceiving concepts as sets we naturally put an intersection between two sets below the two sets.

Every subset of this partial order describes a more specific relationship between the concepts a, b and c as shown in figure 2. Here b and c are both subsets of a so the set of intersections now is $\{a, ab, ac, abc\}$. Notice, that if two intersections are identical the most specific intersection is used in the partial order. In figure 2 the intersections $bb \sim b$ and ab are identical – as b is a subset of a – so ab is used.



Venn Diagram for the concepts a, b and c — Hasse Diagram for the corresponding partial order

Figure 2: Concept relations described by concept-intersections

Below we define concept-intersections as a non-empty set of named concepts and a partial order as a set of concept-intersections:

types

1.0 $C = \mathsf{token}$ — The type of concept constants;

2.0 $Cset = C\text{-}\mathsf{set}$

.1 $\mathsf{inv}\ cset \triangleq cset \neq \{\}$;

3.0 $CI ::\ \ Cset$;

4.0 $PO = CI\text{-}\mathsf{set}$

Here CI ($\sim$ *concept-intersection*) is the type of non-empty sets of concepts representing the intersection between these concepts.

5.0 $\mathcal{P} : Cset \to PO$

.1 $\mathcal{P}\,(cs) \triangleq \{mk\text{-}CI(cs') \mid cs' : Cset \cdot cs' \subseteq cs\}$

Given a set cs of concepts, $\mathcal{P}(cs) : PO$ defines the power-set consisting of all possible subsets of concept-intersections. It is a partial order with the ordering relation ISA_P defined below:

6.0 $ISA_P : CI \times CI \to \mathbb{B}$

.1 $ISA_P\,(mk\text{-}CI(cs_1), mk\text{-}CI(cs_2)) \triangleq cs_2 \subseteq cs_1$

In the sequel any subset p of $\mathcal{P}(cs)$ is considered a partial order with the same (induced) ordering ISA_P.

Notation When showing examples we often use a shorthand notation for concept-intersections. If a, b and c are one-letter concept names, the concept-intersection $mk\text{-}CI(\{a, b, c\})$ is just written as abc (as already shown in the figures 1 and 2). When the involved concepts are more complex, $mk\text{-}CI(\{c_1, c_2, \ldots, c_n\})$ is sometimes written as $[c_1, c_2, \ldots, c_n]$.

Covers For a given partial order $p : PO$ the cover relation is defined as

$$ci_1 \preceq_p ci_2 \text{ iff } \forall\, ci \in p \cdot ISA_P(ci_1, ci) \wedge ISA_P(ci, ci_2) \;\Rightarrow\; ci = ci_1 \vee ci = ci_2$$

Later we will need the set of elements immediately below a given element ci, called the lower covers of ci:

7.0 $lCOVERS_P : PO \to CI \to CI\text{-}\mathsf{set}$

.1 $lCOVERS_P\,(p)(ci) \triangleq \{ci' \mid ci' \in p \cdot ci' \preceq_p ci\}$

The set of upper covers may be defined in a similar way.

2.1 Anti-chains and Down-sets

Anti-chains A subset of a partial order is an *anti-chain* iff every pair of different elements in the subset are non-comparable:

8.0 $IsAntiChain : CI\text{-set} \rightarrow \mathbb{B}$

.1 $IsAntiChain\,(ac) \triangleq$

.2 $\quad \forall\, ci_1 \in ac, ci_2 \in ac \cdot ci_1 \neq ci_2 \;\Rightarrow\; \neg\, ISA_P(ci_1, ci_2) \wedge \neg\, ISA_P(ci_2, ci_1)$

Down-sets A set *ciset* of elements in a partial order $p : PO$ is called a down-set iff it is closed under going down in the partial order:

9.0 $IsDownset : PO \rightarrow CI\text{-set} \rightarrow \mathbb{B}$

.1 $IsDownset\,(p)(ciset) \triangleq \forall\, ci_1 \in ciset, ci_2 \in p \cdot ISA_P(ci_2, ci_1) \;\Rightarrow\; ci_2 \in ciset$

Notice that according to the definition above, everything in p which is below some element in *ciset* must also be in *ciset*. Thus, *ciset* need not be a subset of p to get an affirmative answer.

Given a set *ciset* of elements in $p : PO$ the down-set of *ciset* is all the elements in p below some element in *ciset* :

10.0 $DownSet : PO \rightarrow CI\text{-set} \rightarrow CI\text{-set}$

.1 $DownSet\,(p)(ciset) \triangleq \{ci \mid ci \in p \cdot \exists\, ci' \in ciset \cdot ISA_P(ci, ci')\}$

.2 $\text{pre}\;\; ciset \subseteq p$

The down-set $DownSet(p)(\{ci_1, ci_2, \ldots, ci_n\})$ is often written as $\downarrow\{ci_1, ci_2, \ldots, ci_n\}$ when p is assumed. It is easily seen that $DownSet(p)(ciset)$ is the smallest down-set containing *ciset* provided $ciset \subseteq p$.
Concerning down-sets we have

11.0 $ISA_P(ci_1, ci_2) \equiv DownSet(p)(\{ci_1\}) \subseteq DownSet(p)(\{ci_2\})$

12.0 $DownSet(p)(cis_1 \cup cis_2) = DownSet(p)(cis_1) \cup DownSet(p)(cis_2)$

13.0 $IsDownset(p)(cis_1) \wedge IsDownset(p)(cis_2) \;\Rightarrow\; IsDownset(p)(cis_1 \cup cis_2)$

.1 $IsDownset(p)(cis_1) \wedge IsDownset(p)(cis_2) \;\Rightarrow\; IsDownset(p)(cis_1 \cap cis_2)$

For down-sets in different partial orders we have some useful relations: Let $cis \subseteq p \subseteq \mathcal{P}(cset)$ for some *cset*, then

14.0 $DownSet(p)(cis) = DownSet(\mathcal{P}(cset))(cis) \cap p$

i.e. the down-set is that part of the down-set of *cis* in $\mathcal{P}(cset)$ which is in p. We also have

15.0 $DownSet(p \setminus d)(cis) = DownSet(p)(cis) \setminus d$

.1 $p_1 \subseteq p_2 \;\Rightarrow\; DownSet(p_1)(cis) = DownSet(p_2)(cis) \cap p_1$

Anti-chains and Down-sets in Collaboration Every downset *dset* in a partial order p has a unique anti-chain of which it is a down-set:

16.0 $\forall\, p : PO, dset : CI\text{-set} \cdot$

.1 $\quad IsDownset(p)(dset) \;\Rightarrow$

.2 $\quad \exists!\, ac : CI\text{-set} \cdot ac \subseteq dset \wedge IsAntiChain(ac) \wedge dset = DownSet(p)(ac)$

In the sequel we use this maximal anti-chain as a representation for the downset, and we will just call it the anti-chain of the down-set:

17.0 $AntiCh\,(cis : CI\text{-set})\ ac : CI\text{-set}$

.1 $\mathsf{post}\ ac \subseteq cis \wedge IsAntiChain(ac) \wedge \forall\, ci_1 \in cis \cdot \exists\, ci_2 \in ac \cdot ISA_P(ci_1, ci_2)$

Thus $AntiCh(cis)$ is an anti-chain subset of cis such that all other ci's in cis are below some element in the anti-chain (in the partial order p). Combining 16 and 17 gives

18.0 $\forall\, p : PO, dset : CI\text{-set} \cdot$

.1 $\quad IsDownset(p)(dset) \;\Rightarrow\; DownSet(p)(AntiCh(dset)) = dset$

A set cis of concept-intersections is always a downset in the partial order consisting of the set cis itself:

19.0 $\forall\, cis : CI\text{-set} \cdot IsDownSet(cis)(cis)$

Thus, it is always meaningful to ask for the maximal anti-chain of a set of concept-intersections. For the $AntiCh$-function we have:

20.0 $\forall\, ac : CI\text{-set} \cdot IsAntiChain(ac) \;\Rightarrow\; AntiCh(ac) = ac$

21.0 $\forall\, ds, ds1 : CI\text{-set} \cdot AntiCh(ds) \subseteq ds1 \subseteq ds \;\Rightarrow\; AntiCh(ds1) = AntiCh(ds)$

In words, if a subset $ds1$ of a downset ds includes the anti-chain of ds, then $ds1$ has the same anti-chain. Finally we have

22.0 $DownSet(p)(AntiCh(cis)) = DownSet(p)(cis)$

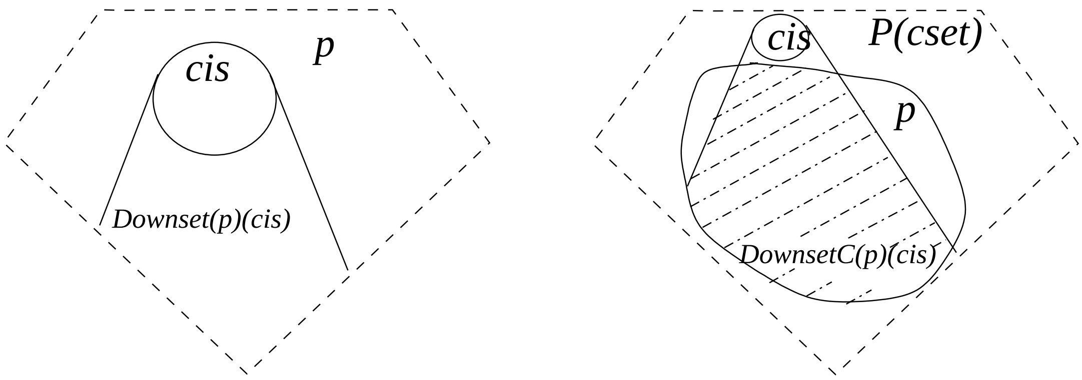


Figure 3: *Downset* and *DownsetC*

DownsetC. In lattice theory, when talking about "downset of *ciset*", it is always assumed that *ciset* is a subset of the considered partial order (here p). In later sections it will be convenient to ask for $DownSet(p)(ciset)$ even when *ciset* is not a subset of p. In order to make it formally correct, we define a new downset function without the precondition $ciset \subseteq p$:

23.0 $DownSetC : PO \rightarrow CI\text{-set} \rightarrow CI\text{-set}$

.1 $DownSetC\,(p)(ciset) \triangleq \{ci \mid ci \in p \cdot \exists\, ci' \in ciset \cdot ISA_P(ci, ci')\}$

The difference between *Downset* and *DownsetC* is illustated in figure 3. The name *DownSetC* ($\sim$ *DownSetCut*) emphasizes that some of the elements in *ciset* may be cut away. From 14 it is easy to see that $DownSetC(p)(ciset)$ always is a downset (in p), even when *ciset* is not a subset of p. However, if *ciset* and p are disjoint the downset may sometimes be empty. All the down-set properties from 11 to 22 are also valid for *DownSetC*.

Finally consider

24.0 $\forall\, cis : CI\text{-set} \cdot cis \subseteq p \;\Rightarrow\; AntiCh(cis) = AntiCh(DownSet(p)(cis))$

which is true for both *DownSet* and *DownSetC*, whereas if we do not assume $cis \subseteq p$ then we must apply *DownSetC* and we have

25.0 $\neg\,\forall\, cis : CI\text{-set} \cdot AntiCh(cis) = AntiCh(DownSetC(p)(cis))$

because now $AntiCh(cis)$ is a subset of *cis*, but $AntiCh(DownSetC(p)(cis))$ is a subset of p.

2.2 Projection

In subsequent sections we will need the concept of projection. Given a partial order p and a set of concept-intersections *cis*, the projection of *cis* in p is the anti-chain in p having the same down-set in p as *cis* has. The definition of projection is illustrated in fig 4.

26.0 $CISproj\;(p : PO)(cis : CI\text{-set})\;\; ac : CI\text{-set}$

.1 post $ac \subseteq p\,\wedge$

.2 $\quad IsAntiChain(ac)\,\wedge$

.3 $\quad DownSetC(p)(ac) = DownSetC(p)(cis)$

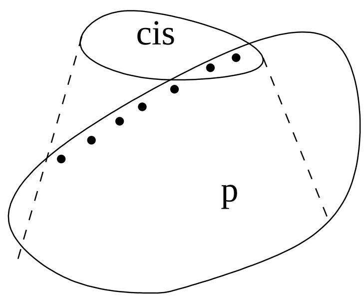


Figure 4: Definition of projection

Projecting a set of concept-intersections *cis* or its anti-chain into a partial order yields the same result:

27.0 $CISproj(p)(AntiCh(cis)) = CISproj(p)(cis)$

Let the left and right hand side anti-chains be $ac1$ and $ac2$ respectively. To see that the equation above is true, we notice that $ac1$ and $ac2$ are both anti-chains in p and show that they have the same down-set in p. For $ac1$ we have

$$\begin{aligned} & DownSetC(p)(ac1) \\ & \qquad\qquad\qquad\qquad \text{from } 26.3 \\ & = DownSetC(p)(AntiCh(cis)) \\ & \qquad\qquad\qquad\qquad \text{from } 22 \\ & = DownSetC(p)(cis) \end{aligned}$$

For $ac2$ we have

$$\begin{aligned} & DownSetC(p)(ac2) \\ & \qquad\qquad\qquad\qquad \text{from } 26.3 \\ & = DownSetC(p)(cis) \end{aligned}$$

So $ac1$ and $ac2$ have the same down-set in p, consequently they are the same anti-chain.

3 The Concept Lattice

From a finite partial order $p : PO$ of concept-intersections we now define the family of all down-sets in p:

28.0 $\mathcal{O} : PO \to CI\text{-set-set}$

.1 $\mathcal{O}\,(p) \triangleq \{DownSet(p)(ac) \mid ac : CI\text{-set} \cdot ac \subseteq p \wedge IsAntiChain(ac)\}$

According to lattice theory $\mathcal{O}(p)$ is a lattice of sets with the ordering relation, the join- and the meet-operation corresponding to the subset-relation, set-union and set-intersection respectively:

29.0 $ISA_L : CI\text{-set} \times CI\text{-set} \to \mathbb{B}$

.1 $ISA_L\,(cis_1, cis_2) \triangleq cis_1 \subseteq cis_2$

30.0 $Join_L : CI\text{-set} \times CI\text{-set} \to CI\text{-set}$

.1 $Join_L\,(cis_1, cis_2) \triangleq cis_1 \cup cis_2$

31.0 $Meet_L : CI\text{-set} \times CI\text{-set} \to CI\text{-set}$

.1 $Meet_L\,(cis_1, cis_2) \triangleq cis_1 \cap cis_2$

The *bottom-element* of the lattice $\mathcal{O}(p)$ is the empty down-set, and the *top-element* is p. As $\mathcal{O}(p)$ is a finite lattice of sets it is also a finite distributive lattice.

Figure 5 shows the lattice $\mathcal{O}(p_1)$ and figure 6 the lattice $\mathcal{O}(p_2)$ where p_1 and p_2 are the two partial orders shown in figure 1 and 2.

The elements in $\mathcal{O}(p)$ are sets of concept-intersections. In the figures 5 and 6 each set is shown on two lines. The upper bolded line shows the concept-intersections in the anti-chain of which the element is a down-set. The second line shows the remaining concept-intersections in the down-set. The empty down-set is the bottom-element shown as $\perp$. In the set-interpretation of concepts one may think of the lattice elements as the union of set-intersections. The first line is the union of non-comparable sets and the second line is the

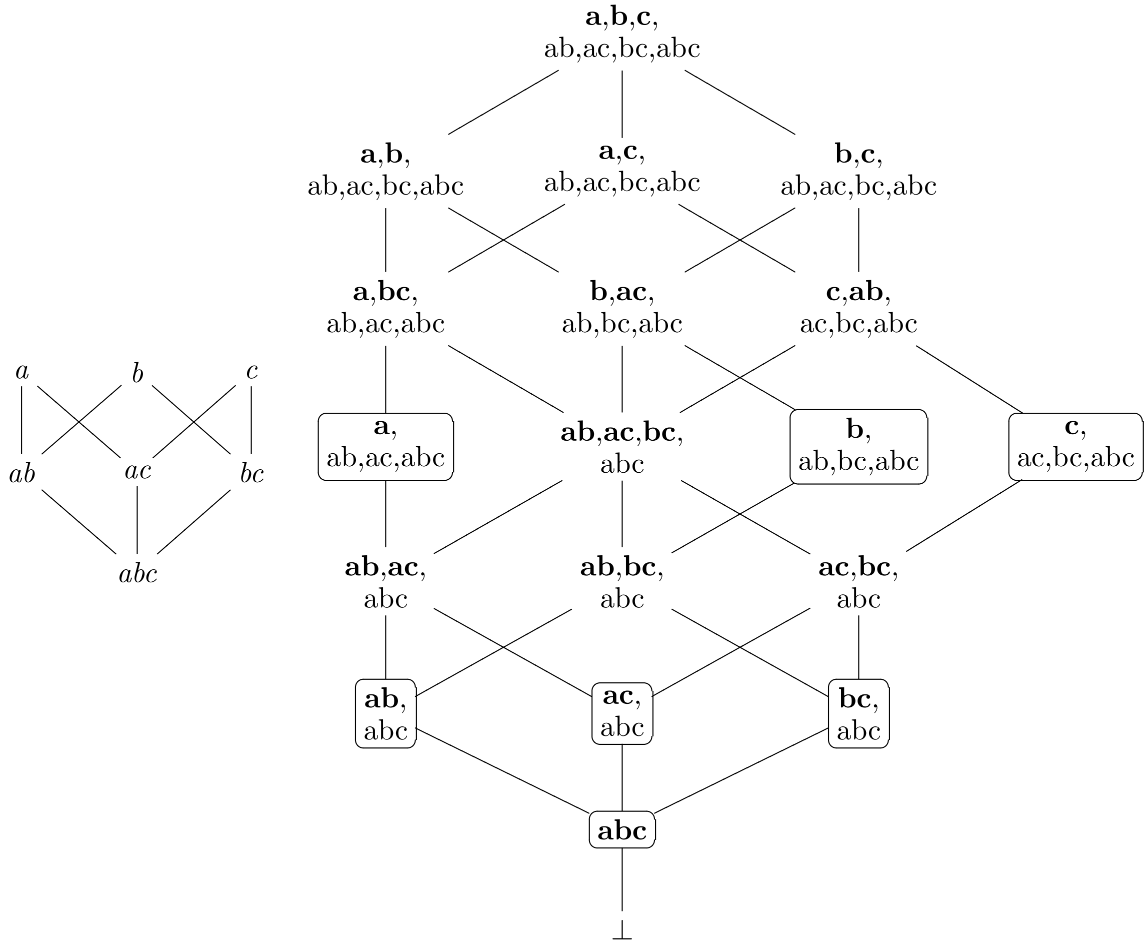


Figure 5: The partial order p from fig. 1 and the corresponding lattice $\mathcal{O}(p)$

union of the subsets in these sets. Given an element $cis \in \mathcal{O}(p)$ the anti-chain part can be extracted by $AntiCh(cis)$ (17).

In the lattice to the right in figure 5 and 6 some of the lattice elements are framed. As can be seen, these lattice elements have exactly one lower cover. Such elements cannot be constructed as the join of other elements in the lattice and are consequently called *join-irreducible* elements. We will see in the next section that these *join-irreducible* elements play a crucial role in the representation of distributive lattices.

3.1 Birkhoff's Representation Theorem

The relationship between a partial order $p : PO$ and the lattice $\mathcal{O}(p)$ is described in general in Birkhoff's representation theorem for finite distributive lattices. We denote the set of join-irreducible elements in a lattice L by $\mathcal{J}(L)$. From [1, pages 171 – 172] we have

"Let L be a finite distributive lattice. Then the map $\eta : L \to \mathcal{O}(\mathcal{J}(L))$ defined by

$$\eta(a) = \{x \mid x \in \mathcal{J}(L) \cdot x \leq a\}$$

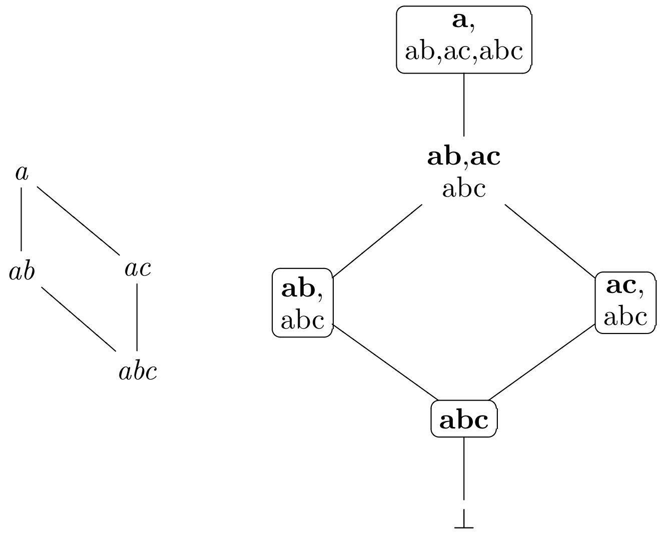


Figure 6: The partial order p from fig. 2 and the corresponding lattice $\mathcal{O}(p)$

> is an isomorphism of L onto $\mathcal{O}(\mathcal{J}(L))$.

Furthermore

> Suppose p is a finite ordered set. Then the map $\varepsilon : x \mapsto \downarrow x$ is an order-isomorphism from p onto $\mathcal{J}(\mathcal{O}(p))$.

The two statements above reveal a duality between finite distributive lattices and finite ordered sets. Up to isomorphism, we have a one-to-one correspondence

$$\mathcal{O}(p) = L \rightleftarrows p = \mathcal{J}(L)$$

So special properties of the finite distibutive lattice L are reflected in special properties of its dual set p." □

For the partial orders $p : PO, p \subseteq \mathcal{P}(cset)$ and the corresponding lattices $\mathcal{O}(p)$ we are considering in this paper some of this can be concretized as follows. The map ε corresponds to the function *DownSet* applied to $\{ci\}$, $ci \in p$. Hence

$$\mathcal{J}(\mathcal{O}(p)) = \{DownSet(p)(\{ci\}) \mid ci \in p\}$$

So the join-irreducible elements in $\mathcal{O}(p)$ is characterized by having a single concept-intersection in the anti-chain part. In figure 5 and 6 one can see that the elements having this property are exactly the framed elements, i.e. the join-irreducible elements.

The function *AntiCh* is the function mapping a join-irreducible element in the lattice back to the corresponding concept-intersection in p:

$$\begin{array}{l} \forall\, p : PO, cis : CI\text{-}\mathsf{set}, ci \in p \cdot \\ \quad cis = DownSet(p)(\{ci\}) \;\Leftrightarrow\; AntiCh(cis) = \{ci\} \end{array}$$

In the sequel we use this relation between a partial order p and the lattice $\mathcal{O}(p)$ to find a lattice satisfying a given set of equations.

3.2 Computing Covers in $\mathcal{O}(p)$ from p

Now given a partial order $p : PO$ we would like to construct or inspect the lattice $\mathcal{O}(p)$. So assume we already have an element e in $\mathcal{O}(p)$ we should be able to inspect elements in $\mathcal{O}(p)$ immidiatly above or below e, i.e. we need a function to compute the set of lower- and upper covers of e in $\mathcal{O}(p)$ from the partial order p. A function to compute the set of lower covers is defined below.

32.0 $lCOVERS_L : PO \rightarrow CI\text{-set} \rightarrow CI\text{-set-set}$
.1 $lCOVERS_L\,(p)(cis) \triangleq$
.2 $\quad \text{let } ac = AntiCh(cis) \text{ in}$
.3 $\quad \{cis \setminus \{ci\} \mid ci \in ac\}$

Let $cis \in \mathcal{O}(p)$. To see that $lCOVERS_L(cis)$ actually cumputes the set of lower covers of cis in $\mathcal{O}(p)$ let $lcis \in lCOVERS_L(cis)$. Furthermore let $ac = AntiCh(cis)$ so $cis = DownSet(p)(ac)$. According to the definition of $lCOVERS_L(cis)$ there is a $ci \in ac$ such that $lcis = cis \setminus \{ci\}$. We now have

$$\begin{aligned} lcis &= DownSet(p)(ac) \setminus \{ci\} \\ &= DownSet(p)(ac \setminus \{ci\}) \cup DownSet(p)(lCOVERS_P(ci)) \\ &= DownSet(p)(ac \setminus \{ci\} \cup lCOVERS_P(ci)) \end{aligned}$$

hence $lcis \in \mathcal{O}(p)$. From $lcis = cis \setminus \{ci\}$ we have $lcis \subset cis$ so $ISA_L(lcis, cis)$. Finally assume $cis' \in \mathcal{O}(p)$ and $lcis \subset cis' \subset cis$ but this is obviously a contradiction so $lcis \preceq_L cis$.

From the equations above we see that the set of lower covers could just as well be computed by the function defined below:

33.0 $lCOVERS_{LP} : PO \rightarrow CI\text{-set} \rightarrow CI\text{-set-set}$
.1 $lCOVERS_{LP}\,(p)(cis) \triangleq$
.2 $\quad \text{let } ac = AntiCh(cis) \text{ in}$
.3 $\quad \{DownSet(p)((ac \setminus \{ci\}) \cup lCOVERS_P(ci)) \mid ci \in ac\}$

The set of upper covers can be computed in a similar way.

34.0 $uCOVERS_L : PO \rightarrow CI\text{-set} \rightarrow CI\text{-set-set}$
.1 $uCOVERS_L\,(p)(cis) \triangleq$
.2 $\quad \{cis \cup \{ci\} \mid ci \in p \setminus csi \cdot IsDownSet(p)(cis \cup \{ci\})\}$

One get an upper-cover of cis by adding an arbitrary new single concept-intersection ci such that the new set is a down-set.

4 Lattices as Algebras

In the previous sections we have used the ordered set view of the distributive lattices. In order to be able to talk about distributive lattices satisfying a set of equations we must also use the algebraic view. Here a distributive lattice is an algebra with the two binary operators *Join* and *Meet* satisfying the axioms shown below. *Join* and *Meet* are represented by the two infix operators $+$ and $*$:

Idempotency	$X + X = X, \quad X * X = X$
Commutativity	$X + Y = Y + X, \quad X * Y = Y * X$
Associativity	$X + (Y + Z) = (X + Y) + Z, \quad X * (Y * Z) = (X * Y) * Z$
Absorbtion	$X * (X + Y) = X, \quad X + X * Y = X$
Distributivity	$X * (Y + Z) = X * Y + X * Z, \quad X + Y * Z = (X + Y) * (X + Z)$
Bounds	$X + \bot = X, \quad X * \bot = \bot, \quad X + \top = \top, \quad X * \top = X$

So assume *cset* is a set of concepts and $p \subseteq \mathcal{P}(cset)$ is a partial order of concept-intersections. The lattice $\mathcal{O}(p)$ can now be viewed as a (one sorted) algebra

35.0 $\mathcal{LA}(cset, p) =< \mathcal{O}(p); Join_L, Meet_L, C_L(p)(cset), \textsc{top}_L, \textsc{bottom}_L >$

Here $\mathcal{O}(p)$ is the carrier set and $Join_L$ and $Meet_L$ are the two binary operators defined in 30 and 31. Corresponding to the set of named concepts $c \in cset$ we now have a set of constants/values in $\mathcal{O}(p)$:

$$C_L(p)(cset) = \{cValue_L(p)(c) \mid c \in cset\}$$

The value of a named concept $c \in cset$ is $cValue_L(p)(c)$ where

36.0 $cValue_L : PO \to C \to CI\text{-set}$

.1 $cValue_L\,(p)(c) \triangleq DownSetC(p)(\{mk\text{-}CI(\{c\})\})$

In the definition above notice that *DownSetC* (rather than *DownSet*) has been used, because the concept-intersection $mk\text{-}CI(\{c\})$ is not necessarily in p. From the discussion in section 2.1 we know that the value of $cValue_L(p)(c)$ is a down-set in p so it is in $\mathcal{O}(p)$. Finally, the value of the two constants $\textsc{top}_L$ and $\textsc{bottom}_L$ is p respectively $\{\}$.

The $Join_L$ and $Meet_L$ operators which corresponds to set union and intersection operations are known to satisfy the axioms shown above.

Terms and Equations The syntax for (ground) terms and equations is defined below:

types

37.0 $Term = Join \mid Meet \mid C \mid \textsc{top} \mid \textsc{bottom};$

38.0 $Join :: \quad Term \times Term$;

39.0 $Meet :: \quad Term \times Term$;

40.0 $Eq :: \quad Term \times Term$ — term-equation

For terms to be of the same signature as the considered algebra $\mathcal{LA}(cset, p)$, we restrict term-constants $c : C$ to be in *cset*. When writing terms in examples we use the two infix operators $+$ and $*$ to represent *Join* and *Meet* respectively.

The Value of Terms in the Algebra $\mathcal{LA}(cset, p)$. Now given the algebra $\mathcal{LA}(cset, p)$ we define the value of terms in this algebra:

41.0 $Eval_L : PO \rightarrow Term \rightarrow CI\text{-set}$

.1 $Eval_L\,(p)(t) \triangleq$

.2 $\quad$ cases t :

.3 $\qquad mk\text{-}Join(t_1, t_2) \rightarrow Join_L(Eval_L(p)(t_1), Eval_L(p)(t_2)),$

.4 $\qquad mk\text{-}Meet(t_1, t_2) \rightarrow Meet_L(Eval_L(p)(t_1), Eval_L(p)(t_2)),$

.5 $\qquad (\text{TOP}) \rightarrow p,$

.6 $\qquad (\text{BOTTOM}) \rightarrow \{\},$

.7 $\qquad c \rightarrow cValue_L(p)(c)$

.8 $\quad$ end

The value of a term is a downset in $\mathcal{O}(p)$. The algebra $\mathcal{LA}(cset, p)$ is a generated algebra, i.e. every value in $\mathcal{O}(p)$ is the value of some term.

There exists a set of useful relationships between the value of a term and the underlying partial order as shown below:

Term Values: Let t be a term, p, p_1 and p_2 partial orders and cis a subset of p (i.e. $cis \subseteq p \subseteq \mathcal{P}(cset)$) then

42.0 $Eval_L(p)(t) \subseteq p$

.1 $Eval_L(p \setminus cis)(t) = Eval_L(p)(t) \setminus cis$

.2 $p_1 \subseteq p_2 \;\Rightarrow\; Eval_L(p_1)(t) = Eval_L(p_2)(t) \cap p_1$

.3 $Eval_L(p_1 \cup p_2)(t) = Eval_L(p_1)(t) \cup Eval_L(p_2)(t)$

.4 $Eval_L(p_1 \cap p_2)(t) = Eval_L(p_1)(t) \cap Eval_L(p_2)(t)$

.5 $p_1 \subseteq p_2 \;\Rightarrow\; Eval_L(p_1)(t) \subseteq Eval_L(p_2)(t)$

All properties in 42 can easily be proved by structural induction on the term structure. A proof of 42.1 is in section B.1 and a proof of 42.2 is in section B.2. [1]. The equation 42.5 follows trivially from 42.2. Similarly the equations 42.3 and 42.4 may be proved from 42.0 and 42.2 as shown in section B.3.

4.1 The Lattice of Anti-chains

From section 2.1 we know that a down-set cis has a unique anti-chain of which it is a down-set, namely $AntiCh(cis)$. So a downset cis may be represented by its unique anti-chain $AntiCh(cis)$. Consequently, given a lattice $\mathcal{O}(p)$ and the corresponding algebra $\mathcal{LA}(cset, p)$, we can easily define an isomorphic lattice where the elements are the anti-chain-part of the elements in $\mathcal{O}(p)$. We denote the set of new lattice elements $\mathcal{N}(p)$.

43.0 $\mathcal{N} : PO \rightarrow CI\text{-set-set}$

.1 $\mathcal{N}\,(p) \triangleq \{ac \mid ac : CI\text{-set} \cdot ac \subseteq p \wedge IsAntiChain(ac)\}$

Compare the above formula with 28. The corresponding algebra is

44.0 $\mathcal{NA}(cset, p) =< \mathcal{N}(p); Join_N, Meet_N, C_N(p)(cset), \text{TOP}_N, \text{BOTTOM}_N >$

where

[1] A proof for some of the other properties in the extended case including attribution may be found in 9

$$C_N(p)(cset) = \{cValue_N(p)(c) \mid c \in cset\}$$

The value of the two constants TOP_N and BOTTOM_N is $AntiCh(p)$ respectively $\{\}$. The new functions $Join_N$, $Meet_N$ and $cValue_N$ are defined below:

45.0 $Join_N : PO \rightarrow CI\text{-set} \times CI\text{-set} \rightarrow CI\text{-set}$

.1 $Join_N\ (p)(ac_1, ac_2) \triangleq AntiCh(DownSet(p)(ac_1) \cup DownSet(p)(ac_2))$

46.0 $Meet_N : PO \rightarrow CI\text{-set} \times CI\text{-set} \rightarrow CI\text{-set}$

.1 $Meet_N\ (p)(ac_1, ac_2) \triangleq AntiCh(DownSet(p)(ac_1) \cap DownSet(p)(ac_2))$

47.0 $cValue_N : PO \rightarrow C \rightarrow CI\text{-set}$

.1 $cValue_N\ (p)(c) \triangleq AntiCh(DownSetC(p)(\{mk\text{-}CI(\{c\})\}))$

48.0 $ISA_N : PO \rightarrow CI\text{-set} \times CI\text{-set} \rightarrow \mathbb{B}$

.1 $ISA_N\ (p)(ac_1, ac_2) \triangleq DownSet(p)(ac_1) \subseteq DownSet(p)(ac_2)$

The definitions of $Join_N$, $Meet_N$, $cValue_N$ and ISA_N above follow directly the definitions of $Join_L$, $Meet_L$, $cValue_L$ and ISA_L by converting between down-sets and anti-chains using *DownSet* and *AntiCh*.

Given this new definition of join, meet and concept constants, we can now define the value of terms in $\mathcal{NA}(cset, p)$:

49.0 $Eval_N : PO \rightarrow Term \rightarrow CI\text{-set}$

.1 $Eval_N\ (p)(t) \triangleq$

.2 cases t :

.3 $mk\text{-}Join(t_1, t_2) \rightarrow Join_N(p)(Eval_N(p)(t_1), Eval_N(p)(t_2))$,

.4 $mk\text{-}Meet(t_1, t_2) \rightarrow Meet_N(p)(Eval_N(p)(t_1), Eval_N(p)(t_2))$,

.5 $(\text{TOP}) \rightarrow AntiCh(p)$,

.6 $(\text{BOTTOM}) \rightarrow \{\}$,

.7 $c \rightarrow cValue_N(p)(c)$

.8 end

In an implementation of lattice algebras it will be an advantage to represent the lattice elements as anti-chains rather than down-sets because the down-sets usually will require considerable more space than the corresponding anti-chains. But if an implementation represents the elements as anti-chains it must also be able to compute the lattice operations efficiently. So rather than computing meet and join by converting between anti-chains and down-sets, we must find a way to compute meet and join directly as operations on anti-chains.

The join operation is easy to implement:

50.0 $Join_N : PO \rightarrow CI\text{-set} \times CI\text{-set} \rightarrow CI\text{-set}$

.1 $Join_N\ (p)(ac_1, ac_2) \triangleq AntiCh(ac_1 \cup ac_2)$

To implement the meet operation as an operation on anti-chains is more difficult. We need the concept of projection as defined in section 2.2. Having the projection function *CISproj* available we can implement $Meet_N$ as shown below:

51.0 $Meet_N : PO \rightarrow CI\text{-set} \times CI\text{-set} \rightarrow CI\text{-set}$

.1 $Meet_N\,(p)(ac_1, ac_2) \triangleq$

.2 $\quad \text{let } cis = \{mk\text{-}CI(cs_1 \cup cs_2) \mid mk\text{-}CI(cs_1) \in ac_1, mk\text{-}CI(cs_2) \in ac_2\} \text{ in}$

.3 $\quad\; CISproj(p)(cis)$

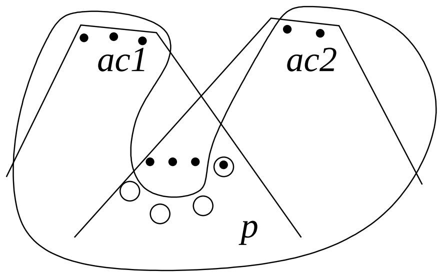


Figure 7: Implementation of meet as an operation on anti-chains

The implementation of the meet operation is illustrated in figure 7. The two definitions of $Meet_N$ in 46 and 51 both define an anti-chain in p. In section B.4 it is proved that they have the same downset in p, i.e.

52.0 $DownSet(AntiCh(DownSet(p)(ac_1) \cap DownSet(p)(ac_2))) =$

.1 $DownSet$

.2 $\quad (\text{let } cis = \{mk\text{-}CI(cs_1 \cup cs_2) \mid mk\text{-}CI(cs_1) \in ac_1, mk\text{-}CI(cs_2) \in ac_2\} \text{ in}$

.3 $\quad\; CISproj(p)(cis))$

Hence the two definitions define the same anti-chain in p.

The projection operation makes the meet operation less efficient than the join operation. The efficiency of the projection operation depends much on the actual implementation/representation of the partial order p.

53.0 $cValue_N : PO \rightarrow C \rightarrow CI\text{-set}$

.1 $cValue_N\,(p)(c) \triangleq CISproj(p)(\{mk\text{-}CI(\{c\})\})$

In the new definition of $cValue_N$ the value of $AntiCh(DownSetC(p)(\{mk\text{-}CI(\{c\})\}))$ (def. 47) is now computed more directly as the projection in p of $(mk\text{-}CI(\{c\}))$.

Finally, the ISA_N relation also has a more direct (and efficient) implementation.

54.0 $ISA_N : CI\text{-set} \times CI\text{-set} \rightarrow \mathbb{B}$

.1 $ISA_N\,(ac_1, ac_2) \triangleq \forall\, ci_1 \in ac_1 \cdot \exists\, ci_2 \in ac_2 \cdot ISA_P(ci_1, ci_2)$

5 Computing Lattices $\mathcal{O}(p)$ Satisfying a Set of Equations

We now consider how to compute a partial order $p \subseteq \mathcal{P}(cset)$ such that the lattice $\mathcal{O}(p)$ satisfies a given set of (ground) equations $eqs : Eq\text{-set}$ (besides the set of basic lattice equations). In the sequel we call a partial order in which a set of equations is satisfied a solution for the set of equations. Let

55.0 $IsEqsSol : Eq\text{-set} \to PO \to \mathbb{B}$

.1 $IsEqsSol\,(eqs)(p) \triangleq \forall\, mk\text{-}Eq(t_1, t_2) \in eqs \cdot Eval_L(p)(t_1) = Eval_L(p)(t_2)$

So we are looking for a way to compute the set of solutions:

56.0 $\{p \mid p \subseteq \mathcal{P}(cset) \cdot IsEqsSol(eqs)(p)\}$

If $n = \mathsf{card}\ (cset)$ then $\mathsf{card}\ (\mathcal{P}(cset)) = 2^n$-1 so there are $2^{2^n\text{-}1}$ candidate subsets of $\mathcal{P}(cset)$. Luckily, we do not have to test all the candidates.

Assume we have an equation $t_1 = t_2$. The property below concerns the relation between solutions for such an equation.

57.0 $\forall\, p_1, p_2 : PO, t_1, t_2 : Term \cdot$

.1 $\quad p_2 \subseteq p_1 \wedge Eval_L(p_1)(t_1) = Eval_L(p_1)(t_2) \;\Rightarrow\; Eval_L(p_2)(t_1) = Eval_L(p_2)(t_2)$

In words, if we have a solution p_1 to an equation $t_1 = t_2$ then every subset p_2 of that solution is also a solution. Property 57 may easily be proved from 42.1. Property 57 can easily be extended to a set of equations:

58.0 $\forall\, p_1, p_2 : PO, eqs : Eq\text{-set} \cdot$

.1 $\quad p_2 \subseteq p_1 \wedge IsEqsSol(eqs)(p_1) \;\Rightarrow\; IsEqsSol(eqs)(p_2)$

Consequently, if we compute the maximal partial order satisfying the set of equations, then all subsets of the maximal partial order are also solutions.

The relationships between the term value and the underlying partial order p shown in 42 provides material for two different ways to compute the set of concept-intersections corresponding to the maximal partial order p. Either one can start with the empty partial order and then *add* the concept-intersections which let the terms in an equation have equal values, or one can start with the power-set partial order $\mathcal{P}(cset)$ and then *subtract* the concept-intersections which make the terms in an equation unequal. Both methods relies on property 42.1.

5.1 The Additive Method

Let $t_1 = t_2$ be an equation, and let p_1 and p_2 be two partial orders in which the equation is satisfied, i.e.

59.0 $Eval_L(p_1)(t_1) = Eval_L(p_1)(t_2)$ and

.1 $Eval_L(p_2)(t_1) = Eval_L(p_2)(t_2)$

Using first 42.3 and then the equations in 59 we get

$$\begin{aligned}
&Eval_L(p_1 \cup p_2)(t_1) \\
&= Eval_L(p_1)(t_1) \cup Eval_L(p_2)(t_1) = Eval_L(p_1)(t_2) \cup Eval_L(p_2)(t_2) \\
&= Eval_L(p_1 \cup p_2)(t_2)
\end{aligned}$$

Hence

60.0 $Eval_L(p_1 \cup p_2)(t_1) = Eval_L(p_1 \cup p_2)(t_2)$

In words, if an equation is satisfied in two partial orders p_1 and p_2 it will also be satisfied in the union of these partial orders. This can easily be extended to a set of equations so 60 now becomes

61.0 $\forall\, eqs : Eq\text{-}\mathsf{set}, p_1, p_2 : PO \cdot$
.1 $IsEqsSol(eqs)(p_1) \wedge IsEqsSol(eqs)(p_2) \;\Rightarrow\; IsEqsSol(eqs)(p_1 \cup p_2)$

The property above shows us that if we have found two small solutions we can get a new bigger solution by making the union of the small solutions. So a strategy for finding a big solution might be to find many small solutions and making the union of these. But what are the smallest solutions? Property 58 shows that the smallest solutions are single concept-intersection partial orders (and the not so useful empty partial order.) Together 61 and 57 shows us that we can compute the maximal partial order in which a set of equations is satisfied by accumulating all the single concept-intersection partial orders in which the equations are satisfied:

62.0 $MaxPO : C\text{-}\mathsf{set} \to Eq\text{-}\mathsf{set} \to PO$
.1 $MaxPO\,(cset)(eqs)) \triangleq \{ci \mid ci \in \mathcal{P}(cset) \cdot IsEqsSol(eqs)(\{mk\text{-}CI(ci)\})\}$

So if $n = \mathsf{card}\,(cset)$ then $pmax = MaxPO(cset)(eqs)$ is computed by testing the set of equations with all 2^n-1 concept-intersections.

5.2 The Subtractive Method

Let $t_1 = t_2$ be an equation, let $p_c = \mathcal{P}(cset)$ and let

$$cis_1 = Eval_L(p_c)(t_1) \text{ and } cis_2 = Eval_L(p_c)(t_2)$$

then $eqrej = (cis_1 \cup cis_2) \setminus (cis_1 \cap cis_2)$ is the set of all the concept-intersections not occurring in both cis_1 and cis_2, i.e. the set of concept-intersections causing t_1 and t_2 to evaluate to different values.

Using 42.1 we get

63.0 $Eval_L(p_c \setminus eqrej)(t_1) = Eval_L(p_c)(t_1) \setminus eqrej = cis_1 \cap cis_2$ and
.1 $Eval_L(p_c \setminus eqrej)(t_2) = Eval_L(p_c)(t_2) \setminus eqrej = cis_1 \cap cis_2$

so if $p' = p_c \setminus eqrej$ then $Eval_L(p')(t_1) = Eval_L(p')(t_2)$.

From 57 we know that if p_1 is a solution to an equation $t_1 = t_2$ then every subset p_2 of that solution is also a solution. Hence, in 63 above, subtracting any superset of $eqrej$ results in a partial order in which the equation is satisfied. So having a set of equations rather than just a single equation we compute the $eqrej$ set for each equation and subtract the union of these from $\mathcal{P}(cset)$:

64.0 $EqRej : CI\textsf{-set} \rightarrow Eq \rightarrow CI\textsf{-set}$
.1 $EqRej\ (p)(mk\text{-}Eq(t_1, t_2)) \triangleq$
.2 $\quad \textsf{let}\ cis_1 = Eval_L(p)(t_1),$
.3 $\qquad cis_2 = Eval_L(p)(t_2)\ \textsf{in}$
.4 $\quad (cis_1 \cup cis_2) \setminus (cis_1 \cap cis_2)$

65.0 $MaxPO : C\textsf{-set} \rightarrow Eq\textsf{-set} \rightarrow PO$
.1 $MaxPO\ (cset)(eqs) \triangleq$
.2 $\quad \textsf{let}\ p_c = \mathcal{P}(cset)\ \textsf{in}$
.3 $\quad \textsf{let}\ rejected = \bigcup \{EqRej(p_c)(eq) \mid eq \in eqs\}\ \textsf{in}$
.4 $\quad p_c \setminus rejected$

$EqRej(p)(mk\text{-}Eq(t_1, t_2))$ (64) computes the set of concept-intersections in p that causes t_1 and t_2 to evaluate to different values. $MaxPO(cset)(eqs)$ (65) first computes the powerset partial order (containing the 2^n-1 different concept-intersections, if n=card (cset)). Next it removes from the powerset partial order the rejected concept-intersections, i.e. the concept-intersections that causes an equation not to be satisfied.

Example As an example of using the subtractive method, consider the equations

$$b = b * a$$
$$c = c * a$$

We evaluate the terms in $\mathcal{P}(\{a, b, c\})$:

equation	left term	right term	reject
$b = b * a$	$\{b, ab, bc, abc\}$	$\{b, ab, bc, abc\} \cap \{a, ab, ac, abc\} = \{ab, abc\}$	$\{b, bc\}$
$c = c * a$	$\{c, ac, bc, abc\}$	$\{c, ac, bc, abc\} \cap \{a, ab, ac, abc\} = \{ac, abc\}$	$\{c, bc\}$
all eqs			$\{b, c, bc\}$

The biggest subset of $\mathcal{P}(\{a, b, c\})$ which causes no conflicts in the equations is

$$pmax = \mathcal{P}(\{a, b, c\}) \setminus \{b, c, bc\} = \{a, ab, ac, abc\}$$

The Hasse diagram for this partial order and the corresponding lattice $\mathcal{O}(pmax)$ is shown in figure 6.

5.3 The Lattices Satisfying a Set of Equations

The two functions: *MaxPO* defined in 62 and *MaxPO* defined in 65 computes one and the same greatest subset of $\mathcal{P}(cset)$ which is a solution for the given set of equations. This should be rather obvious, but we do not give a formal proof. So the two definitions actually define the same function. In the sequel we just refer to *MaxPO* without worrying about how it is implemented.

Let $pmax = MaxPO(cset)(eqs)$. We define the class of lattices

66.0 $C_{eqs} = \{\mathcal{LA}(cset, p) \mid p : PO \cdot p \subseteq pmax\}$

According to 57 all lattices in C_{eqs} satisfies the given set of equations. Among the lattices in C_{eqs} we consider two with interesting properties. The initial lattice is theoretical interesting and is described below. The most disjoint lattice is a lattice that can be constructed by efficient algorithms. The lattice is described in section 6. An efficient implementation of the most disjoint lattice is described in section 7.

5.4 The Initial Lattice

In the class C_{eqs} of lattices defined above $\mathcal{LA}(cset, pmax)$ is the initial algebra/lattice or most general lattice. This is equivalent to saying that for each $p \subseteq pmax$ there is a unique homomorphism from $\mathcal{LA}(cset, pmax)$ to $\mathcal{LA}(cset, p)$. For a given $p \subseteq pmax$, that homomorphism is defined by the function $h : \mathcal{O}(pmax) \to \mathcal{O}(p)$ such that

$$h(cis) = cis \cap p, cis \in \mathcal{O}(pmax)$$

Now

$$h(Join_L(cis1, cis2)) = (cis1 \cup cis2) \cap p = (cis1 \cap p) \cup (cis2 \cap p) = Join_L(h(cis1), h(cis2))$$
$$h(Meet_L(cis1, cis2)) = (cis1 \cap cis2) \cap p = (cis1 \cap p) \cap (cis2 \cap p) = Meet_L(h(cis1), h(cis2))$$

According to the definition of constants (41.7, 36) we have

$$\begin{aligned} & h(c_{pmax}()) = cValue_L(pmax)(c) \cap p = DownSet(pmax)(\{mk\text{-}CI(\{c\})\}) \cap p \\ & = \{mk\text{-}CI(cs) \mid mk\text{-}CI(cs) \in pmax \cdot c \in cs\} \cap p \\ & = \{mk\text{-}CI(cs) \mid mk\text{-}CI(cs) \in pmax \cap p \cdot c \in cs\} \\ & = cValue_L(p)(c) = c_p() \end{aligned}$$

Similar for the constants TOP and BOTTOM. Consequently h is a homomorphism.

6 The Most Disjoint Lattice

The initial lattice has elements corresponding to all possible concept-intersections not excluded by the given set of equations. In practice, when specifying a concept hierarchy, this is not always what is wanted. Often we only want to see lattice points that are ”relevant” in some way. For instance, if the lattice is constructed as part of an ontology/taxonomy one might have available a set of terms denoting interesting points in the lattice, points to which some information should be associated. From this point of view we might consider the lattice as a lattice structured database containing concept-intersections corresponding to some *inserted* terms. These inserted terms will be specified by the user of such a database. In some situations these inserted terms would include the terms from the given set of equations. This database point of view was illustrated in section 1.

So, having constructed the maximal partial order *pmax* satisfying the given set of equations *eqs*, we want to find a subset of *pmax* only containing the concept-intersections which are made “relevant” by a given set of inserted terms. Relevance should not be a function of the terms actual *form* but rather a function of the terms actual *value* in *pmax*. For example, $a * (b + c)$, $a * b + a * c$ and $a * (b + c) + a * b * c$ should all result in the same set of relevant concept-intersections.

When concept-intersections are removed from *pmax* some terms will evaluate to a new value and some terms will keep their value. The more concept-intersections that are removed the more terms will have their value changed. When concept-intersections are removed the concepts in the lattice have fewer overlaps, i.e. the concepts become more disjoint. The idea with the most disjoint lattice is to remove as many concept-intersections as possible without changing the values of the given set of inserted terms. That is why the lattice is called the *most* disjoint lattice with respect to the given set of inserted terms.

From the database point of view the class of lattices C_{eqs} (66) is the set of all possible database instances. Each lattice represents a different set of inserted terms. If we insert a new term in a lattice we get a new lattice (provided the term denotes new concept-intersections).

As shown in section 4 (and 4.1) we have two evaluation functions $Eval_L$ and $Eval_N$, where $Eval_N(p)(t)$ is the anti-chain part of $Eval_L(p)(t)$ and thus usually contains fewer concept-intersections than $Eval_L(p)(t)$. The definition of the most disjoint lattice is based on $Eval_N$ so we must study the term-value preserving properties of $Eval_N$. However, as a warm-up exercise we first study the similar properties for $Eval_L$.

6.1 Term Value Preserving Properties for $Eval_L$

Given a set of concepts *cset* we consider the evaluation of a term t in an arbitrary subset partial order $pm \subseteq \mathcal{P}(cset)$. The partial orders and sets used in this section are illustrated in figure 8. The value of the term t in the partial order $\mathcal{P}(cset)$ is the downset indicated by the straight lines and the horizontal line indicates the anti-chain of this downset. The value of the term t in the partial order pm is the subset pt of this downset which is in pm; it is hatched with dotted lines.

Lemma

67.0 $\forall\, t : Term, pm : PO \cdot$

.1 $\quad$ let $pt = Eval_L(pm)(t)$ in

.2 $\quad Eval_L(pt)(t) = Eval_L(pm)(t)$

Evaluating a term t in a partial order pm and in the subset partial order pt, which is the value of the term t in pm, yields the same value.

Proof: Let

$$dp = pm \setminus pt$$

hence dp and pt are disjoint:

$$dp \cap pt = \{\}$$

and

$$pt = pm \setminus dp$$

We now have

$$Eval_L(pt)(t) = Eval_L(pm \setminus dp)(t) \qquad \text{from } 42.1$$
$$= Eval_L(pm)(t) \setminus dp$$
$$= Eval_L(pm)(t)$$

because $Eval_L(pm)(t) = pt$ and $dp \cap pt = \{\}$

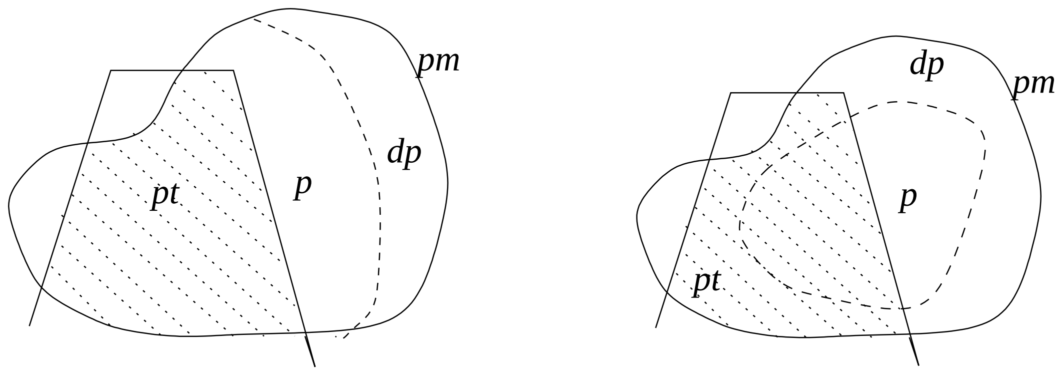


Figure 8: lemma 67, 68 and 70

Lemma

68.0 $\forall\, t : \mathit{Term}, pm : \mathit{CI}\text{-set} \cdot$
.1 $\quad$ let $pt = \mathit{Eval}_L(pm)(t)$ in
.2 $\quad$ $\forall\, p : PO \cdot pt \subseteq p \wedge p \subseteq pm \;\Rightarrow\; \mathit{Eval}_L(p)(t) = \mathit{Eval}_L(pm)(t)$

Evaluating a term t in a partial order pm and in any subset partial order p, which includes the value of the term t in pm, yields the same value. This is a generalization of 67.

Proof: Assume the left hand side of the implication above

69.0 $pt \subseteq p \wedge p \subseteq pm$

We then have the following equalities

$$\mathit{Eval}_L(p)(t) = \mathit{Eval}_L(pm)(t) = pt \cap p \qquad \text{from } 42.2$$
$$= pt = \mathit{Eval}_L(pm)(t) \text{ (from 69 and 68.1)}$$

Lemma

70.0 $\forall\, t : \mathit{Term}, pm : \mathit{CI}\text{-set} \cdot$
.1 $\quad$ let $pt = \mathit{Eval}_L(pm)(t)$ in
.2 $\quad$ $\forall\, p : PO \cdot \neg\, pt \subseteq p \wedge p \subseteq pm \;\Rightarrow\; \mathit{Eval}_L(p)(t) \neq \mathit{Eval}_L(pm)(t)$

Evaluating a term t in a partial order pm and in any subset partial order p, which does not include the value of the term t in pm yields different values.

Proof: Assume the left hand side of the implication above

$$\neg\, pt \subseteq p$$

hence using 70.1 we get

$$\neg\, \mathit{Eval}_L(pm)(t) \subseteq p$$

so

$$\mathit{Eval}_L(pm)(t) \cap p \neq \mathit{Eval}_L(pm)(t)$$

Using this inequality we get

$$\mathit{Eval}_L(p)(t)$$

from 42.2

$= Eval_L(pm)(t) \cap p \neq Eval_L(pm)(t)$

The properties in lemma 68 and 70 can now be combined in the following theorem:

Theorem

71.0 $\forall\, t : Term, pm : CI\text{-set} \cdot$

.1 $\quad$ let $pt = Eval_L(pm)(t)$ in

.2 $\quad \forall\, p : PO \cdot p \subseteq pm \;\Rightarrow\; (pt \subseteq p \;\Leftrightarrow\; Eval_L(p)(t) = Eval_L(pm)(t))$

The proof follows directly from the lemmas 67, 68, and 70.

The theorem above shows that if we want to have the term t evaluated to the same value as in the given partial order pm, then we can use exactly all the subsets of pm containing the value of the term in pm.

Now, what if we want to preserve the value of *two* terms t_1 and t_2? For t_1 we can use all the partial orders between $Eval_L(pm)(t_1)$ and pm, and for t_2 we can use all the partial orders between $Eval_L(pm)(t_2)$ and pm. Consequently, to keep the value of both t_1 and t_2 we can use all the partial orders between $Eval_L(pm)(t_1) \cup Eval_L(pm)(t_2)$ and pm. The next theorem generalizes this to an arbitrary set of terms:

Theorem

72.0 $\forall\, tset : Term\text{-set}, pm : CI\text{-set} \cdot$

.1 $\quad$ let $pts = \bigcup\{Eval_L(pm)(t) \mid t \in tset\}$ in

.2 $\quad \forall\, p : PO \cdot p \subseteq pm \;\Rightarrow\; (pts \subseteq p \;\Leftrightarrow\; (\forall\, t \in tset \cdot Eval_L(p)(t) = Eval_L(pm)(t)))$

Proof: Assume the left hand side of the implication above

73.0 $p \subseteq pm$

Now, to prove the right hand side equivalence, we prove the left to right and right to left implications individually.

LEFT TO RIGHT: So we first assume the left hand side

74.0 $pts \subseteq p$

Next, let t be an arbitrary term in *tset*:

$$t \in tset$$

According to the theorem 71 we then have

$$Eval_L(pm)(t) \subseteq p \;\Leftrightarrow\; Eval_L(p)(t) = Eval_L(pm)(t)$$

From 74 and the definition of *pts* the left hand side above is true and consequently also the right hand side:

$$Eval_L(p)(t) = Eval_L(pm)(t)$$

RIGHT TO LEFT: Next, we must prove the right to left implication in the equivalence:

$$(\forall\, t \in tset \cdot Eval_L(p)(t) = Eval_L(pm)(t)) \;\Rightarrow\; pts \subseteq p$$

so assume

$$\forall\, t \in tset \cdot Eval_L(p)(t) = Eval_L(pm)(t)$$

From 73 and theorem 42.0 this may be transformed to

$$\forall\, t \in tset \cdot Eval_L(pm)(t) \subseteq p$$

So

$\bigcup\{Eval_L(pm)(t) \mid t \in tset\} \subseteq p$

which is equivalent to

$pts \subseteq p$

6.2 Term Value Preserving Properties for $Eval_N$

In this section we show that $Eval_N$ has properties similar to the properties for $Eval_L$ proved in section 6.1. The partial orders and sets used in this section is illustrated in figure 9. Because we now use $Eval_N$ which yields an anti-chain the figure now also shows the anti-chain ptn of the term value pt. This subset of pt is indicated by the dashed line in the top-part of the down-set pt. Below the value of $Eval_N(p)(t)$ is called a normal form value.

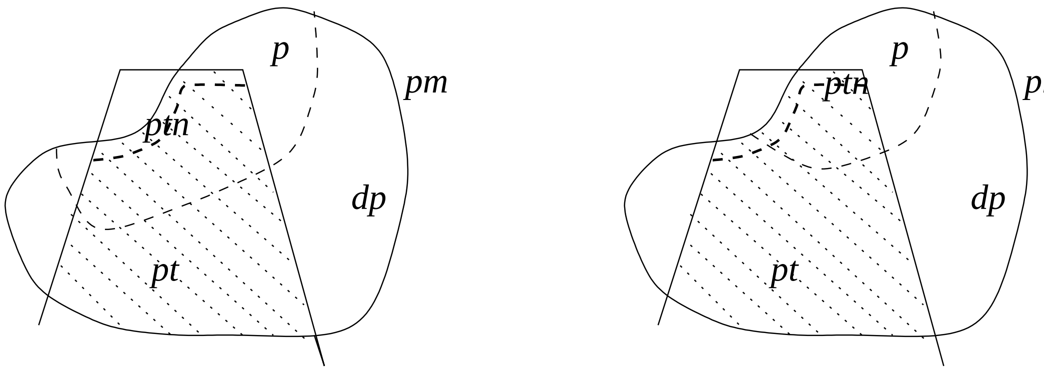


Figure 9: lemma 75 and 79

Lemma

75.0 $\forall\, t : Term, pm : PO \cdot$

.1 $\quad$ let $ptn = Eval_N(pm)(t)$ in

.2 $\quad \forall\, p : PO \cdot ptn \subseteq p \wedge p \subseteq pm \;\Rightarrow\; Eval_N(p)(t) = Eval_N(pm)(t) = Eval_L(ptn)(t)$

Evaluating a term t (using $Eval_N$) in a partial order pm and in any subset partial order p, which includes the normal form value ptn of the term t in pm, yields the same normal form value.

Proof: Assume the left hand side

76.0 $ptn \subseteq p \wedge p \subseteq pm$

Using term-value properties 42 and the right conjunct in 76 gives

77.0 $Eval_L(p)(t) \subseteq Eval_L(pm)(t)$

.1 $Eval_L(p)(t) = Eval_L(pm)(t) \cap p$

From 75.1 we now get

$ptn = Eval_N(pm)(t) = AntiCh(Eval_L(pm)(t)) \subseteq Eval_L(pm)(t)$

because an anti-chain of a set ds is a subset of ds. Combining this subset inclusion with the left conjunct in 76 gives

$$\begin{aligned} & ptn \subseteq Eval_L(pm)(t) \cap p \\ & = Eval_L(p)(t) \end{aligned} \qquad \text{from 77.1}$$

Again, combining this subset inclusion with the one in 77.0 gives

$$ptn = AntiCh(Eval_L(pm)(t)) \subseteq Eval_L(p)(t) \subseteq Eval_L(pm)(t)$$

If we use the anti-chain property 21 to the above subset inclusion of an antichain we get

$$AntiCh(Eval_L(p)(t)) = AntiCh(Eval_L(pm)(t))$$

which is equivalent to

78.0 $\quad Eval_N(p)(t) = Eval_N(pm)(t)$

Next we prove the equality to $Eval_L(ptn)(t)$. From 42.0 we get

$$Eval_L(ptn)(t) \subseteq ptn$$

ptn is an anti-chain and so are all of its subsets, so $Eval_L(ptn)(t)$ is an anti-chain. From the anti-chain property 20 we then get

$$Eval_N(ptn)(t) = AntiCh(Eval_L(ptn)(t)) = Eval_L(ptn)(t)$$

Finally from 78 for $p = ptn$ and the equality above we get

$$Eval_N(pm)(t) = Eval_N(ptn)(t) = Eval_L(ptn)(t)$$

which together with the equality in 78 gives the equalities in 75.2.

Lemma

79.0 $\quad \forall\, t : Term, pm : PO \cdot$

.1 $\quad\quad$ let $ptn = Eval_N(pm)(t)$ in

.2 $\quad\quad \forall\, p : PO \cdot \neg\, ptn \subseteq p \;\Rightarrow\; Eval_N(p)(t) \neq Eval_N(pm)(t)$

Evaluating a term t (using $Eval_N$) in a partial order pm and in any subset partial order p, which does not include the normal form value of the term t in pm yields different normal form values.

Proof: Assume the left hand side of the implication:

$$\neg\, ptn \subseteq p$$

so ptn has a non-empty subset not in p:

$$\{\} \subset ptn \setminus p \subseteq ptn = Eval_N(pm)(t)$$

So $Eval_N(pm)(t)$ has a nonempty subset, which is not in p. But for $Eval_N(p)(t)$ we have

$$Eval_N(p)(t) \subseteq Eval_L(p)(t) \subseteq p$$

Hence

$$Eval_N(pm)(t) \neq Eval_N(p)(t)$$

The properties in lemma 75 and 79 can now be combined in the following theorem:

Theorem

80.0 $\forall\, t : \mathit{Term}, pm : PO \cdot$
.1 $\quad$ let $ptn = \mathit{Eval}_N(pm)(t)$ in
.2 $\quad \forall\, p : PO \cdot p \subseteq pm \;\Rightarrow\; (ptn \subseteq p \;\Leftrightarrow\; \mathit{Eval}_N(p)(t) = \mathit{Eval}_N(pm)(t))$

The proof follows directly from the lemmas 75, and 79. The theorem above shows that if we want to have the term t evaluated to the same normal form value as in the given partial order pm, then we can use exactly all the subsets of pm containing the normal form value of the term in pm.

The next theorem corresponds to theorem 72 for Eval_L. So the first question is what to do if we want to preserve the value of *two* terms t_1 and t_2? For t_1 we can use all the partial orders between $\mathit{Eval}_N(pm)(t_1)$ and pm, and for t_2 we can use all the partial orders between $\mathit{Eval}_N(pm)(t_2)$ and pm. Consequently, to keep the value of both t_1 and t_2 we can use all the partial orders between $\mathit{Eval}_N(pm)(t_1) \cup \mathit{Eval}_N(pm)(t_2)$ and pm. The next theorem generalizes this to an arbitrary set of terms: Given a *set* of terms and a partial order pm. In which sub partial orders will all the given terms have the same normal form value as in pm?

Theorem

81.0 $\forall\, tset : \mathit{Term}\text{-set}, pm : PO \cdot$
.1 $\quad$ let $pts = \bigcup\{\mathit{Eval}_N(pm)(t) \mid t \in tset\}$ in
.2 $\quad \forall\, p : PO \cdot p \subseteq pm \;\Rightarrow\; (pts \subseteq p \;\Leftrightarrow\; \forall\, t \in tset \cdot \mathit{Eval}_N(p)(t) = \mathit{Eval}_N(pm)(t))$

Proof: Assume the left hand side of the implication above:

82.0 $p \subseteq pm$

Now, to prove the right hand side equivalence, we prove the left to right and right to left implications individually.
LEFT TO RIGHT: So we first assume the left hand side

83.0 $pts \subseteq p$

Next, let t be an arbitrary term in $tset$:

$$t \in tset$$

According to theorem 80 we then have

$$\mathit{Eval}_N(pm)(t) \subseteq p \;\Leftrightarrow\; \mathit{Eval}_N(p)(t) = \mathit{Eval}_N(pm)(t)$$

From 83 and the definition of pts the left hand side above is true and consequently also the right hand side:

$$\mathit{Eval}_N(p)(t) = \mathit{Eval}_N(pm)(t)$$

RIGHT TO LEFT: Next, we must prove the right to left implication in the equivalence:

$$(\forall\, t \in tset \cdot \mathit{Eval}_N(p)(t) = \mathit{Eval}_N(pm)(t)) \;\Rightarrow\; pts \subseteq p$$

so assume

$$\forall\, t \in tset \cdot \mathit{Eval}_N(p)(t) = \mathit{Eval}_N(pm)(t)$$

From 82 and theorem 80 this may be transformed to

$$\forall\, t \in tset \cdot \mathit{Eval}_N(pm)(t) \subseteq p$$

So

$$\bigcup\{Eval_N(pm)(t) \mid t \in tset\} \subseteq p$$

which is equivalente to

$$pts \subseteq p$$

6.3 The Most Disjoint Lattice and its Properties

Given a set of concepts *cset* and a set of user-specified inserted terms *insterms*, the function below now finds the partial order for the lattice which we call the most disjoint lattice with respect to the given set of terms.

84.0 $TMdisjPO : Cset \to Eq\text{-set} \to Term\text{-set} \to PO$

.1 $TMdisjPO\,(cset)(eqs)(insterms) \triangleq$

.2 $\quad \textsf{let}\ pmax = MaxPO(cset)(eqs)\ \textsf{in}$

.3 $\quad\quad \bigcup\{Eval_N(pmax)(t) \mid t \in insterms\}$

In 84.2 *pmax* is the maximal partial order satisfying the given set of equations *eqs*. In line 84.3 the most disjoint lattice is defined to be the set of concept-intersections which is the union of the normal form value of all the inserted terms in *pmax*.

From theorem 81 and the definition of the most disjoint lattice above we can easily derive the following property for most disjoint lattices:

Term-value Preserving Property of the Most Disjoint Lattice Let *cset* be a set of concepts, *eqs* a set of equations about these concepts and *insterms* a set of user specified inserted terms.

85.0 $\textsf{let}\ pmax = MaxPO(cset)(eqs),$

.1 $\quad pmdsj = TMdisjPO(cset)(eqs)(insterms)\ \textsf{in}$

.2 $\forall\, p : PO \cdot p \subseteq pmax \;\Rightarrow$

.3 $\quad (pmdsj \subseteq p \;\Leftrightarrow\; \forall\, t \in insterms \cdot Eval_N(p)(t) = Eval_N(pmax)(t))$

In the most disjoint lattice $\mathcal{NA}(cset, pmdsj)$ all the inserted terms evaluate to the same normal form value as they do in the initial lattice $\mathcal{NA}(cset, pmax)$. Furthermore, the most disjoint lattice is the smallest lattice having this property in the sense that all lattices based on a partial order not containing *pmdsj* do *not* have this property.

If we have a system that— from a set of equations — implements the most disjoint lattice rather then the initial lattice, the question naturally arises if there are lattices which cannot be constructed. Luckily, such a system can construct all the lattices that an initial lattice based system can do, even with the same set of equations:

Power of Most Disjoint Lattice Below *pterms* is a (huge) set of terms, which evaluates to all possible set of concept-intersections. We have

86.0 $\textsf{let}\ pterms\ \textsf{be st}\ \bigcup\{AntiCh(Eval_L(\mathcal{P}(cset))(t)) \mid t \in pterms\} = \mathcal{P}(cset)\ \textsf{in}$

.1 $TMdisjPO(cset)(eqs)(pterms) = MaxPO(cset)(eqs)$

So in the most disjoint lattice system we must supply the system with a set of inserted terms consisting of op to 2^n-1 product terms (where $n = \mathsf{card}\ (cset)$) in order to get all the overlaps in the initial lattice. On the other hand, in an initial lattice based system one must supply the system with up to 2^n-1 equations of the form $c_1 * c_2 * \ldots * c_n = \textsc{bottom}$ to get the most disjoint lattice.

Incremental Construction. From the definition of *TMdisjPO* (84) we have

87.0 $\mathit{TMdisjPO}(cset)(eqs)(insterms1 \cup insterms2) =$
.1 $\mathit{TMdisjPO}(cset)(eqs)(insterms1) \cup \mathit{TMdisjPO}(cset)(eqs)(insterms2)$

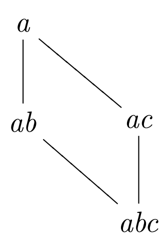
Each inserted terms contribution to the set of concept-intersections in the partial order is independent of the already inserted terms, so the partial order for the most disjoint lattice may be constructed incrementally by inserting the terms one after the other.

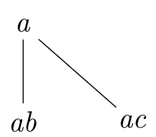
In section 7 we show how to make an efficient implementation of the most disjoint lattice. Section 6.4 shows 4 examples of the partial order *pmdsj* for the most disjoint lattice and the corresponding partial order *pmax*.

6.4 Examples of Most Disjoint Lattices

Below we first consider three examples with $cset = \{a, b, c\}$ so $\mathcal{P}(cset)$ is as shown in the right part of figure 1. We use *pmax* for $\mathit{MaxPO}(cset)(eqs)$ and *pmdsj* for $\mathit{TMdisjPO}(cset)(eqs)(insterms)$.

Example 1 Let $eqs = \{b = b * a, c = c * a\}$, see the example page 21. As inserted terms we use $insterms = \{a, b, c\}$. The corresponding partial orders *pmax* and *pmdsj* are shown below:

$pmax =$ 

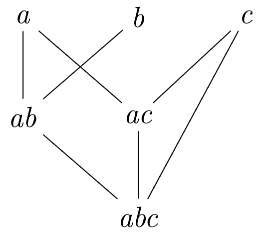
$pmdsj =$ 

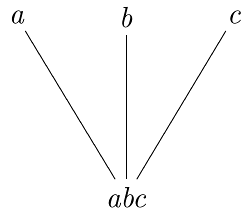
The partial order *pmdsj* is computed as follows:

t	$tval = Eval_L(\mathcal{P}(cset))(t)$	$Eval_L(pmax)(t)$	$Eval_N(pmax)(t)$
a	a, ab, ac, abc	a, ab, ac, abc	a
b	b, ab, bc, abc	ab, abc	ab
c	c, ac, bc, abc	ac, abc	ac
		$pmdsj =$	a, ab, ac

□

Example 2 Let $eqs = \{a = a + b * c\}$ and $insterms = \{a, b, c, b * c\}$. When computing $MaxPO(cset)(eqs)$ (def. 65) we get $rejected = \{bc\}$. The corresponding partial orders $pmax$ and $pmdsj$ are shown below:

$pmax =$ 

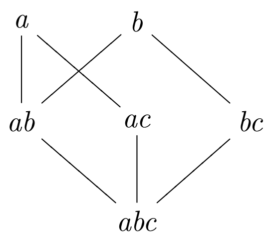
$pmdsj =$ 

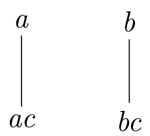
The partial order $pmdsj$ is computed as follows:

t	$tval = Eval_L(\mathcal{P}(cset))(t)$	$Eval_L(pmax)(t)$	$Eval_N(pmax)(t)$
a	a, ab, ac, abc	a, ab, ac, abc	a
b	b, ab, bc, abc	b, ab, abc	b
c	c, ac, bc, abc	c, ac, abc	c
$b * c$	bc, abc	abc	abc
		$pmdsj =$	a, b, c, abc

□

Example 3 Let $eqs = \{c = c * (a + b)\}$ and $insterms = \{a, b, c\}$. When computing $MaxPO(cset)(eqs)$ (def. 65) we get $rejected = \{c\}$, i.e. c does not have its own existence (but has "sunk" down into a and b). The corresponding partial orders $pmax$ and $pmdsj$ are shown below:

$pmax =$ 

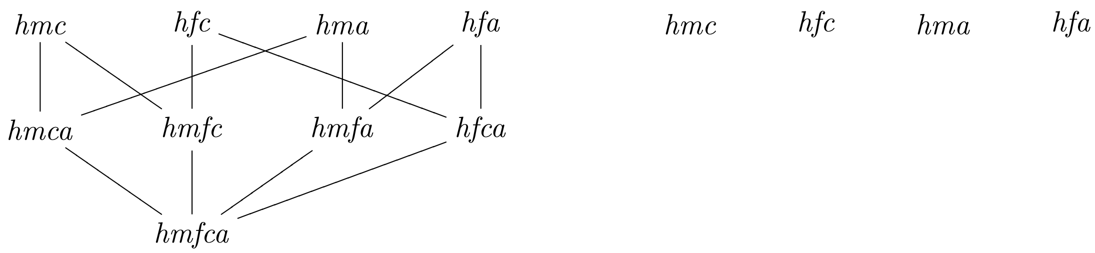
$pmdsj =$ 

The partial order $pmdsj$ is computed as follows:

t	$tval = Eval_L(\mathcal{P}(cset))(t)$	$tac = AntiCh(tval)$	$TermRel(cset)(pmax)(t)$
a	a, ab, ac, abc	a, ab, ac, abc	a
b	b, ab, bc, abc	b, ab, bc, abc	b
c	c, ac, bc, abc	ac, bc, abc	ac, bc
		$pmdsj =$	a, b, ac, bc

□

Example 4 In this example we have $cset = \{h, m, f, c, a\}$ (for *human*, *male*, *female*, *child*, *adult*), $insterms = \{m, f, c, a\}$ and the equations $h = m + f, h = c + a$. A more extensive computation yields the partial orders *pmax* and *pmdsj* shown below to the left and right respectively :



Here the concept-intersections *hmc*, *hfc*, *hma* and *hfa* corresponds to the concepts boy, girl, man and woman. In the most disjoint partial order *pmdsj*, all concept-intersections containing *mf* and *ca* vanishes, i.e. the concepts m and f become disjoint and similar for c and a. The remaining concept-intersections are not related so $\mathcal{O}(pmdsj)$ becomes a powerset lattice with the four concept-intersections as atoms. □

7 An Efficient Implementation of the Most Disjoint Lattice

In this section we consider how to make an efficient implementation of the most disjoint lattice as defined in 84 section 6.3. The definition is repeated below:

88.0 $\quad TMdisjPO : Cset \to Eq\text{-}\mathsf{set} \to Term\text{-}\mathsf{set} \to PO$

.1 $\quad TMdisjPO\,(cset)(eqs)(insterms) \triangleq$

.2 $\quad\quad \mathsf{let}\ pmax = MaxPO(cset)(eqs)\ \mathsf{in}$

.3 $\quad\quad \bigcup\{Eval_N(pmax)(t) \mid t \in insterms\}$

The definition first defines the partial order *pmax* for the initial lattice and then evaluates the inserted terms in *pmax*. But the size of *pmax* may grow exponentially with the number of concepts and so also the number of computation steps. However, the resulting partial order computed in 88.3 usually contains a considerable smaller number of concept-intersections. So we must avoid the explicit computation of *pmax* and try to find a way to compute

$$Eval_N(pmax)(t)$$

for each inserted term t directly from the equations *eqs*, without having *pmax* available.

7.1 Term Evaluation using Projection

In the method shown in this section the term t is first evaluated in the power-set partial order using $Eval_N$ and the resulting anti-chain is then projected down in *pmax* using a projection function similar to the projection function defined in 26 in section 4.1. We have

Lemma

89.0 $\quad Eval_N(pmax)(t) = CISproj(pmax)(Eval_N(\mathcal{P}(cset))(t))$

Proof: The equation above is between two anti-chains, $ac1$ and $ac2$. We show that they are both anti-chains in $pmax$ with the same downset in $pmax$, so they are the same anti-chain. According to the definition of $Eval_N$, the left-hand anti-chain $ac1$ is the anti-chain in $pmax$ such that

$$DownSet(pmax)(ac1) = Eval_L(pmax)(t)$$

From the projection properties used to define $CISproj$ (26) we know that the right-hand side anti-chain $ac2$ is an anti-chain in $pmax$. For the downset of $ac2$ in $pmax$ we have the following sequence of equalities:

$$\begin{aligned}
& DownSetC(pmax)(ac2) \\
& \qquad\qquad\qquad\qquad \text{from } 26.3 \\
&= DownSetC(pmax)(Eval_N(\mathcal{P}(cset))(t)) \\
& \qquad\qquad\qquad\qquad \text{from } 14 \\
&= DownSetC(\mathcal{P}(cset))(Eval_N(\mathcal{P}(cset))(t)) \cap pmax \\
& \qquad\qquad \text{according to the definition of } Eval_N \text{ in sect. } 4.1 \\
&= Eval_L(\mathcal{P}(cset))(t) \cap pmax \\
& \qquad\qquad\qquad\qquad \text{from } 42.2 \\
&= Eval_L(pmax)(t)
\end{aligned}$$

Consequently we have $DownSetC(pmax)(ac1) = DownSetC(pmax)(ac2)$. □

The method based on equation 89 above is sketched in figure 10. The value of the term t in the partial order $\mathcal{P}(cset)$ is the downset indicated by the straight lines and the black bullets at the top of this area indicates the elements in the anti-chain of this down-set, i.e. the elements in $Eval_N(\mathcal{P}(cset))(t)$. The solid lined circles along the dotted line in $pmax$ indicates the anti-chain which is the resulting projection into $pmax$, i.e. the elements in $Eval_N(pmax)(t)$.

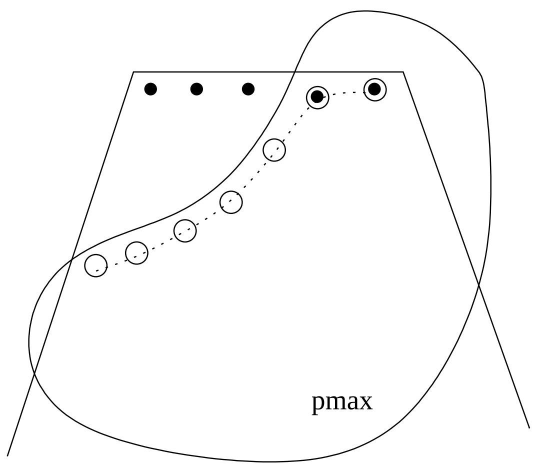


Figure 10: Projecting $Eval_N(\mathcal{P}(cset)(t))$ into $pmax$

Following this scheme we must have an efficient way to compute $Eval_N(\mathcal{P}(cset))(t)$ and next we must implement a special projection function which projects into the partial order $pmax$ given the equations.

7.2 Evaluation in the Power Set Partial Order

We can easily make a specialized and efficient version of $Eval_N$ which evaluates in the power-set partial order. Below we define the specialized function $Eval_{NPc}$ such that

90.0 $Eval_{NPc}(cset)(t) = Eval_N(\mathcal{P}(cset))(t)$

91.0 $Eval_{NPc} : Cset \to Term \to CI\text{-set}$
.1 $Eval_{NPc}\,(cset)(t) \triangleq$
.2 cases t :
.3 $mk\text{-}Join(t_1, t_2) \to Join_{NPc}(Eval_{NPc}(cset)(t_1), Eval_{NPc}(cset)(t_2)),$
.4 $mk\text{-}Meet(t_1, t_2) \to Meet_{NPc}(Eval_{NPc}(cset)(t_1), Eval_{NPc}(cset)(t_2)),$
.5 $(\text{TOP}) \to \{mk\text{-}CI(\{c\}) \mid c \in cset\},$
.6 $(\text{BOTTOM}) \to \{\},$
.7 $c \to \{mk\text{-}CI(\{c\})\}$
.8 end

92.0 $Join_{NPc} : CI\text{-set} \times CI\text{-set} \to CI\text{-set}$
.1 $Join_{NPc}\,(ac_1, ac_2) \triangleq AntiCh(ac_1 \cup ac_2)$

93.0 $Meet_{NPc} : CI\text{-set} \times CI\text{-set} \to CI\text{-set}$
.1 $Meet_{NPc}\,(ac_1, ac_2) \triangleq$
.2 let $cis = \{mk\text{-}CI(cs_1 \cup cs_2) \mid mk\text{-}CI(cs_1) \in ac_1, mk\text{-}CI(cs_2) \in ac_2\}$ in
.3 $AntiCh(cis)$

7.3 Projection into *pmax*.

We now look for a projection function *cProj* such that

94.0 $cProj(neqs)(cis) = CISproj(pmax)(cis)$

where *neqs* (normalized equations, discussed below) is a representation of the equations that were used to construct *pmax*. By combining 89, 90 and 94 we can easily compute $Eval_N(pmax)(t)$:

95.0 $Eval_N(pmax)(t) = cProj(neqs)(Eval_{NPc}(cset)(t))$

The projection function must take the elements in $Eval_N(\mathcal{P}(cset))(t)$ which are not already in *pmax* (in figure 10 the black bullets outside *pmax*) and project them down into *pmax*. An element is outside *pmax* if and only if it is rejected by an equation. It is important to realize that the number of *rejected* elements in $Eval_L(\mathcal{P}(cset))(t)$ in general is of the same order of magnitude as the number of *all* the elements in $Eval_L(\mathcal{P}(cset))(t)$, i.e. of exponential size. Consequently, a projection method, where elements outside *pmax* are moved to lower covers and then again tested for acceptance/rejection will be very inefficient.

Recall, from section 5.2, how an equation rejects concept-intersections from $\mathcal{P}(cset)$: Let $t_1 = t_2$ be an equation, let $p_c = \mathcal{P}(cset)$ and let

$$cis_1 = Eval_L(p_c)(t_1) \text{ and } cis_2 = Eval_L(p_c)(t_2)$$

then $eqrej = (cis_1 \cup cis_2) \setminus (cis_1 \cap cis_2)$ is the set of concept-intersections that is rejected, whereas the set of concept-intersections $(cis_1 \cap cis_2)$ is accepted by *this equation*, but of course it may be rejeted by another equation. We call the set $(cis_1 \cup cis_2) \setminus (cis_1 \cap cis_2)$ the equations reject-region and the set $cis_1 \cap cis_2$ the equations accept-region. The term-values cis_1 and cis_2 are both down-sets and so are the union and intersection of these down-sets (13), consequently the equations accept-region is also a down-set. To get an efficient way to determine if a concept-intersection is in an equations reject- or accept-region we first evaluate every equation so we can easily get the accept and reject region. To make the computations effective we represent a downset by its anti-chain. An equation is now represented by a "normalized" equation of type NEq:

types

96.0 $NEq :: \quad CI\text{-set} \times CI\text{-set}$

The first and second component are the anti-chains of the join and meet of the left and right hand term values in p_c.

97.0 $EvalEq : Cset \rightarrow Eq \rightarrow NEq$

.1 $EvalEq\,(cset)(mk\text{-}Eq(t_1, t_2)) \triangleq$

.2 $\quad$ let $ac_1 = Eval_{NPc}(cset)(t_1),$

.3 $\quad\quad ac_2 = Eval_{NPc}(cset)(t_2)$ in

.4 $\quad mk\text{-}NEq(Join_{NPc}(ac_1, ac_2), Meet_{NPc}(ac_1, ac_2))$

We will evaluate all the equations in this way:

98.0 $EvalEqs : Cset \rightarrow Eq\text{-set} \rightarrow NEq\text{-set}$

.1 $EvalEqs\,(cset)(eqs) \triangleq \{EvalEq(cset)(eq) \mid eq \in eqs\}$

Let $cset$ and eqs be the given set of concepts and equations repectively, then we will use the set of normalized equations $neqs = EvalEqs(cset)(eqs)$ to (implicitly) represent $pmax$. The accept-region for a normalized equation $mk\text{-}NEq(u, m)$ is now $DownSet(p_c)(m)$ – see fig. 11 – and the reject-region is

$$DownSet(p_c)(u) \setminus DownSet(p_c)(m)$$

We can now tell if a concept-intersection is in the reject-region of a normalized equation using the function defined below:

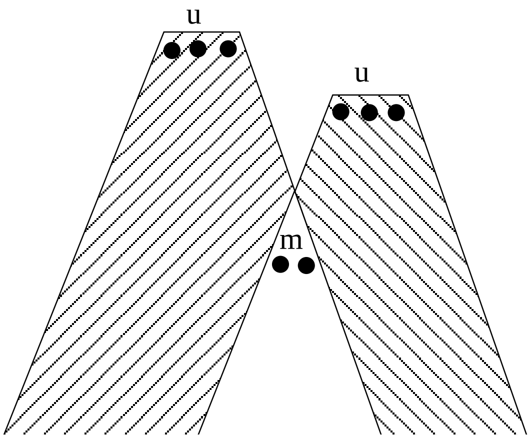
99.0 $InEqRej : NEq \rightarrow CI \rightarrow \mathbb{B}$

.1 $InEqRej\,(mk\text{-}NEq(u, m))(ci) \triangleq$

.2 $\quad ISA_N(\{ci\}, u) \wedge \neg\, ISA_N(\{ci\}, m)$

In words, the concept-intersection ci is in the reject-region if it is in one of the down-sets for the equation term values but not in the accept-region. Notice that $InEqRej$ uses the ISA_N relation between anti-chains (as defined in 48 and 54). If the concept-intersection ci is in the reject-region of an equation then all the elements below (ISA_P) ci, which are not rejected by any equation (i.e. are in $pmax$) must be below some element in m, so we have

100.0 $InEqRej(mk\text{-}NEq(\text{-}, m))(ci) \;\Rightarrow\; ISA_N(CISproj(pmax)(\{ci\}), m)$



The reject-region (hatched) and the accept-region corresponding to the normalized equation *mk-NEq*(u, m)

Figure 11: Reject-region and accept-region

7.4 Projection Step of a Concept-intersection

Given a set of normalized equations *neqs*, the projection of a set *cis* of concept-intersections into *pmax* is done iteratively. In each step we take from *cis* an element *ci* which is rejected by one of the equations *neq* and make a partial projection of *ci* using the equation *neq*.

So let *ci* be a concept-intersection that is in the reject-region for one of the normalized equations *mk-NEq*(u, m). Then *ci* itself is not in *pmax*, so *CISproj*$(pmax)(\{ci\})$ must be below *ci*. Furthermore *CISproj*$(pmax)(\{ci\})$ must also be in the equations accept-region i.e. (according to 100) below m. So — in the lattice $\mathcal{N}(\mathcal{P}(cset))$ — the anti-chain *CISproj*$(pmax)(\{ci\})$ is below both anti-chains *ci* and m:

$$ISA_N(CISproj(pmax)(\{ci\}), Meet_{NPc}(\{ci\}, m))$$

Hence we may use $ac = Meet_{NPc}(\{ci\}, m)$ as a first approximation for the projection of *ci*. Below *CIProjStep* defines the relation between a concept-intersection *ci*, that is rejected by one of the equations in the given set *neqs* of normalized equations, and the partial projection *ac* as described above.

101.0 $CIProjStep : NEq \rightarrow CI \times CI\text{-set} \rightarrow \mathbb{B}$
.1 $CIProjStep(neq)(ci, ac) \triangleq$
.2 $\quad InEqRej(neq)(ci) \wedge$
.3 $\quad$ let $mk\text{-}NEq(\text{-}, m) = neq$ in
.4 $\quad Meet_{NPc}(\{ci\}, m) = ac$

We have the following lemma for such a single concept-intersection projection step:

Lemma Let *neqs* be the given set of normalized equations.

102.0 $\forall\, neq \in neqs, ci : CI, ac : CI\text{-set} \cdot$
.1 $\quad CIProjStep(neq)(ci, ac) \Rightarrow$
.2 $\qquad ISA_N(ac, \{ci\}) \wedge ac \neq \{ci\} \wedge$
.3 $\qquad ISA_N(CISproj(pmax)(\{ci\}), ac)$

Proof To prove 102 above assume the left hand side of the implication above, i.e the lines 101.2-4 and prove the right hand side. Line 102.2 follows directly from the definition of ac in 101.4 and the fact that ci is in the reject region of the equation neq whereas ac is in the accept region of the equation. To prove Line 102.3 we use the assumptions and the *InEqRej*-property (100) which gives us

$$ISA_N(CISproj(pmax)(\{ci\}), m)$$

From the definition of *CISproj* we also have that the elements in $CISproj(pmax)(\{ci\})$ must be below (ISA_P) ci so

$$ISA_N(CISproj(pmax)(\{ci\}), \{ci\})$$

Hence $CISproj(pmax)(\{ci\})$ is a lower bound for both m and $\{ci\}$ and consequently we also have

$$ISA_N(CISproj(pmax)(\{ci\}), Meet_{NPc}(m, \{ci\}))$$

□

7.4.1 Projection Step of a Concept-intersection Set

Now we must return to the problem of projecting a set cis of concept-intersections into $pmax$, i.e. computing $CISproj(pmax)(cis)$. Let

$$cis = \{ci_1\} \cup \{ci_2\} \ldots \cup \{ci_j\} \cup \ldots \cup \{ci_n\}$$

In each projection step one element ci_j is partially projected using the projection described by the *CIProjStep* relation explained in the previous section. So let $neqs$ be the given set of normalized equations and let $ac : CI\text{-set}$ be such that $CIProjStep(neqs)(ci_j, ac)$ then in the next step cis is projected to:

$$newcis = \{ci_1\} \cup \{ci_2\} \ldots \cup ac \cup \ldots \cup \{ci_n\}$$

Even if cis is an anti-chain the new set of concept-intersection's is not necessarily an anti-chain. As we will see below, we may choose to convert it to an anti-chain or leave it as it is. We define the relation between a set of concept-intersection's — where an arbitrary ci is rejected by a normalized equation — and the new set of concept-intersection's by the relation *CISProjStep* defined below:

103.0 $CISProjStep : NEq\text{-set} \rightarrow CI\text{-set} \times CI\text{-set} \rightarrow \mathbb{B}$
.1 $CISProjStep\,(neqs)(cis_1, cis_2) \triangleq$
.2 $\quad \exists\, neq \in neqs, ci \in cis_1, ac : CI\text{-set} \cdot$
.3 $\qquad CIProjStep(neq)(ci, ac) \wedge$
.4 $\qquad \text{let } newcis = (cis_1 \setminus \{ci\}) \cup ac \text{ in}$
.5 $\qquad AntiCh(newcis) \subseteq cis_2 \subseteq newcis$

As can be seen from the last line in the definition of *CISProjStep* the new set of concept-intersection's cis_2 can be any set between the $newcis$ and its anti-chain. Consequently, when projecting a set of concept-intersection's to $CISproj(pmax)(cis)$, the intermediate sequence of concept-intersection's between cis and the final resulting anti-chain is not necessarily a sequence of anti-chains. The situation is illustrated in figure 12.

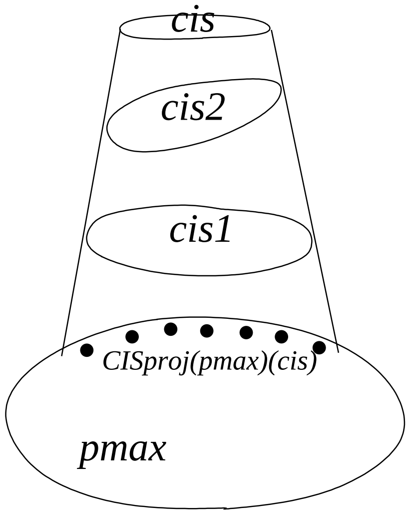


Figure 12: Computing $\mathit{CISproj}(\mathit{pmax})(\mathit{cis})$

We need some new concepts to handle the situation. In section 4.1 definition 54 the ISA_N relation between anti-chains in $\mathcal{N}(p)$ was defined. We have $\mathit{ISA}_N(ac_1, ac_2)$ iff every $ci_1 \in ac_1$ is below (ISA_P) some $ci_2 \in ac_2$. The ISA_N relation can be used on any set of concept-intersections, but in order to avoid confusion we reserve ISA_N to anti-chains and define a new relation for arbitrary sets:

104.0 $\mathit{ISA}_S : \mathit{CI}\text{-set} \times \mathit{CI}\text{-set} \rightarrow \mathbb{B}$

.1 $\mathit{ISA}_S\,(cis_1, cis_2) \triangleq \forall\, ci_1 \in cis_1 \cdot \exists\, ci_2 \in cis_2 \cdot \mathit{ISA}_P(ci_1, ci_2)$

If ac_1 and ac_2 are anti-chains then $\mathit{ISA}_N(ac_1, ac_2) = \mathit{ISA}_S(ac_1, ac_2)$. When ISA_S is applied to arbitrary concept-intersection-sets it is still transitive and reflexive, but not antisymmetric. E.g. we have $\mathit{ISA}_S(cis, \mathit{AntiCh}(cis))$ and $\mathit{ISA}_S(\mathit{AntiCh}(cis), cis)$.

For the ISA_S-relation we have the following simple properties:

105.0 $\mathit{ISA}_S(cis, cis \cup cis_1)$,

.1 $\mathit{ISA}_S(cis_1, cis_2) \wedge \mathit{ISA}_S(cis_3, cis_4) \;\Rightarrow\; \mathit{ISA}_S(cis_1 \cup cis_3, cis_2 \cup cis_4)$

In the sequel we also need the following property:

106.0 $\mathit{ISA}_S(\mathit{CISproj}(\mathit{pmax})(cis), cis1) \wedge \mathit{ISA}_S(cis1, cis)$

.1 $\quad\Rightarrow\; \mathit{CISproj}(\mathit{pmax})(cis1) = \mathit{CISproj}(\mathit{pmax})(cis)$

In words, if we have a set cis and its projection into pmax, then any set $cis1$ between cis and its projection will have the same projection into pmax as cis. A proof is in B.9.

Finally, in order to understand the projection of a set of concept-intersection's we also need the property below for the projection function $\mathit{CISproj}$ (defined in 26):

107.0 $\mathit{CISproj}(p)(cis_1 \cup cis_2) = \mathit{AntiCh}(\mathit{CISproj}(p)(cis_1) \cup \mathit{CISproj}(p)(cis_2))$

Informally, the result of $\mathit{CISproj}(p)(cis)$ is an anti-chain (see 26.2), but the union of two anti-chains is usually not an anti-chain, because a part of one anti-chain may have crept below the

other. Hence we need to take the anti-chain of the union. The property above shows us, that when all elements are in *pmax* we must remember to take the anti-chain of the final *InP*. A proof is in B.10.
We can now formulate the following lemma about the previously defined relation *CISProjStep*:

Lemma Let *neqs* be the given set of normalized equations.

108.0 $\forall\, cis_1 : CI\text{-set}, cis_2 : CI\text{-set} \cdot$
.1 $\quad CISProjStep(neqs)(cis_1, cis_2) \;\Rightarrow$
.2 $\quad\quad ISA_S(cis_2, cis_1) \wedge cis_2 \neq cis_1 \wedge$
.3 $\quad\quad CISproj(pmax)(cis_1) = CISproj(pmax)(cis_2)$

Proof: Assume the left hand side of the implication above, i.e.

$$CISProjectStep(neqs)(cis_1, cis_2)$$

According to 103 this is equivalent to

109.0 $\exists\, neq \in neqs, ci \in cis_1, ac : CI\text{-set} \cdot$
.1 $\quad CIProjStep(neq)(ci, ac) \wedge$
.2 $\quad \text{let } newcis = (cis_1 \setminus \{ci\}) \cup ac \text{ in}$
.3 $\quad AntiCh(newcis) \subseteq cis_2 \subseteq newcis$

Let neq, ci, ac be the values that exist in 109.0. We then have

110.0 $CIProjStep(neq)(ci, ac) \wedge$
.1 $ci \in cis_1 \wedge$
.2 $\text{let } newcis = (cis_1 \setminus \{ci\}) \cup ac \text{ in}$
.3 $AntiCh(newcis) \subseteq cis_2 \subseteq newcis$

Using 102 to 110.0 above gives

111.0 $ISA_N(ac, \{ci\}) \wedge ac \neq \{ci\} \wedge$
.1 $ISA_N(CISproj(pmax)(\{ci\}), ac) \wedge$

proof of 108.2 We first prove

$$ISA_S(newcis, cis_1)$$

which (according to 104) is equivalent to

$$\forall\, ci_2 \in newcis \cdot \exists\, ci_1 \in cis_1 \cdot ISA_P(ci_2, ci_1)$$

First let ci_2 be an arbitrary element in *newcis* i.e.

$$ci_2 \in (cis_1 \setminus \{ci\}) \cup ac$$

We consider the two case $cis_1 \setminus \{ci\}$ and ac.

$ci_2 \in (cis_1 \setminus \{ci\})$: Then

$$ci_2 \in cis_1 \wedge ISA_P(ci_2, ci_2) \text{ so } \exists\, ci_1 \in cis_1 \cdot ISA_P(ci_2, ci_1)$$

$ci_2 \in ac$: From 111.0, the definition of ISA_N and $ci_2 \in ac$ we have

$$ISA_P(ci_2, ci) \text{ where } ci \in cis_1 \text{ so } \exists\, ci_1 \in cis_1 \cdot ISA_P(ci_2, ci_1)$$

From the two cases above we may conclude 109 so we have $ISA_S(newcis, cis_1)$. Finally from 110.3 and 105 we get $ISA_S(cis_2, cis_1)$.

proof of 108.3 From the definition of *CISproj* (26) we have $ISA_S(CISproj(cis), cis)$ for any *cis*. So we also have

112.0 $ISA_S(CISproj(pmax)(cis_1 \setminus \{ci\}), cis_1 \setminus \{ci\})$

From 111.1 we also have

113.0 $ISA_S(CISproj(pmax)(\{ci\}), ac)$

If we now combine these two ISA_S relations using 105 we get

114.0 $ISA_S(CISproj(pmax)(cis_1 \setminus \{ci\}) \cup CISproj(pmax)(\{ci\}), (cis_1 \setminus \{ci\}) \cup ac)$

Next, if we use that $ISA_S(AntiCh(cis), cis)$ and the transitivity of ISA_S to 114 we get

115.0 $ISA_S(AntiCh(CISproj(pmax)(cis_1 \setminus \{ci\}) \cup CISproj(pmax)(\{ci\})), (cis_1 \setminus \{ci\}) \cup ac)$

Finally, using 107 and 110.2 gives us

116.0 $ISA_S(CISproj(pmax)(cis_1), newcis)$

Now we want the property above not just for $cis_2 = newcis$ but for all cis_2 ranging between *newcis* and $AntiCh(newcis)$. From 27 we get

117.0 $\forall\, cis \cdot AntiCh(newcis) \subseteq cis \subseteq newcis \;\Rightarrow$
.1 $CISproj(pmax)(cis) = CISproj(pmax)(newcis)$

This together with 110.3 let us conclude that

118.0 $ISA_S(CISproj(pmax)(cis_1), cis_2)$

Finally, from 118 above, 108.2 and 106 we get 108.3.

7.5 Projection Sequence

We define a projection sequence for a given concept-intersection-set cis_0 as a finite sequence of concept-intersections

119.0 $cis_0, cis_1, cis_2, \ldots, cis_n$ such that
.1 $CISProjStep(neqs)(cis_j, cis_{j+1})$ for $0 \leq j \leq n\text{-}1 \wedge$
.2 $IsAntiChain(cis_n) \wedge$
.3 $\neg\, \exists\, cis : CI\text{-set} \cdot CISProjStep(cis_n, cis)$

A projection sequence ends with a concept-intersection-set cis_n which is an anti-chain and which can not be projected further.

Projection Theorem

120.0 Every $cis_0 : CI$-set has a projection sequence
.1 $cis_0, cis_1, cis_2, \ldots, cis_n$ and
.2 $cis_n = CISproj(pmax)(cis_0)$

proof of 120.2 We first prove by induction that

121.0 $ISA_S(cis_n, cis_0) \wedge CISproj(pmax)(cis_0) = CISproj(pmax)(cis_n)$

The base case

122.0 $ISA_S(cis_0, cis_0) \wedge CISproj(pmax)(cis_0) = CISproj(pmax)(cis_0)$

is obviously true. For the inductive case we assume

123.0 $\mathit{ISA}_S(\mathit{cis}_j, \mathit{cis}_0) \wedge \mathit{CISproj}(\mathit{pmax})(\mathit{cis}_0) = \mathit{CISproj}(\mathit{pmax})(\mathit{cis}_j)$

From the definition of a projection sequence (119) we have (119.1)

124.0 $\mathit{CISProjStep}(\mathit{neqs})(\mathit{cis}_j, \mathit{cis}_{j+1})$

Now, using the lemma about *CISProjStep* (108) we get

125.0 $\mathit{ISA}_S(\mathit{cis}_{j+1}, \mathit{cis}_j) \wedge$
.1 $\mathit{CISproj}(\mathit{pmax})(\mathit{cis}_j) = \mathit{CISproj}(\mathit{pmax})(\mathit{cis}_{j+1})$

Using this and the transitivity of ISA_S to the assumption in 123 gives

126.0 $\mathit{ISA}_S(\mathit{cis}_{j+1}, \mathit{cis}_0) \wedge \mathit{CISproj}(\mathit{pmax})(\mathit{cis}_0) = \mathit{CISproj}(\mathit{pmax})(\mathit{cis}_{j+1})$

So combining the base case and the inductive case we have proved 121.

To prove 120.2 we know that cis_n in 120.1 according to the definition of a projection sequence is an anti-chain and cannot be projected further (119.2-3). Using this to the definition of *CISproj* (26) gives that

127.0 $\mathit{CISproj}(\mathit{pmax})(\mathit{cis}_n) = \mathit{cis}_n$

which combined with 121 gives 120.2.

proof of 120.0-1 Every cis_j in the projection sequence can be given a size, namely the sum of size of each $ci \in \mathit{cis}_j$, where the size of a concept-intersection ci is the size of its set of basic concepts. That size will grow in each step of the projection sequence and thus we will never during the projection return to a previous situation.

7.6 Implementation of *cProj*

We are now ready to look for an implementation of the function *cProj* as defined in 94:

128.0 $\mathit{cProj}(\mathit{neqs})(\mathit{cis}) = \mathit{CISproj}(\mathit{pmax})(\mathit{cis})$

From the projection theorem (120) we can see that $\mathit{cProj}(\mathit{neqs})(\mathit{cis})$ in some way must make a concrete projection sequence for *cis* and return the final cis_n as result. Here the most difficult task is to choose in the current concept-intersection set one of the concept-intersection's that is rejected by a normalized equation. The choice will result in different projection sequence constructions (breadth first/depth first). Now we will just specify the task for such a function:

129.0 $\mathit{ChooseCiNeq}\,(\mathit{neqs} : \mathit{NEq}\text{-set})(\mathit{cis} : \mathit{CI}\text{-set})\ \mathit{answer} : \mathit{CAnswer}$
.1 post $(\exists\, \mathit{neq} \in \mathit{neqs}, ci \in \mathit{cis} \cdot$
.2 $\mathit{InEqRej}(\mathit{neq})(ci) \wedge$
.3 let $\mathit{mk}\text{-}\mathit{NEq}(\text{-}, m) = \mathit{neq}$ in $\mathit{answer} = \mathit{mk}\text{-}\mathit{Rej}(ci, m))$
.4 $\vee\ (\neg \exists\, \mathit{neq} \in \mathit{neqs}, ci \in \mathit{cis} \cdot \mathit{InEqRej}(\mathit{neq})(ci) \wedge \mathit{answer} = \mathtt{Inp})$

where

types

130.0 $\mathit{CAnswer} = \mathit{Rej} \mid \mathtt{InP};$

131.0 $\mathit{Rej} :: \ \ \mathit{CI} \times \mathit{CI}\text{-set}$

Having such a function available makes it easy to define the *cProj*-function:

132.0 $cProj : NEq\text{-set} \rightarrow CI\text{-set} \rightarrow CI\text{-set}$

.1 $cProj\ (neqs)(cis) \triangleq$

.2 $\quad$ cases $ChooseCiNeq(neqs)(cis)$:

.3 $\quad\quad mk\text{-}Rej(ci, m) \rightarrow cProj(neqs)(cis \setminus \{ci\} \cup Meet_{NPc}(m, \{ci\})),$

.4 $\quad\quad InP \rightarrow AntiCh(cis)$

.5 $\quad$ end

Notice that the sequence of argument concept-intersection's for *cProj* combined with the final anti-chain result makes up a projection sequence.

An Implementation of *cEvalN* **and** *cTMdisjPO***.** Given the *cProj*-function defined above we are now able to implement $Eval_N(pmax)(t)$ using projection as defined in 95.

133.0 $cEvalN : C\text{-set} \rightarrow NEq\text{-set} \rightarrow Term \rightarrow CI\text{-set}$

.1 $cEvalN\ (cset)(neqs)(t) \triangleq cProj(neqs)(Eval_{NPc}(cset)(t))$

Finally, from the definition of *TMdisjPO* (84) we can define the function that computes the most disjoint lattice with respect to a given set of inserted terms :

134.0 $cTMdisjPO : Cset \rightarrow Eq\text{-set} \rightarrow Term\text{-set} \rightarrow PO$

.1 $cTMdisjPO\ (cset)(eqs)(insterms) \triangleq$

.2 $\quad$ let $neqs = EvalEqs(cset)(eqs)$ in

.3 $\quad \bigcup\{cEvalN(cset)(neqs)(t) \mid t \in insterms\}$

8 Introducing Attributes

We now consider the full concept algebra including functional binary relations in the form of attributes (as described in [3]), so we now assume available a set of attributes *aset*. Besides the axioms for distributive lattices we now also have the axioms below for each attribute $\alpha \in aset$:

135.0	Strictness	$\alpha(\perp) = \perp$
	Distribution of +	$\alpha(X + Y) = \alpha(X) + \alpha(Y)$
	Distribution of $*$	$\alpha(X * Y) = \alpha(X) * \alpha(Y)$

Notice the terminology: the concept $\alpha(X)$ is called the *α-attribution* of the concept X. From the rules above it is easy to derive the monotonicity rule below:

Monotonicity $\quad X \leq Y \Rightarrow \alpha(X) \leq \alpha(Y)$

8.1 Terms and Equations

The syntax for general (ground) terms (Term) which may have TOP as a subterm are now:

types

136.0 $\quad BasicTerm = C \mid \text{TOP} \mid \text{BOTTOM};$

137.0 $\mathit{Term} = \mathit{Join} \mid \mathit{Meet} \mid \mathit{Attr} \mid \mathit{BasicTerm};$

138.0 $\mathit{Join} :: \ \mathit{Term} \times \mathit{Term}$ — Join-term ;

139.0 $\mathit{Meet} :: \ \mathit{Term} \times \mathit{Term}$ — Meet-term ;

140.0 $\mathit{Attr} :: \ A \times \mathit{Term}$ — Attribute-term

These general terms will only be used when making queries to the lattice database. When constructing the lattice database we will only allow terms not containing the TOP-term, so equations and inserted terms must be terms without the TOP-term. Such restricted terms are defined by the type *Termr*:

141.0 $\mathit{BTerms} : \mathit{Term} \to \mathit{BasicTerm}\text{-set}$

.1 $\mathit{BTerms}\,(t) \triangleq$

.2 cases t :

.3 $\mathit{mk\text{-}Join}(t_1, t_2) \to \mathit{BTerms}(t_1) \cup \mathit{BTerms}(t_2),$

.4 $\mathit{mk\text{-}Meet}(t_1, t_2) \to \mathit{BTerms}(t_1) \cup \mathit{BTerms}(t_2),$

.5 $\mathit{mk\text{-}Attr}(\alpha, t) \to \mathit{BTerms}(t),$

.6 $bt \to \{bt\}$

.7 end

types

142.0 $\mathit{Termr} = \mathit{Term}$

.1 inv $tr \triangleq \text{TOP} \notin \mathit{BTerms}(tr);$

143.0 $\mathit{Eq} :: \ \mathit{Termr} \times \mathit{Termr}$ — term-equation

When writing terms in examples we use as usual the two infix operators $+$ and $*$ to represent *Join* and *Meet* respectively. The attribution $\mathit{mk\text{-}Attr}(\alpha, t)$ is written as $\alpha(t)$. When we evaluate the terms in a given algebra we restrict term-constants $c : C$ to be in a given set of concepts *cset* and similar attributes $\alpha : A$ to be in a given set of attributes *aset*. But before we can evaluate terms we must first consider how attribution concepts influence the concept lattice and with that the lattice algebra in which to evaluate terms.

8.2 Concept Intersections with Attributions

In order to handle attributes we must redefine the type of concept intersections so the new attribution concepts are included. If α is an attribute and $c : C$ a basic concept then we may also need the attribution $\alpha(c)$ of this concept as a new basic concept. But how do we represent the attribution of concept-intersections, i.e. if $[c_1, c_2, \ldots, c_n]$ is a concept-intersection, how do we represent the attribution $\alpha(\{[c_1, c_2, \ldots, c_n]\})$? The attribution axiom concerning distributivity of $*$ suggests that $\{[\alpha(c_1), \alpha(c_2), \ldots, \alpha(c_n)]\}$ would be a valid representation. Hence, in the sequel we let the basic concepts used to construct concept-intersections include the given set of named concepts *and* every possible attribution of these named concepts. Of course, we must later verify that this representation actually makes it possible to construct an algebra satisfying the attribution axioms.

types

144.0 $C = \mathsf{token}$ — The type of named concepts / concept constants;

145.0 $A = \mathsf{token}$ — The type of attributes, $\alpha, \beta, \ldots : A$;

146.0 $B = C \mid At$ — The type of basic concepts, $a, b, \alpha(a), \alpha(\beta(a)), \ldots : B$;

147.0 $At :: \quad A \times B$ — The type of attributions, $\alpha(a), \alpha(\beta(a)), \ldots : At$;

148.0 $Bset = B\text{-}\mathsf{set}$

.1 $\mathsf{inv}\ (bset) \triangleq bset \neq \{\}$;

149.0 $CI :: \quad Bset$ — The type of concept intersections ;

150.0 $PO = CI\text{-}\mathsf{set}$

A basic concept (B) is a named concept (C) or an attribution (At). We can now define the ordering relation ISA_P between concept-intersections exactly as in 6:

151.0 $ISA_P : CI \times CI \rightarrow \mathbb{B}$

.1 $ISA_P\ (mk\text{-}CI(bs_1), mk\text{-}CI(bs_2)) \triangleq bs_2 \subseteq bs_1$

Because there is an infinite number of possible attributions there will also be an infinite number of basic concepts. Consequently, the number of possible concept-intersections will also be infinite. In order to be able to continue using the approach based on Birkhoff's representation theorem, the considered lattice $\mathcal{O}(p)$ must be finite and consequently also the partial order p from which the lattice is constructed. Therefore, in the sequel, when we construct a finite partial order p for a concept algebra satisfying a given set of equations, we do not start from the infinite powerset partial order. Instead we assume that a finite set of basic concepts is provided as part of a lattice specification by specifying both a set of equations and a set of inserted terms:

types

152.0 $LatSpec :: \quad Eq\text{-}\mathsf{set} \times Termr\text{-}\mathsf{set}$

The set of named concepts, attributes and the finite set of basic concepts are now extracted from the terms in a lattice specification in the following way:

153.0 $SpecTerms : LatSpec \rightarrow Termr\text{-}\mathsf{set}$

.1 $SpecTerms\ (mk\text{-}LatSpec(eqs, terms)) \triangleq$

.2 $\{t \mid mk\text{-}Eq(t_1, t_2) \in eqs \cdot t \in \{t_1, t_2\}\} \cup terms$

154.0 $\mathit{TBset} : \mathit{Term} \to B\text{-}\mathsf{set}$
.1 $\mathit{TBset}\,(t) \triangleq$
.2 $\quad \mathsf{cases}\ t:$
.3 $\qquad \mathit{mk\text{-}Join}(t_1, t_2) \to \mathit{TBset}(t_1) \cup \mathit{TBset}(t_2),$
.4 $\qquad \mathit{mk\text{-}Meet}(t_1, t_2) \to \mathit{TBset}(t_1) \cup \mathit{TBset}(t_2),$
.5 $\qquad \mathit{mk\text{-}Attr}(\alpha, t) \to \{\mathit{mk\text{-}At}(a, b) \mid b \in \mathit{TBset}(t)\} \cup \mathit{TBset}(t),$
.6 $\qquad (\textsc{top}) \to \{\},$
.7 $\qquad (\textsc{bottom}) \to \{\},$
.8 $\qquad c \to \{c\}$
.9 $\quad \mathsf{end}$

155.0 $\mathit{extrLatSpec} : \mathit{LatSpec} \to C\text{-}\mathsf{set} \times A\text{-}\mathsf{set} \times B\text{-}\mathsf{set}$
.1 $\mathit{extrLatSpec}\,(\mathit{spec}) \triangleq$
.2 $\quad \mathsf{let}\ \mathit{allTerms} = \mathit{SpecTerms}(\mathit{spec})\ \mathsf{in}$
.3 $\quad \mathsf{let}\ \mathit{bset} = \bigcup\{\mathit{TBset}(t) \mid t \in \mathit{allTerms}\}\ \mathsf{in}$
.4 $\quad \mathsf{let}\ \mathit{concepts} = \{c \mid c \in \mathit{bset} \cdot \mathit{is\text{-}C}(c)\}\ \mathsf{in}$
.5 $\quad \mathsf{let}\ \mathit{attributes} = \{a \mid \mathit{mk\text{-}At}(a, \text{-}) \in \mathit{bset}\}\ \mathsf{in}$
.6 $\quad \mathit{mk\text{-}}(\mathit{concepts}, \mathit{attributes}, \mathit{bset})$

Example Given the specification

```
equations
   a(a(y)+z)= y * b(a(x))
terms a(x),b(x)
```

Applying *extrLatSpec* to the specification above yields

$$\begin{aligned}
&\mathit{concepts} = \{x, y, z\}\\
&\mathit{attributes} = \{a, b\}\\
&\mathit{bset} = \{a(x), b(x), x, y, z, a(y), a(a(y)), a(z), b(a(x))\}
\end{aligned}$$

□

Given a finite set $\mathit{bset} : \mathit{Bset}$ of basic concepts (extracted from a specification), we can construct the powerset partial order $\mathcal{P}(\mathit{bset})$ with ISA_P as the ordering relation. The powerset is constructed exactly as in 5:

156.0 $\mathcal{P} : \mathit{Bset} \to \mathit{PO}$
.1 $\mathcal{P}\,(\mathit{bs}) \triangleq \{\mathit{mk\text{-}CI}(\mathit{bs}') \mid \mathit{bs}' : \mathit{Bset} \cdot \mathit{bs}' \subseteq \mathit{bs}\}$

In the sequel any subset of $\mathcal{P}(\mathit{bset})$ is considered a partial order with the same induced ordering ISA_P.

Remark. It is essential to understand the consequence of this finite attribution approach. If an attribution is not in the set of basic concepts it actually means that the attribution collapses to bottom in the lattice. For instance, consider the specification below:

```
equations
   a(x)>= x
terms a(a(x))
```

which produces the $bset = \{x, a(x), a(a(x))\}$. Consequently the terms $a(a(a(x))), a(a(a(a(x)))), \ldots$ all evaluate to bottom. Now, according to the specification we have $a(x) \geq x$ and because of the monotonicity rule for attribution we also have $a(a(x)) \geq a(x)$, $a(a(a(x))) \geq a(a(x))$, etc. As $a(a(a(x)))$ is bottom and $a(a(x)), a(x)$ and x all are below they also become bottom.

As another example consider the situation where $\{x, y, a(x)\} \subseteq bset$ but $a(y) \notin bset$ and assume that $y > x$ according to the equations (may be in some very indirect way). Then according to the monotonicity rule for attribution we also have $a(y) > a(x)$ and because $a(y)$ is bottom we also have that $a(x)$ is bottom. This was probably not the intention when $a(x)$ was mentioned among the terms.

We may conclude that it may be difficult for a user to specify a correct $bset$. Later (in section 15) we will show how to construct the so-called most disjoint concept algebra without having the $bset$ explicitly available.

8.3 Auxiliary Functions

We can easily extract the concepts and attributes used in a set of concept-intersections:

157.0 $\mathit{UsedAttrsInB} : B \to A\text{-set}$

.1 $\mathit{UsedAttrsInB}\,(b) \triangleq$
.2 cases b :
.3 $\mathit{mk\text{-}At}(a, b1) \to \{a\} \cup \mathit{UsedAttrsInB}(b1)$,
.4 - $\to \{\}$
.5 end

158.0 $\mathit{UsedAttrsInCIS} : \mathit{CI}\text{-set} \to A\text{-set}$

.1 $\mathit{UsedAttrsInCIS}\,(cis) \triangleq \bigcup\{\mathit{UsedAttrsInB}(b) \mid \mathit{mk\text{-}CI}(bset) \in cis, b \in bset\}$

159.0 $\mathit{UsedConcInB} : B \to C$

.1 $\mathit{UsedConcInB}\,(b) \triangleq$
.2 cases b :
.3 $\mathit{mk\text{-}At}(\text{-}, b1) \to \mathit{UsedConcInB}(b1)$,
.4 $c \to c$
.5 end

160.0 $\mathit{UsedConceptsInCIS} : \mathit{CI}\text{-set} \to C\text{-set}$

.1 $\mathit{UsedConceptsInCIS}\,(cis) \triangleq \{\mathit{UsedConceptInB}(b) \mid \mathit{mk\text{-}CI}(bset) \in cis, b \in bset\}$

A partial order p conforms with a set of concepts and attributes if it only uses concepts and attributes from these sets:

161.0 $\mathit{Conforms} : C\text{-set} \times A\text{-set} \to PO \to \mathbb{B}$

.1 $\mathit{Conforms}\,(cset, aset)(p) \triangleq$
.2 $(\mathit{UsedConceptsInCIS}(p) \subseteq cset) \wedge (\mathit{UsedAttrsInCIS}(p) \subseteq aset)$

We will also need the top level attributes in a set of concept-intersections:

162.0 $AttrsInCI : CI \rightarrow A\text{-set}$

.1 $AttrsInCI\,(mk\text{-}CI(bs)) \triangleq \{a \mid mk\text{-}At(a,\text{-}) \in bs\}$

163.0 $AttrsInCIS : CI\text{-set} \rightarrow A\text{-set}$

.1 $AttrsInCIS\,(cis) \triangleq \bigcup\{AttrsInCI(ci) \mid ci \in cis\}$

9 The Lattice Algebra with Attribution

Assume *cset* is a set of concepts, *aset* a set of attributes. Furthermore assume *bset* is a finite set of basic concepts and $p \subseteq \mathcal{P}(bset)$ is a finite partial order of concept-intersections such that $Conforms(cset, aset)(p)$. The lattice $\mathcal{O}(p)$ can now be viewed as the (one sorted) algebra

164.0 $\mathcal{CA}(cset, aset, p) =$

.1 $< \mathcal{O}(p); Join_C, Meet_C, A_C(p)(aset), C_C(p)(cset), \text{TOP}_C, \text{BOTTOM}_C >$

The components in the tuple above are as follows: $\mathcal{O}(p)$ is the carrier set and $Join_C$ and $Meet_C$ are defined similar to the two binary operators $Join_L$ and $Meet_L$ (defined in 30 and 31):

165.0 $Join_C : CI\text{-set} \times CI\text{-set} \rightarrow CI\text{-set}$

.1 $Join_C\,(cis_1, cis_2) \triangleq cis_1 \cup cis_2$

166.0 $Meet_C : CI\text{-set} \times CI\text{-set} \rightarrow CI\text{-set}$

.1 $Meet_C\,(cis_1, cis_2) \triangleq cis_1 \cap cis_2$

We also have

167.0 $ISA_C : CI\text{-set} \times CI\text{-set} \rightarrow \mathbb{B}$

.1 $ISA_C\,(cis_1, cis_2) \triangleq cis_1 \subseteq cis_2$

To the set of named concepts *cset* there is a set of constants/values in $\mathcal{O}(p)$:

$$C_C(p)(cset) = \{cValue_C(p)(c) \mid c \in cset\}$$

where $cValue_C$ is similar to the function $cValue_L$ defined in 36:

168.0 $cValue_C : PO \rightarrow C \rightarrow CI\text{-set}$

.1 $cValue_C\,(p)(c) \triangleq DownSetC(p)(\{mk\text{-}CI(\{c\})\})$

The value of the two constants TOP_C and BOTTOM_C is defined to p respectively $\{\}$. Corresponding to the set of attributes *aset* we now have a set of unary operators

$$A_C(p)(aset) = \{Attribution_{CA}(p)(\alpha) \mid \alpha \in aset\}$$

So for each attribute α we now have a unary operator $Attribution_{CA}(p)(\alpha)$ as defined below:

169.0 $Attribution_{CA} : PO \rightarrow A \rightarrow CI\text{-set} \rightarrow CI\text{-set}$

.1 $Attribution_{CA}\,(p)(\alpha)(cis) \triangleq DownSetC(p)(\{attrCI(\alpha)(ci) \mid ci \in cis\})$

170.0 $attrCI : A \to CI \to CI$

.1 $attrCI\,(\alpha)(mk\text{-}CI(bs)) \triangleq mk\text{-}CI(\{mk\text{-}At(\alpha, b) \mid b \in bs\})$

Notice that *DownSetC* (the cut version of *DownSet*) is used because $attrCI(\alpha)(ci)$ may produce a concept-intersection ci' which is not in p. This may happen for two reasons: 1) the concept-intersection ci' contains a basic concept which is not in *bset* so ci' cannot be in $\mathcal{P}(bset)$ or 2) the considered concept-intersection $ci' \in \mathcal{P}(bset)$ but is not in $p \subseteq \mathcal{P}(bset)$. A concept-intersection not in p is simply cut away.

Later, when making proofs about $Attribution_{CA}$ it will be convenient to have the following function and equality available

171.0 $atCIs : A \to CI\text{-set} \to CI\text{-set}$

.1 $atCIs\,(\alpha)(cis) \triangleq \{attrCI(\alpha)(ci) \mid ci \in cis\}$

Applying this to the definition of $Attribution_{CA}$ (169) gives the equality below

172.0 $Attribution_{CA}(p)(\alpha)(cis) = DownSetC(p)(atCIs(\alpha)(cis))$

Attribution Properties For $Attribution_{CA}$ we have a number of useful properties. Let $p \subseteq \mathcal{P}(bset)$ be an arbitrary subset of $\mathcal{P}(bset)$ then we have

173.0 $Attribution_{CA}(p)(\alpha)(cis) \subseteq p$

.1 $cis_1 \subseteq cis_2 \;\Rightarrow\; Attribution_{CA}(p)(\alpha)(cis_1) \subseteq Attribution_{CA}(p)(\alpha)(cis_2)$

.2 $Attribution_{CA}(p \setminus d)(\alpha)(cis) = Attribution_{CA}(p)(\alpha)(cis) \setminus d$

.3 $p_1 \subseteq p_2 \;\Rightarrow\; Attribution_{CA}(p_1)(\alpha)(cis) = Attribution_{CA}(p_2)(\alpha)(cis) \cap p_1$

.4 $Attribution_{CA}(p_1)(\alpha)(cis) \cup Attribution_{CA}(p_2)(\alpha)(cis) = Attribution_{CA}(p_1 \cup p_2)(\alpha)(cis)$

The properties in 173.0– 173.3 can easily be seen from definitions 169 and 170 and the downset properties 14 and 15. For example, to prove 173.0 notice that $Attribution_{CA}(p)(\alpha)(cis)$ is a downset $Downset(p)(\ldots)$, and $Downset(p)(\ldots) \subseteq p$. The properties in 173.2 and 173.3 are very similar ($p \setminus d \sim p_1, p \sim p_2$). A proof of 173.4 is shown in section B.5.

Concerning the above defined algebra $\mathcal{CA}(cset, aset, p)$ we know that it is a distributive lattice for the same reasons as for the algebra $\mathcal{LA}(cset, p)$ defined in 35. But it remains to be shown that the new attribute operations work correctly. First of all we can see directly from the definition of $Attribution_{CA}$ (169) that $\alpha(cis)$ is a downset and hence a value in $\mathcal{O}(p)$. Next we must consider the attribute axioms.

9.1 Attribute Axioms.

For the algebra $\mathcal{CA}(cset, aset, p)$ to be a concept algebra it must fulfill the axioms for attributes as defined in 135. But, as we will see below, if we do not make any further assumptions, then only the attribute axioms concerning strictness and distribution of join is fulfilled.

Strictness:

$$\alpha(\bot) = Attribution_{CA}(p)(\alpha)(\{\}) = DownSet(p)(\{attrCI(\alpha)(ci) \mid ci \in \{\}\})$$
$$= DownSet(p)(\{\}) = \{\} = \bot$$

Distribution of Join:

$$
\begin{aligned}
&\alpha(Join_L(cis_1, cis_2))\\
&= Attribution_{CA}(p)(\alpha)(Join_L(cis_1, cis_2))\\
&= Attribution_{CA}(p)(\alpha)(cis_1 \cup cis_2)\\
&= DownSetC(p)(\{attrCI(\alpha)(ci) \mid ci \in cis_1 \cup cis_2\})\\
&= DownSetC(p)(\{attrCI(\alpha)(ci) \mid ci \in cis_1\} \cup \{attrCI(\alpha)(ci) \mid ci \in cis_2\})\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\text{from } 12\\
&= DownSetC(p)(\{attrCI(\alpha)(ci) \mid ci \in cis_1\}) \cup\\
&\quad DownSetC(p)(\{attrCI(\alpha)(ci) \mid ci \in cis_2\})\\
&= Attribution_{CA}(p)(\alpha)(cis_1) \cup Attribution_{CA}(p)(\alpha)(cis_2)\\
&= Join_L(Attribution_{CA}(p)(\alpha)(cis_1), Attribution_{CA}(p)(\alpha)(cis_2))
\end{aligned}
$$

The proof above uses the downset property (12)

$$DownSet(p)(cis_1 \cup cis_2) = DownSet(p)(cis_1) \cup DownSet(p)(cis_2)$$

To prove that the axiom for distribution of meet is satisfied would be easy if we had a similar downset property for intersection. But we don't. As an example consider the partial order to the right in figure 1. Let $cis_1 = \{a\}$ and $cis_2 = \{c\}$. Then $DownSet(p)(cis_1 \cap cis_2) = \{\}$, but $DownSet(p)(cis_1) \cap DownSet(p)(cis_2) = \{a, ab, ac, abc\} \cap \{c, ac, bc, abc\} = \{ac, abc\}$. Also for overlapping sets like $cis_1 = \{a, b\}$ and $cis_2 = \{b, c\}$ we have the same problem.

One can easily make examples which shows that the algebra $\mathcal{CA}(cset, aset, p)$ does not fulfil the attribution axiom concerning distribution of meet. Hence, $\mathcal{CA}(cset, aset, p)$ is in general *not* a concept algebra. In sections 10 and 11 it is shown, that $\mathcal{CA}(cset, aset, p)$ is a concept algebra if p satisfies certain properties.

9.2 The Value of Terms in the Algebra $\mathcal{CA}(cset, aset, p)$.

Now, given the algebra $\mathcal{CA}(cset, aset, p)$, we define the value of terms in this algebra, i.e. the value of a term is a set of concept-intersections from p:

174.0 $Eval_{CA} : PO \to Term \to CI\text{-set}$

.1 $Eval_{CA}\,(p)(t) \triangleq$

.2 cases t :

.3 $mk\text{-}Join(t_1, t_2) \to Join_C(Eval_{CA}(p)(t_1), Eval_{CA}(p)(t_2))$,

.4 $mk\text{-}Meet(t_1, t_2) \to Meet_C(Eval_{CA}(p)(t_1), Eval_{CA}(p)(t_2))$,

.5 $mk\text{-}Attr(\alpha, t) \to Attribution_{CA}(p)(\alpha)(Eval_{CA}(p)(t))$,

.6 $(\text{TOP}_C) \to p$,

.7 $(\text{BOTTOM}_C) \to \{\}$,

.8 $c \to cValue_C(p)(c)$

.9 end

Before attribution was introduced there were simple relations between the partial order p and the result of term evaluation as shown in section 4, def. 42. These properties showed a kind of "linear" relation between the partial order and the resulting term-value. After attribution has been introduced this linearity has disappeared and we are now left with the properties below:

Term Value Properties: Let *bset* be a set of basic concepts (extracted from a lattice specification). Furthermore, let t be a term, p, p_1 and p_2 subsets of $\mathcal{P}(bset)$ and $cis \in p$ a set of concept-intersections then

175.0 $Eval_{CA}(p)(t) \subseteq p$
.1 $p_1 \subseteq p_2 \Rightarrow Eval_{CA}(p_1)(t) \subseteq Eval_{CA}(p_2)(t)$
.2 $Eval_{CA}(p \setminus cis)(t) \subseteq Eval_{CA}(p)(t) \setminus cis$

The properties above can be shown by structural induction on the term-structure (and proofs for 175.0 and 175.1 may be found in section B.7 and B.8 respectively). However, property 175.2 may also easily be proved from 175.0 and 175.1 as shown below:

$Eval_{CA}(p \setminus cis)(t) \subseteq p \setminus cis$ from 175.0
$Eval_{CA}(p \setminus cis)(t) \subseteq Eval_{CA}(p)(t)$ from 175.1

So $Eval_{CA}(p \setminus cis)(t)$ does not contain values from *cis* and is a subset of $Eval_{CA}(p)(t)$. Put together we get 175.2 above.

Notice that the equations 42.1 ($Eval_{CA}(p \setminus cis)(t) = Eval_{CA}(p)(t) \setminus cis$) and 42.2 are no longer valid, but 42.1 has been replaced by the inclusion 175.2. The equation 42.1 was the foundation for the two approaches shown in section 5 for finding lattices satisfying a set of equations .

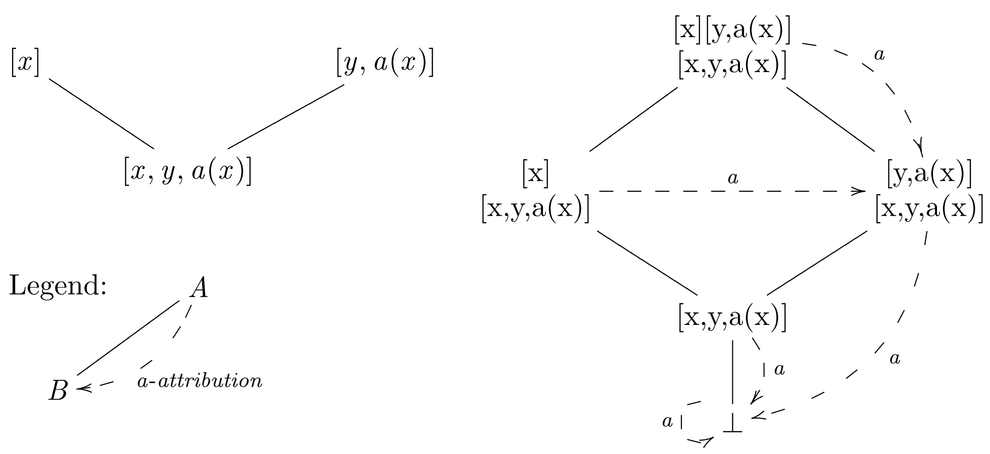


Figure 13: A partial order p_0 with attribution and the corresponding lattice $\mathcal{O}(p_0)$

Example Let $bset = \{x, y, a(x)\}$ be a set of basic concepts. Figure 13 shows a partial order $p_0 \subseteq \mathcal{P}(bset)$ and the corresponding lattice of downsets. In the lattice, attribution is shown as dashed arrows: there is an arrow from cis_1 to cis_2 iff $cis_2 = Attribution_{CA}(p_0)(a)(cis_1)$. As an example of attribution consider the a-attribution of $\{[x], [x, y, a(x)]\}$:

$$\begin{aligned}&Attribution_{CA}(p_0)(a)(\{[x], [x, y, a(x)]\})\\&= DownSetC(p_0)(\{[a(x)], [a(x), a(y), a(a(x))]\}) = \{[y, a(x)], [x, y, a(x)]\}\end{aligned}$$

Notice that the concept-intersection $[a(x), a(y), a(a(x))]$ is cut away when making the downset. Furthermore notice that $DownSetC(p_0)(\{[a(x)]\})$ does not contain $[a(x)]$ itself.

The table below shows the values of the terms $x, a(x)$ and y in p_0 and several subsets of p_0. The fields in the last three columns shows the set of concept-intersections constituting the value of a term. An empty field represents the empty set. A concept-intersection $mk\text{-}CI(\{b_1, b_2, \ldots, b_n\})$ is shown as $[b_1, b_2, \ldots, b_n]$.

p	partial order	$Eval_{CA}(p)(x)$	$Eval_{CA}(p)(a(x))$	$Eval_{CA}(y)$
p_0	$[x], [y, a(x)], [x, y, a(x)]$	$[x], [x, y, a(x)]$	$[y, a(x)], [x, y, a(x)]$	$[y, a(x)], [x, y, a(x)]$
p_1	$[x], [x, y, a(x)]$	$[x], [x, y, a(x)]$	$[x, y, a(x)]$	$[x, y, a(x)]$
p_2	$[x], [y, a(x)]$	$[x]$	$[y, a(x)]$	$[y, a(x)]$
p_3	$[y, a(x)], [x, y, a(x)]$	$[x, y, a(x)]$		$[y, a(x)], [x, y, a(x)]$
p_4	$[x]$	$[x]$		
p_5	$[y, a(x)]$			$[y, a(x)]$

Consider the evaluation of the terms $x, a(x)$ and y in p_0. For x we get

$$\begin{aligned} & Eval_{CA}(p_0)(x) = cValue_C(p_0)(x) \\ & = DownSetC(p_0)(\{mk\text{-}CI(\{x\})\}) = \{[x], [x, y, a(x)]\} \end{aligned}$$

For $a(x)$ we have

$$\begin{aligned} & Eval_{CA}(p_0)(a(x)) = Attribution_{CA}(p_0)(a)(Eval_{CA}(p_0)(x)) \\ & = Attribution_{CA}(p_0)(a)(\{[x], [x, y, a(x)]\}) = \{[y, a(x)], [x, y, a(x)]\} \end{aligned}$$

We can see that y and $a(x)$ evaluate to the same value, so the partial order p_0 satisfies the equation $y = a(x)$. From the table we can see that the two subset partial orders p_1 and p_2 also satisfies the equation $y = a(x)$. Notice however that the subset partial order p_3 does *not* satisfy the equation $y = a(x)$! Consequently, in the lattice algebra with attribution we no longer have a property corresponding to 57 for lattice algebras. That is, if we have a partial order which is a solution for a set of equations, we can no longer take for granted that all subset partial orders also are solutions.

Finally, $Eval_{CA}(p_4)(a(x)) = \{\}$ and $Eval_{CA}(p_5)(a(x)) = \{\}$, but

$$Eval_{CA}(p_4 \cup p_5)(a(x)) = Eval_l(p_1)(a(x)) = \{[x], [x, y, a(x)]\}$$

Consequently, in the considered algebra property 42.3 is no longer valid. □

10 Attribute Consistent Partial Orders

Let *bset* be a set of basic concepts. Until now we have considered all subsets $p \subseteq \mathcal{P}(bset)$ as equal candidates from which to construct the lattices $\mathcal{O}(p)$. However, after attribution has been introduced it turns out that if we restrict the subsets $p \subseteq \mathcal{P}(bset)$ to what we call attribute consistent partial orders, we get a lot of useful properties: The axiom concerning distribution of meet is fulfilled and it becomes possible to construct concept algebras along the same lines as used in sections 5 and 6 before attribution was introduced.

Informally, for a partial order to be attribute consistent, if it contains the attribution concept $\alpha(X)$ it must also contain the concept X of which it is an attribution. We need some auxiliary functions:

176.0 $\mathit{CbattrsCI} : A \rightarrow \mathit{CI} \rightarrow \mathit{At}\text{-set}$

.1 $\mathit{CbattrsCI}\,(\alpha)(\mathit{mk}\text{-}\mathit{CI}(\mathit{bs})) \triangleq \{\mathit{mk}\text{-}\mathit{At}(\alpha, b) \mid \mathit{mk}\text{-}\mathit{At}(\alpha', b) \in \mathit{bs} \cdot \alpha' = \alpha\}$

The function *CbattrsCI* gives the the complete set of basic α-attributions in the given concept-intersection ci. Consider a concept-intersection

$$ci = [\ldots \underbrace{\alpha(b_1) \ldots \alpha(b_i)}_{cbas} \ldots]$$

where the leftmost and rightmost dots denote basic concepts which are *not* basic α-attributions. Then $\mathit{CbattrsCI}(\alpha)(ci)$ is the complete set of basic α-attributions indicated by $cbas$ in the figure above. The complete set of basic α-attributions $\mathit{CbattrsCI}(\alpha)(ci)$ is non-empty for all the attributes that occur at the top-level in the concept-intersection ci:

177.0 $\forall\, ci : \mathit{CI}, \alpha : A \cdot \alpha \in \mathit{AttrsInCI}(ci) \;\Leftrightarrow\; \mathit{CbattrsCI}(\alpha)(ci) \neq \{\}$

This equivalence is used to formulate the precondition of the next function:

178.0 $\mathit{AttrArgCI} : A \rightarrow \mathit{CI} \rightarrow \mathit{CI}$

.1 $\mathit{AttrArgCI}\,(\alpha)(ci) \triangleq \mathit{mk}\text{-}\mathit{CI}(\{b \mid \mathit{mk}\text{-}\mathit{At}(\text{-}, b) \in \mathit{CbattrsCI}(\alpha)(ci)\})$

.2 pre $\alpha \in \mathit{AttrsInCI}(ci)$

Given a concept-intersection ci which has a non-empty complete set of basic α-attributions, then — loosely speaking — $\mathit{AttrArgCI}(\alpha)(ci)$ yields the concept-intersection the α-attribution of which gives the α-attribution part of ci. For instance, if $ci = [a, b, \alpha(a), \alpha(c), \alpha(d), e, f]$ then $\mathit{AttrArgCI}(\alpha)(ci) = [a, c, d]$. For a partial order p to be attribute consistent we will now require that if $ci \in p$, then we also have $[a, c, d] \in p$:

179.0 $\mathit{IsAttrConsistent} : \mathit{PO} \rightarrow \mathbb{B}$

.1 $\mathit{IsAttrConsistent}\,(p) \triangleq$

.2 $\quad \forall\, ci \in p \cdot \forall\, \alpha \in \mathit{AttrsInCI}(ci) \cdot \mathit{AttrArgCI}(\alpha)(ci) \in p$

We have a set of useful properties concerning the functions introduced above:

180.0 $\forall\, ci : \mathit{CI}, \alpha : A \cdot \mathit{ISA}_P(ci, \mathit{attrCI}(\alpha)(\mathit{AttrArgCI}(\alpha)(ci)))$

181.0 $\forall\, ci : \mathit{CI}, \alpha : A \cdot \mathit{AttrArgCI}(\alpha)(\mathit{attrCI}(\alpha)(ci)) = ci$

182.0 $\mathit{ISA}_P(ci_1, ci_2) \;\Rightarrow\; \mathit{CbattrsCI}(\alpha)(ci_2) \subseteq \mathit{Cbattrs}(\alpha)(ci_1)$

183.0 $\mathit{ISA}_P(ci_1, ci_2) \;\Rightarrow\; \mathit{ISA}_P(\mathit{AttrArgCI}(\alpha)(ci_1), \mathit{AttrArgCI}(\alpha)(ci_2))$

184.0 $\mathit{IsAttrConsistent}(p_1) \wedge \mathit{IsAttrConsistent}(p_2) \;\Rightarrow\; \mathit{IsAttrConsistent}(p_1 \cup p_2)$

185.0 $\mathit{IsAttrConsistent}(p_1) \wedge \mathit{IsAttrConsistent}(p_2) \;\Rightarrow\; \mathit{IsAttrConsistent}(p_1 \cap p_2)$

The properties 180 and 181 are illustrated in figure 14. Property 180 is about getting the α-attribution argument using *AttrArgCI* and then attributing the concept-intersection using *attrCI*. For example, if we reuse the small example above, we have $ci = [a, b, \alpha(a), \alpha(c), \alpha(d), e, f]$. Then getting the α-attribution argument gives $[a, c, d]$, and finally α-attributing this concept-intersection gives $[\alpha(a), \alpha(c), \alpha(d)]$ which is above ci in the partial order.

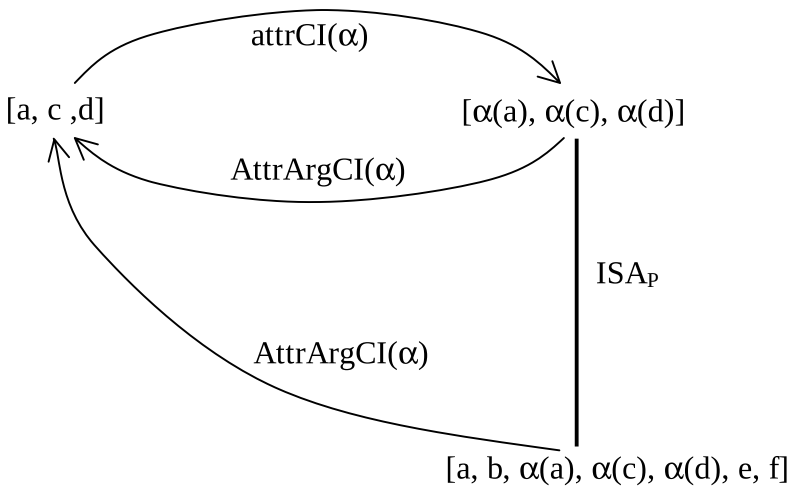


Figure 14: Using *attrCI* and *AttrArgCI*

Concerning 182 we know — from the definition of ISA_P (151) — that if $ISA_P(mk\text{-}CI(bs_1), mk\text{-}CI(bs_2))$ then $bs_2 \subseteq bs_1$. Consequently any subset of bs_2 must also be a subset of bs_1. Applying *CbattrsCI* (for any attribute α) to a concept-intersection yields a subset of its basic concepts. The proofs of these properties are rather straightforward (and are left to the reader).

10.1 Lemma for Attribute Consistent Partial Orders

For attribute consistent partial orders we have the following lemma, which is used in subsequent proofs:

186.0 $\forall\, cis : CI\text{-}\mathsf{set}, p : PO \cdot$
.1 $\forall\, \alpha \in AttrsInCIS(p) \cdot$
.2 $IsDownset(p)(cis) \wedge IsAttrConsistent(p) \;\Rightarrow$
.3 $\forall\, aci : CI \cdot aci \in Attribution_{CA}(p)(\alpha)(cis)$
.4 $\Rightarrow\; AttrArgCI(\alpha)(aci) \in cis \cap p$

In words: if p is an arbitrary attribute consistent partial order and cis a downset in p then all concept-intersections (aci) in the α-attribution of cis have there α-attribution argument in the part of cis which is also in p.

In order to make a proof we define an auxiliary function:

187.0 $ats : A \times Bset \rightarrow Bset$

.1 $ats\,(\alpha, bs) \triangleq \{mk\text{-}At(\alpha, b) \mid b \in bs\}$

We will need a few properties related to the functions *atCIs* and *ats* defined above. From the definitions of *atCIs* (171) and *ats* (187) it is easy to get

188.0 $\forall\, aci \in atCIs(\alpha)(cis) \cdot \exists\, mk\text{-}CI(bs) \in cis \cdot aci = mk\text{-}CI(ats(\alpha, bs))$

Furthermore, the definition of *CbattrsCI* (176) and *ats* (187) shows that

189.0 $\mathit{CbattrsCI}(\alpha, \mathit{mk}\text{-}\mathit{CI}(\mathit{ats}(\alpha, \mathit{bs}))) = \mathit{ats}(\alpha, \mathit{bs})$

The proof below is illustrated in figure 15. On the figure the partial order p is shown as the union of two sets p_1 and p_2. The set *cis* is shown to the left with bolded line and the order isomorphic set $\mathit{atCIs}(\alpha)(\mathit{cis})$ is shown to the right also with a bolded line.

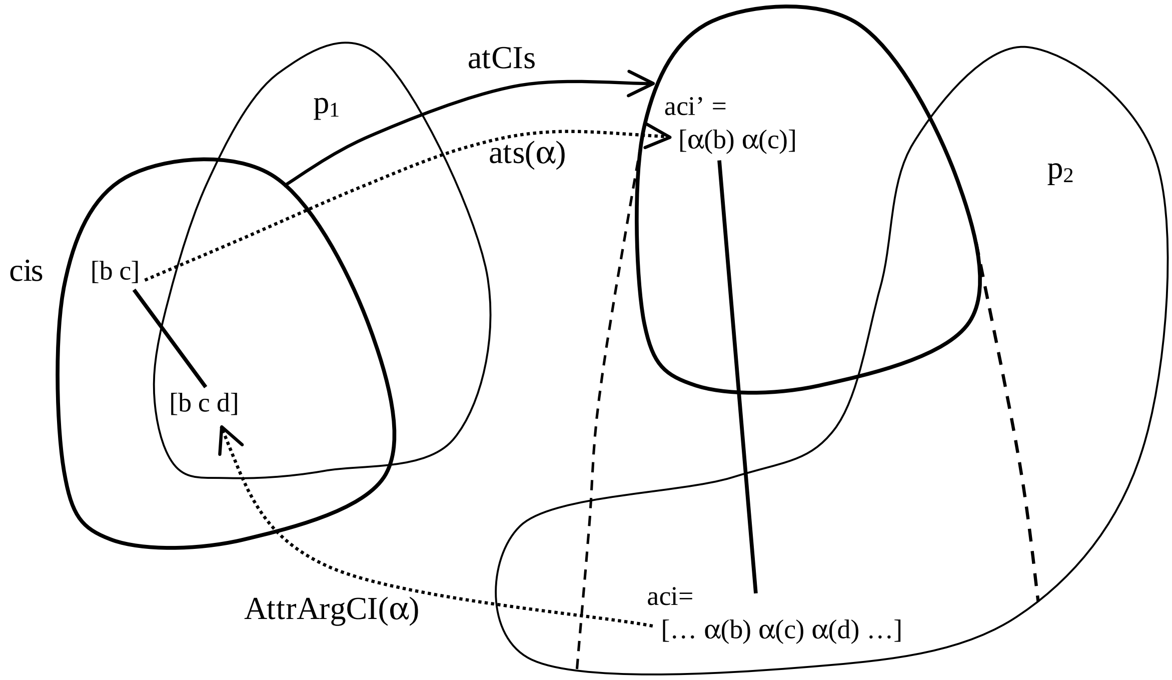


Figure 15: Illustration of proof for Attribute Consistent Partial Order lemma

So to prove 186 assume that *cis*, p, and α are arbitrary values such that

190.0 $\mathit{cis} : \mathit{CI}\text{-}\mathsf{set}, p : \mathit{PO}, \alpha \in \mathit{AttrsInCIS}(p)$

and furthermore assume the left hand side of the outermost implication above

191.0 $\mathit{IsDownset}(p)(\mathit{cis}) \wedge \mathit{IsAttrConsistent}(p)$

Next, corresponding to the innermost implication, we assume *aci* is an arbitrary concept-intersection such that

192.0 $\mathit{aci} \in \mathit{Attribution}_{CA}(p)(\alpha)(\mathit{cis})$

According to 172 this is equivalent to

193.0 $\mathit{aci} \in \mathit{DownSetC}(p)(\mathit{atCIs}(\alpha)(\mathit{cis}))$

From 193 and the definition of *DownSetC* we get

194.0 $\mathit{aci} \in p,$

.1 $\exists\, \mathit{aci}' \in \mathit{atCIs}(\alpha)(\mathit{cis}) \cdot \mathit{ISA}_P(\mathit{aci}, \mathit{aci}')$

Next, from 194.1 using 188 and the invariant in 148.1 we get

195.0 $\exists\, \mathit{mk}\text{-}\mathit{CI}(\mathit{bs}') \in \mathit{cis} \cdot \mathit{bs}' \neq \{\} \wedge \mathit{ISA}_P(\mathit{aci}, \mathit{mk}\text{-}\mathit{CI}(\mathit{ats}(\alpha, \mathit{bs}')))$

Now, let bs'' be the bs' that exists according to 195. We then have

196.0 $mk\text{-}CI(bs'') \in cis \wedge bs'' \neq \{\}$,
.1 $ISA_P(aci, mk\text{-}CI(ats(\alpha, bs''))$

If we now use 182 to 196.1 we get

197.0 $CbattrsCI(\alpha)(mk\text{-}CI(ats(\alpha, bs''))) \subseteq CbattrsCI(\alpha)(aci)$

Applying 189 now gives

198.0 $ats(\alpha, bs'') \subseteq CbattrsCI(\alpha)(aci)$

From 196.0 and the definition of *ats* (187)

199.0 $ats(\alpha, bs'') \neq \{\}$

Consequently, according to 198 we also have

200.0 $CbattrsCI(\alpha)(aci) \neq \{\}$

Now we use the fact that $aci \in p$ (194.0) and that the partial order p is attribute consistent (191). Applying the definition of *IsAttrConsistent* (179) then gives

201.0 $AttrArgCI(\alpha)(aci) \in p$

Next, if we apply 183 to 196.1 we get

202.0 $ISA_P(AttrArgCI(\alpha)(aci), AttrArgCI(\alpha)(mk\text{-}CI(ats(\alpha, bs''))))$

According to the definition of *AttrArgCI* (178) and *ats* (187) the right operand of ISA_P above can be reduced to $mk\text{-}CI(bs'')$, so we get

203.0 $ISA_P(AttrArgCI(\alpha)(aci), mk\text{-}CI(bs''))$

So now we know that $AttrArgCI(\alpha)(aci)$ is below $mk\text{-}CI(bs'')$, which according to 196.0 is in *cis*. From 191 we know that *cis* is a downset in p, so according to 10 we know that everything in p, which is below $mk\text{-}CI(bs'')$ is in *cis*. But from 201 we know that $AttrArgCI(\alpha)(aci) \in p$ so we have

204.0 $AttrArgCI(\alpha)(aci) \in cis$

which combined with 201 gives

205.0 $AttrArgCI(\alpha)(aci) \in cis \cap p$

10.2 Constructing Attribute Consistent Partial Orders

In subsequent sections we need a function to construct attribute consistent partial orders. Given a set *cis* of concept-intersections, there are two approaches to make the set attribute-consistent. Either we may *extend* the set *cis* to the smallest attribute-consistent partial order containing *cis*, or we may *restrict* the set *cis* to the greates attribute-consistent subset of *cis*. The existence of the mentioned smallest and greatest sets follows from 185 and 184 respectively.

The function *extCIS* defined below yields the smallest attribute consistent set of concept-intersections which contains the given set of concept-intersections *cis* by *extending* the set *cis* with the appropriate α-attribution arguments:

206.0 $extCIS : CI\text{-set} \rightarrow CI\text{-set}$
.1 $extCIS\ (cis) \triangleq$
.2 $\quad \mathsf{let}\ attrArg = \{AttrArgCI(\alpha)(ci) \mid ci \in cis, \alpha \in AttrsInCI(ci)\}\ \mathsf{in}$
.3 $\quad \mathsf{if}\ attrArg = \{\}\ \mathsf{then}\ cis\ \mathsf{else}\ cis \cup extCIS(attrArg)$

Examples

cis	$extCIS(cis)$
$\{[x, a(x), a(y), b(z)]\}$	$\{[x, a(x), a(y), b(z)], \underline{[x, y]}, \underline{[z]}\}$
$\{[x, a(b(y))], [x, y, b(x), b(a(y))]\}$	$\{[x, a(b(y))], [x, y, b(x), b(a(y))], \underline{[b(y)]}, \underline{[x, a(y)]}, \underline{[y]}\}$

□

Below is shown a few obvious facts about extension:

207.0 $\forall\, p : PO \cdot IsAttrConsistent(extCIS(p))$
.1 $\forall\, p_1, p_2 : PO \cdot IsAttrConsistent(p_1) \wedge p_2 \subseteq p_1 \;\Rightarrow\; extCIS(p_2) \subseteq p_1,$
.2 $\forall\, p_1, p_2 : PO \cdot p_2 \subseteq p_1 \;\Rightarrow\; extCIS(p_2) \subseteq extCIS(p_1)$

The function *restrCIS* defined below *restricts* the given set of concept-intersections to the greatest attribute consistent subset by removing those concept-intersections which doesn't have appropriate α-attribution arguments:

208.0 $restrCIS : CI\text{-set} \rightarrow CI\text{-set}$
.1 $restrCIS\ (cis) \triangleq$
.2 $\quad \mathsf{let}\ xx = \{ci \mid ci \in cis, \alpha \in AttrsInCI(ci) \cdot AttrArgCI(\alpha)(ci) \in cis\}\ \mathsf{in}$
.3 $\quad \mathsf{if}\ xx = \{\}\ \mathsf{then}\ cis\ \mathsf{else}\ restrCIS(cis \setminus xx)$

We will also need some facts about restriction:

209.0 $\forall\, p : PO \cdot IsAttrConsistent(restrCIS(p))$
.1 $\forall\, p_1, p_2 : PO \cdot IsAttrConsistent(p_2) \wedge p_2 \subseteq p_1 \;\Rightarrow\; p_2 \subseteq restrCIS(p_1),$
.2 $\forall\, p_1, p_2 : PO \cdot p_2 \subseteq p_1 \;\Rightarrow\; restrCIS(p_2) \subseteq restrCIS(p_1)$

11 Concept Algebras

In this section it is shown that if the partial order p is attribute consistent, i.e. $IsAttrConsistent(p)$ (as defined in 10) then $\mathcal{CA}(cset, aset, p)$ is a concept algebra satisfying the attribute axioms (in 135). It is also shown that the requirement about attribute consistent partial orders makes term values have properties which are close to those for lattice algebras before attribution was introduced (42).

We already know from section 9.1 that the attribute axioms concerning strictness and distribution of join are always fulfilled so we only have to consider distribution of meet.

11.1 Attribute Axiom: Distribution of Meet

As mentioned in section 9.1, it would be easy to prove the axiom for distribution of meet if we had a downset property for intersection corresponding to the downset property for union:

$$DownSet(p)(cis_1 \cup cis_2) = DownSet(p)(cis_1) \cup DownSet(p)(cis_2)$$

For intersection we only have

210.0 $DownSet(p)(cis_1 \cap cis_2) \subseteq DownSet(p)(cis_1) \cap DownSet(p)(cis_2)$

This is easy to prove, see section B.6. But in order to be able to prove that the “distribution of meet” axiom is fulfilled we need equality between the sets, not subset inclusion as in 210 above. Luckily, it turns out that although we don’t have a general downset property for intersection with equality, we have a *specialized* version, namely 212, which is described in section 11.2 below. But before investigating this property we use it to prove that the “distribution of meet” axiom is fulfilled:

Distribution of Meet:

211.0 $\alpha(Meet_L(cis_1, cis_2))$
.1 $= Attribution_{CA}(p)(\alpha)(Meet_L(cis_1, cis_2))$
.2 $= Attribution_{CA}(p)(\alpha)(cis_1 \cap cis_2)$
.3 $= DownSetC(p)(\{attrCI(\alpha)(ci) \mid ci \in cis_1 \cap cis_2\})$
from 171
.4 $= DownSetC(p)(atCIs(\alpha)(cis_1 \cap cis_2))$
from 212 below
.5 $= DownSetC(p)(atCIs(\alpha)(cis_1)) \cap DownSetC(p)(atCIs(\alpha)(cis_2))$
.6 $= Attribution_{CA}(p)(\alpha)(cis_1) \cap Attribution_{CA}(p)(\alpha)(cis_2)$
.7 $= Meet_L(Attribution_{CA}(p)(\alpha)(cis_1), Attribution_{CA}(p)(\alpha)(cis_2))$

11.2 The Downset Intersection Property

212.0 $\forall\, cis_1, cis_2 : CI\text{-set}, p : PO, \alpha : A \cdot$
.1 $IsDownset(p)(cis_1) \wedge IsDownset(p)(cis_2) \wedge IsAttrConsistent(p) \Rightarrow$
.2 $DownSetC(p)(atCIs(\alpha)(cis_1 \cap cis_2)) =$
.3 $DownSetC(p)(atCIs(\alpha)(cis_1)) \cap DownSetC(p)(atCIs(\alpha)(cis_2))$

This property is not at all obvious, but it is this property that actually makes it possible to use the simple representation of attributions as described in section 8.2. The proof of 212 is based on the lemma 186 for attribute consistent partial orders To make the proof we assume the left hand side of the implication in 212 and then prove the equality $LHS = RHS$ of the right hand side of the implication. The equality is proved by proving $LHS \subseteq RHS$ and $RHS \subseteq LHS$.

LHS ⊆ RHS We first prove $LHS \subseteq RHS$, i.e. we must prove

213.0 $DownSetC(p)(atCIs(\alpha)(cis_1 \cap cis_2))$
.1 $\subseteq DownSetC(p)(atCIs(\alpha)(cis_1)) \cap DownSetC(p)(atCIs(\alpha)(cis_2))$

From the definition of *atCIs* (171) this is equivalent to

214.0 $DownSetC(p)(atCIs(\alpha)(cis_1) \cap atCIs(\alpha)(cis_1))$
.1 $\subseteq DownSetC(p)(atCIs(\alpha)(cis_1)) \cap DownSetC(p)(atCIs(\alpha)(cis_2))$

which is fulfilled according to 210.

RHS $\subseteq$ LHS Next we must prove $RHS \subseteq LHS$. This is the difficult part of the proof of 212. From the definition of $attrCI$ (170) we have

215.0 $\forall\, ci : CI, cis : CI\text{-set} \cdot ci \in cis \;\Rightarrow\; attrCI(\alpha)(ci) \in atCIs(\alpha)(cis)$

In order to prove $RHS \subseteq LHS$ we now make assumptions corresponding to the left-hand side of the implication in 212. So assume that cis_1, cis_2, p and α are abitrary values such that

216.0 $cis_1, cis_2 : CI\text{-set}, p : PO, \alpha \in AttrsInCIS(p)$
.1 $IsDownset(p)(cis_1) \wedge IsDownset(p)(cis_2) \wedge IsAttrConsistent(p)$

Using these assumptions we must now prove

217.0 $DownSetC(p)(atCIs(\alpha)(cis_1)) \cap DownSetC(p)(atCIs(\alpha)(cis_2))$
.1 $\subseteq DownSetC(p)(atCIs(\alpha)(cis_1 \cap cis_2))$

If the left-hand side is the empty set then the subset inclusion is obvious, so in the sequel we assume that the left-hand side set is not the empty set. We prove the subset inclusion by showing that every element in the left-hand side set is also in the right-hand side set. So let $aci : CI$ be an arbitrary concept-intersection such that

218.0 $aci \in DownSetC(p)(atCIs(\alpha)(cis_1)) \cap DownSetC(p)(atCIs(\alpha)(cis_2))$

which is equivalent to

219.0 $aci \in DownSetC(p)(atCIs(\alpha)(cis_1))$
.1 $aci \in DownSetC(p)(atCIs(\alpha)(cis_2))$

From the assumption 216 and the lemma 186 for attribute consistent partial orders we get

220.0 $AttrArgCI(\alpha)(aci) \in cis_1$
.1 $AttrArgCI(\alpha)(aci) \in cis_2$

Next, applying 215 gives

221.0 $attrCI(\alpha)(AttrArgCI(\alpha)(aci)) \in atCIs(\alpha)(cis_1)$
.1 $attrCI(\alpha)(AttrArgCI(\alpha)(aci)) \in atCIs(\alpha)(cis_2)$

and consequently also

222.0 $attrCI(\alpha)(AttrArgCI(\alpha)(aci)) \in atCIs(\alpha)(cis_1) \cap atCIs(\alpha)(cis_2)$

Finally 180 gives us

223.0 $ISA_P(aci, attrCI(\alpha)(AttrArgCI(\alpha, aci)))$

So aci is below a point which is in $atCIs(\alpha)(cis_1) \cap atCIs(\alpha)(cis_2)$, and from 219 we know that $aci \in p$, consequently we have

224.0 $aci \in DownSetC(p)(atCIs(\alpha)(cis_1) \cap atCIs(\alpha)(cis_2))$

11.3 The Value of Terms in Concept Algebras.

The value of terms in the algebra $\mathcal{CA}(cset, aset, p)$ was defined in section 9.2 for all partial orders $p \subseteq \mathcal{P}(bset)$. If we only consider evaluation of terms in concept algebras, i.e. we restrict the partial orders $p \subseteq \mathcal{P}(bset)$ to those which are attribute consistent, then we have properties which are close to the essential properties in 42 as we will see in the following. We will need the property for attribution in concept algebras described below:

Attribution in Concept Algebras: Let $bset$ be a set of basic concepts, p and p_1 subsets of $\mathcal{P}(bset)$, cis a subset of p and finally α an attribute, we then have

225.0 $IsAttrConsistent(p_1) \wedge IsDownset(p)(cis) \wedge p_1 \subseteq p$
.1 $\Rightarrow Attribution_{CA}(p)(\alpha)(cis \cap p_1) \cap p_1 = Attribution_{CA}(p)(\alpha)(cis) \cap p_1$

A proof of property 225 is also based on the lemma 186 for attribute consistent partial orders and follows here:
We prove the equality of the two sets by proving that the two sets are subsets of each other. So for the two proofs assume that p and p_1 are partial orders and cis a set of concept-intersections such that

226.0 $IsAttrConsistent(p) \wedge IsAttrConsistent(p_1) \wedge IsDownset(p)(cis) \wedge p_1 \subseteq p$

Proof of $LHS \subseteq RHS$: From 173.1 we get immediately

227.0 $Attribution_{CA}(p)(\alpha)(cis \cap p_1) \subseteq Attribution_{CA}(p)(\alpha)(cis)$

and consequently also

228.0 $Attribution_{CA}(p)(\alpha)(cis \cap p_1) \cap p_1 \subseteq Attribution_{CA}(p)(\alpha)(cis) \cap p_1$

Proof of $RHS \subseteq LHS$: We must prove

229.0 $Attribution_{CA}(p)(\alpha)(cis) \cap p_1 \subseteq Attribution_{CA}(p)(\alpha)(cis \cap p_1) \cap p_1$

If $Attribution_{CA}(p)(\alpha)(cis) \cap p_1 = \{\}$ then the inclusion is obvious, so in the sequel assume that

230.0 $Attribution_{CA}(p)(\alpha)(cis) \cap p_1 \neq \{\}$

Then assume that aci is an arbitrary concept-intersection such that

231.0 $aci \in Attribution_{CA}(p)(\alpha)(cis) \cap p_1$

So, according to the equation 172 for $Attribution_{CA}$ we have

232.0 $aci \in DownSetC(p)(atCIs(\alpha)(cis)) \cap p_1$

Hence

233.0 $aci \in p_1$
.1 $aci \in DownSetC(p)(atCIs(\alpha)(cis))$

From the fact that $p_1 \subseteq p$ and the downset property 15 we also have

234.0 $aci \in DownSetC(p_1)(atCIs(\alpha)(cis))$

From the assumption 226 and the definition of $IsDownSet$ (9) we have

235.0 $IsDownSet(p_1)(cis)$

Next, applying 234 and 235 to the lemma 186 for attribute consistent partial orders gives

236.0 $AttrArgCI(\alpha)(aci) \in cis \cap p_1$

Next, applying 215 gives

237.0 $attrCI(\alpha)(AttrArgCI(\alpha)(aci)) \in atCIs(\alpha)(cis \cap p_1)$

According to 180 we have

238.0 $ISA_P(aci, attrCI(\alpha)(AttrArgCI(\alpha, aci)))$

From 237 and 238 we may now conclude that

239.0 $aci \in DownSetC(p)(atCIs(\alpha)(cis \cap p_1))$

which combined with 233 gives

240.0 $aci \in DownSetC(p)(atCIs(\alpha)(cis \cap p_1)) \cap p_1$

Finally, using the definition of $Attribution_{CA}$ (169) gives

241.0 $aci \in Attribution_{CA}(p)(\alpha)(cis \cap p_1) \cap p_1$

Term Value Properties in Concept Algebras: Let *bset* be a set of basic concepts, t a term, and let p_1 and p_2 be subsets of $\mathcal{P}(bset)$, we then have

242.0 $IsAttrConsistent(p_1) \Rightarrow$
.1 $p_1 \subseteq p_2 \Rightarrow Eval_{CA}(p_1)(t) = Eval_{CA}(p_2)(t) \cap p_1$

243.0 $IsAttrConsistent(p_1) \wedge IsAttrConsistent(p_2) \Rightarrow$
.1 $Eval_{CA}(p_1 \cup p_2)(t) = Eval_{CA}(p_1)(t) \cup Eval_{CA}(p_2)(t)$

244.0 $IsAttrConsistent(p_1) \wedge IsAttrConsistent(p_2) \Rightarrow$
.1 $Eval_{CA}(p_1 \cap p_2)(t) = Eval_{CA}(p_1)(t) \cap Eval_{CA}(p_2)(t)$

Properties 243 and 244 are easily proved from 242 and 175.0 in the same way that 42.3 and 42.4 were proved from 42.0 and 42.2 as shown in section B.3

A proof of 242 is based on the property 225 for attribution and follows here. So let t be a term and p_1 and p_2 subsets of $\mathcal{P}(bset)$ and assume the left-hand side of both implications in 242 above:

245.0 $IsAttrConsistent(p_1)$
.1 $p_1 \subseteq p_2$

We prove the right-hand side of the implication by structural induction on the term structure:

BOTTOM:

$$Eval_{CA}(p_1)(\text{BOTTOM}) = \{\} = Eval_{CA}(p_2)(\text{BOTTOM}) \cap p_1$$

TOP:

$$Eval_{CA}(p_1)(\text{TOP}) = p_1$$

from 245.1

$$= p_2 \cap p_1 = Eval_{CA}(p_2)(\text{TOP}) \cap p_1$$

Named concept c:

$$Eval_{CA}(p_1)(c) = cValue_C(p_1)(c)$$
$$= DownSet(p_1)(\{mk\text{-}CI(\{c\})\})$$

from 15

$$= DownSet(p_2)(\{mk\text{-}CI(\{c\})\}) \cap p_1$$
$$= cValue_C(p_2)(c) \cap p_1$$
$$= Eval_{CA}(p_2)(c) \cap p_1$$

Next, in the induction steps, when considering compound terms, assume 242 is true for the sub-terms (the induction hypothesis):

$mk\text{-}Join(t_1, t_2)$:

$$
\begin{array}{ll}
Eval_{CA}(p_1)(mk\text{-}Join(t_1,t_2)) & \\
 & \text{from } 174 \\
= Join_L(Eval_{CA}(p_1)(t_1), Eval_{CA}(p_1)(t_2)) & \\
 & \text{from } 30 \\
= Eval_{CA}(p_1)(t_1) \cup Eval_{CA}(p_1)(t_2) & \\
 & \text{from induction hypothesis} \\
= (Eval_{CA}(p_2)(t_1) \cap p_1) \cup (Eval_{CA}(p_2)(t_2) \cap p_1) & \\
= (Eval_{CA}(p_2)(t_1) \cup Eval_{CA}(p_2)(t_2)) \cap p_1 & \\
= Join_L(Eval_{CA}(p_2)(t_1), Eval_{CA}(p_2)(t_2)) \cap p_1 & \\
 & \text{from } 174 \\
= Eval_{CA}(p_2)(mk\text{-}Join(t_1,t_2)) \cap p_1 &
\end{array}
$$

$mk\text{-}Meet(t_1, t_2)$: Similar to the proof for Join.

$mk\text{-}Attr(\alpha, t)$:

$$
\begin{array}{ll}
Eval_{CA}(p_1)(mk\text{-}Attr(\alpha,t)) & \\
 & \text{from } 174 \\
= Attribution_{CA}(p_1)(\alpha)(Eval_{CA}(p_1)(t)) & \\
 & \text{from induction hypothesis} \\
= Attribution_{CA}(p_1)(\alpha)(Eval_{CA}(p_2)(t) \cap p_1) & \\
 & \text{from } 173.3 \\
= Attribution_{CA}(p_2)(\alpha)(Eval_{CA}(p_2)(t) \cap p_1) \cap p_1 & \\
 & \text{from } 225 \\
= Attribution_{CA}(p_2)(\alpha)(Eval_{CA}(p_2)(t)) \cap p_1 & \\
 & \text{from } 174 \\
= Eval_{CA}(p_2)(mk\text{-}Attr(\alpha,t)) \cap p_1 &
\end{array}
$$

11.4 A Concept Algebra is a Generated Algebra.

Assume *cset* is a set of concepts, *aset* a set of attributes, *bset* a finite set of basic concepts, and $p \subseteq \mathcal{P}(bset)$ is a finite partial order of concept-intersections such that $Conforms(cset, aset)(p)$. We then have

Lemma The algebra $\mathcal{CA}(cset, aset, p)$ is a generated algebra if and only if p is attribute-consistent.

12 Concept Algebras Satisfying a Set of Equations

Assume *cset* is a set of concepts, *aset* a set of attributes, and *bset* a finite set of basic concepts build from *cset* and *aset*. We now consider how to compute an attribute consistent partial order $p \subseteq \mathcal{P}(bset)$ such that the concept algebra $\mathcal{CA}(cset, aset, p)$ satisfies a given set of (ground) equations $eqs : Eq\text{-}\mathsf{set}$ (besides the set of basic Concept Algebra equations). Thanks to the properties 242 and 243 we are able to proceed almost as in section 5.

In the sequel we call an attribute consistent partial order, in which a set of equations is satisfied, a solution for the set of equations. Let

246.0 $IsEqsSol_{CA} : Eq\text{-set} \rightarrow PO \rightarrow \mathbb{B}$
.1 $IsEqsSol_{CA}\,(eqs)(p) \triangleq$
.2 $\quad IsAttrConsistent(p) \wedge$
.3 $\quad \forall\, mk\text{-}Eq(t_1, t_2) \in eqs \cdot Eval_{CA}(p)(t_1) = Eval_{CA}(p)(t_2)$

So we are looking for a way to compute the set of solutions:

247.0 $\{p \mid p \subseteq \mathcal{P}(cset) \cdot IsEqsSol_{CA}(eqs)(p)\}$

Assume we have an equation $t_1 = t_2$. The property below concerns the relation between solutions for such an equation.

248.0 $\forall\, p_1, p_2 : PO, t_1, t_2 : Term \cdot$
.1 $\qquad IsAttrConsistent(p_1) \wedge$
.2 $\qquad p_1 \subseteq p_2 \wedge Eval_{CA}(p_2)(t_1) = Eval_{CA}(p_2)(t_2) \;\Rightarrow\; Eval_{CA}(p_1)(t_1) = Eval_{CA}(p_1)(t_2)$

In words, if we have a solution p_2 to an equation $t_1 = t_2$ then every attribute consistent subset p_1 of that solution is also a solution. Consequently, we only need to compute the maximal partial order satisfying the equation. Property 248 may easily be proved from 242:

Assume the left-hand side of the implication in 248 above, i.e.

249.0 $IsAttrConsistent(p_1)$,
.1 $p_1 \subseteq p_2$,
.2 $Eval_{CA}(p_2)(t_1) = Eval_{CA}(p_2)(t_2)$

Then we have the following equalities.

$$
\begin{aligned}
& Eval_{CA}(p_1)(t_1) && \\
& && \text{assumptions 249 and 242} \\
&= Eval_{CA}(p_2)(t_1) \cap p_1 && \\
& && 249.2 \\
&= Eval_{CA}(p_2)(t_2) \cap p_1 && \\
& && \text{assumptions 249 and 242} \\
&= Eval_{CA}(p_1)(t_2) &&
\end{aligned}
$$

Property 248 can easily be extended to a set of equations:

250.0 $\forall\, p_1, p_2 : PO, eqs : Eq\text{-set} \cdot$
.1 $\qquad IsAttrConsistent(p_1) \wedge$
.2 $\qquad p_1 \subseteq p_2 \wedge IsEqsSol_{CA}(eqs)(p_2) \;\Rightarrow\; IsEqsSol_{CA}(eqs)(p_1)$

Just as in section 5 we have two approaches for constructing solutions to a set of equations, namely the additive and the subtractive method.

12.1 The Additive Method

Let $t_1 = t_2$ be an equation, and let p_1 and p_2 be two attribute consistent partial orders in which the equation is satisfied, i.e.

251.0 $Eval_{CA}(p_1)(t_1) = Eval_{CA}(p_1)(t_2)$ and
.1 $Eval_{CA}(p_2)(t_1) = Eval_{CA}(p_2)(t_2)$

Using first 243 and then the equations in 251 we get

$$\begin{aligned}&Eval_{CA}(p_1 \cup p_2)(t_1)\\&= Eval_{CA}(p_1)(t_1) \cup Eval_{CA}(p_2)(t_1) = Eval_{CA}(p_1)(t_2) \cup Eval_{CA}(p_2)(t_2)\\&= Eval_{CA}(p_1 \cup p_2)(t_2)\end{aligned}$$

Hence

252.0 $IsAttrConsistent(p_1) \wedge IsAttrConsistent(p_2) \;\Rightarrow$
.1 $\quad Eval_L(p_1 \cup p_2)(t_1) = Eval_L(p_1 \cup p_2)(t_2)$

In words, if an equation is satisfied in two attribute consistent partial orders p_1 and p_2 it will also be satisfied in the union of these partial orders. Notice, that the union of two attribute consistent partial orders is also attribute consistent. This can easily be extended to a set of equations:

253.0 $\forall\, eqs : Eq\text{-set}, p_1, p_2 : PO \cdot$
.1 $\quad IsEqsSol_{CA}(eqs)(p_1) \wedge IsEqsSol_{CA}(eqs)(p_2) \;\Rightarrow\; IsEqsSol_{CA}(eqs)(p_1 \cup p_2)$

The property above shows us that if we have found two small solutions we can get a new bigger solution by making the union of the small solutions. So we can construct a solution by making the union af small solutions. Now the smallest possible solutions are the smallest subsets $p \subseteq \mathcal{P}(bset)$ which are attribute consistent. These subsets are constructed using the function *extCIS* (def. 206) defined in section 10.2. Given the set of basic concepts, the set of building blocks for a solution is

254.0 $SolBB : B\text{-set} \rightarrow CI\text{-set-set}$
.1 $SolBB\,(bset) \triangleq \bigcup\{extCIS(\{mk\text{-}CI(bs)\}) \mid bs : B\text{-set} \cdot bs \subseteq bset\}$

So given a set of equations we can now build the maximal solution from all the building blocks which are solutions to the equations:

255.0 $MaxPO_{CA} : B\text{-set} \rightarrow Eq\text{-set} \rightarrow PO$
.1 $MaxPO_{CA}\,(bset)(eqs) \triangleq \bigcup\{p \mid p \in SolBB(bset) \cdot IsEqsSol_{CA}(eqs)(p)\}$

According to the previous exposition we have

256.0 let $pmax = MaxPO_{CA}(bset)(eqs)$ in
.1 $pmax \subseteq \mathcal{P}(bset) \wedge IsEqsSol_{CA}(eqs)(pmax)$

Example We use the example in section 9.2 and figure 13. So the set of basic concepts is $bset = \{x, y, a(x)\}$. In the table below the first column contains all the possible concept-intersections, one in each row. Each row in the second column contains the building block

$extCI(ci)$ containing the corresponding ci.

ci	$p = extCIS(\{ci\})$	$Eval_{CA}(p)(y)$	$Eval_{CA}(p)(x)$	$Eval_{CA}(p)(a(x))$
$[x]$	$[x]$		$[x]$	
$[y]$	$[y]$	$[y]$		
$[a(x)]$	$[a(x)], [x]$		$[x]$	$[a(x)]$
$[x, y]$	$[x, y]$	$[x, y]$	$[x, y]$	
$[x, a(x)]$	$[x, a(x)], [x]$		$[x], [x, a(x)]$	$[x, a(x)]$
$[y, a(x)]$	$[y, a(x)], [x]$	$[y, a(x)]$	$[x]$	$[y, a(x)]$
$[x, y, a(x)]$	$[x, y, a(x)], [x]$	$[x, y, a(x)]$	$[x, y, a(x)], [x]$	$[x, y, a(x)]$

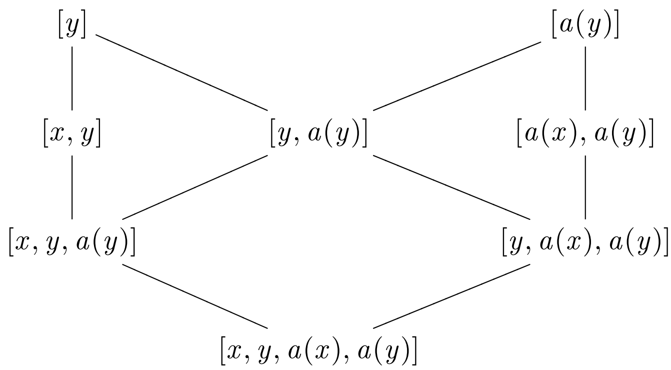
We want to find the maximal solution to the equation $y = a(x)$. So in the next three columns each row contains the values of the terms y, x and $a(x)$ in the partial order constituted by the corresponding building block. We can see that the equation is satisfied in the partial orders in the two last rows. Hence

$$MaxPO_{CA}(bset)(\{y = a(x)\})$$
$$= \{[y, a(x)], [x]\} \cup \{[x, y, a(x)], [x]\} = \{[x], [y, a(x)], [x, y, a(x)]\}$$

This is the solution shown in figure 13. □

Example In this example the set of basic concepts is $bset = \{x, y, a(x), a(y)\}$ and we have one equation $x \leq y$. In figure 16 the table has a row for each building block (as defined by $SolBB(bset)$). Column 3 and 4 shows the value of the terms x, $a(x)$ and y, $a(y)$. In the last column a + indicates that the equation $x \leq y$ is satisfied in the corresponding building block partial order. The reader can easily verify that for these partial orders we also have $a(x) \leq a(y)$ as required by the monotonicity rule 135.

According to 255 we get the maximal solution $pmax = MaxPO_{CA}(bset)(eqs)$ by making the union of all the building block partial orders for which $x \leq y$. This partial order is shown in the figure below.



□

12.2 The Subtractive Method

Let $t_1 = t_2$ be an equation, let $p_c = \mathcal{P}(bset)$ and let

$$cis_1 = Eval_{CA}(p_c)(t_1) \text{ and } cis_2 = Eval_{CA}(p_c)(t_2)$$

ci	$p = extCIS(\{ci\})$	$\frac{\{Eval_{CA}(p)(x)\}}{\{Eval_{CA}(p)(a(x))\}}$	$\frac{\{Eval_{CA}(p)(y)\}}{\{Eval_{CA}(p)(a(y))\}}$	$x \leq y$
$[x]$	$[x]$	$\frac{\{[x]\}}{\{\}}$		-
$[y]$	$[y]$		$\frac{\{[y]\}}{\{\}}$	+
$[x, y]$	$[x, y]$	$\frac{\{[x,y]\}}{\{\}}$	$\frac{\{[x,y]\}}{\{\}}$	+
$[a(x)]$	$[a(x)], [x]$	$\frac{\{[x]\}}{\{[a(x)]\}}$		-
$[a(y)]$	$[a(y)], [y]$		$\frac{\{[y]\}}{\{[a(y)]\}}$	+
$[x, a(x)]$	$[x, a(x)], [x]$	$\frac{\{[x,a(x)],[x]\}}{\{\}}$		-
$[y, a(x)]$	$[y, a(x)], [x]$	$\frac{\{[x]\}}{\{[y,a(x)]\}}$	$\frac{\{[y,a(x)]\}}{\{\}}$	-
$[x, a(y)]$	$[x, a(y)], [y]$	$\frac{\{[x,a(y)]\}}{\{\}}$	$\frac{\{[y]\}}{\{[x,a(y)]\}}$	-
$[y, a(y)]$	$[y, a(y)], [y]$		$\frac{\{[y,a(y)],[y]\}}{\{[y,a(y)]\}}$	+
$[x, y, a(x)]$	$[x, y, a(x)], [x]$	$\frac{\{[x,y,a(x)],[x]\}}{\{[x,y,a(x)]\}}$	$\frac{\{[x,y,a(x)]\}}{\{\}}$	-
$[x, y, a(y)]$	$[x, y, a(y)], [y]$	$\frac{\{[x,y,a(y)]\}}{\{\}}$	$\frac{\{[x,y,a(y)],[y]\}}{\{[x,y,a(y)]\}}$	+
$[a(x), a(y)]$	$[a(x), a(y)], [x, y]$	$\frac{\{[x,y]\}}{\{[a(x),a(y)]\}}$	$\frac{\{[x,y]\}}{\{[a(x),a(y)]\}}$	+
$[x, a(x), a(y)]$	$[x, a(x), a(y)], [x, y]$	$\frac{\{[x,a(x),a(y)],[x,y]\}}{\{[x,a(x),a(y)]\}}$	$\frac{\{[x,y]\}}{\{[x,a(x),a(y)]\}}$	-
$[y, a(x), a(y)]$	$[y, a(x), a(y)], [x, y]$	$\frac{\{[x,y]\}}{\{[y,a(x),a(y)]\}}$	$\frac{\{[y,a(x),a(y)],[x,y]\}}{\{[y,a(x),a(y)]\}}$	+
$[x, y, a(x), a(y)]$	$[x, y, a(x), a(y)], [x, y]$	$\frac{\{[x,y,a(x),a(y)],[x,y]\}}{\{[x,y,a(x),a(y)]\}}$	$\frac{\{[x,y,a(x),a(y)],[x,y]\}}{\{[x,y,a(x),a(y)]\}}$	+

Figure 16: The set of building blocks for $bset = \{x, y, a(x), a(y)\}$. Column 3 and 4 shows the value of x,$a(x)$ and y,$a(y)$ in the building block partial order.

then

$$eqrej0 = (cis_1 \cup cis_2) \setminus (cis_1 \cap cis_2)$$

is the set of all the concept-intersections not occurring in both cis_1 and cis_2, i.e. the set of concept-intersections causing t_1 and t_2 to evaluate to different values. So these concept-intersections must be removed from p_c. However — compared to the subtractive method described in section 5.2 — we must now assure that the remaining set of concept-intersections is attribute-consistent. So let p' be an arbitrary attribute-consistent sub-set of p_c not containing $eqrej0$:

$$p' \subseteq p_c \setminus eqrej0 \wedge IsAttrConsistent(p')$$

We are interested in the part of p' which is in $cis_1 \cup cis_2$. Using the fact that $p' \subseteq p_c \setminus eqrej0$ we now have

257.0 $p' \cap (cis_1 \cup cis_2)$
.1 $\subseteq (p_c \setminus eqrej0) \cap (cis_1 \cup cis_2)$
.2 $= ((cis_1 \cup cis_2) \setminus eqrej0)$
.3 $= (cis_1 \cup cis_2) \setminus ((cis_1 \cup cis_2) \setminus (cis_1 \cap cis_2))$
.4 $= cis_1 \cap cis_2$

So the part of p' which is in $cis_1 \cup cis_2$ is also in $cis_1 \cap cis_2$. We can use this to evaluate t_1

and t_2 in the partial order p':

258.0 $Eval_{CA}(p')(t_1)$

using 242 gives

.1 $= Eval_{CA}(p_c)(t_1) \cap p'$

.2 $= cis_1 \cap p'$

using 257 gives

.3 $\subseteq cis_1 \cap cis_2$

.4 $\subseteq cis_2$

Using this set-inclusion now gives

$Eval_{CA}(p')(t_1)$

from above

$= cis_1 \cap p'$

using the set-inclusion 258 above

$= cis_1 \cap cis_2 \cap p'$

We can of course do the same with t_2, so

$$Eval_{CA}(p')(t_1) = Eval_{CA}(p')(t_2) = cis_1 \cap cis_2 \cap p'$$

Consequently, the equation $t_1 = t_2$ is satisfied in any attribute-consistent partial order p' not containing the concept-intersections rejected by the equation.

Considering a set of equations $\{eq_1, eq_2, \ldots, eq_n\}$, each equation eq_i gives rise to a set of concept-intersections $eqrej0_i$ as defined above, which must not be in the resulting partial order. So let p' be an attribute-consistent partial order not containing any of these rejected sets:

$$p' \subseteq p_c \setminus (eqrej0_1 \cup eqrej0_2 \cup \ldots \cup eqrej_n)$$
$$\wedge\ IsAttrConsistent(p')$$

Then, according to the argumentation above, each equation eq_i is satisfied in p'. Finally, to get the maximal solution, we use *restrCIS* to get the attribute-consistent subset:

259.0 $EqRej : CI\text{-set} \rightarrow Eq \rightarrow CI\text{-set}$

.1 $EqRej\ (p)(mk\text{-}Eq(t_1, t_2)) \triangleq$

.2 $\quad \mathsf{let}\ cis_1 = Eval_{CA}(p)(t_1),$

.3 $\quad\quad cis_2 = Eval_{CA}(p)(t_2)\ \mathsf{in}$

.4 $\quad (cis_1 \cup cis_2) \setminus (cis_1 \cap cis_2)$

260.0 $MaxPO_{CA} : B\text{-set} \rightarrow Eq\text{-set} \rightarrow PO$

.1 $MaxPO_{CA}\ (bset)(eqs) \triangleq$

.2 $\quad \mathsf{let}\ p_c = \mathcal{P}(bset)\ \mathsf{in}$

.3 $\quad \mathsf{let}\ rejected = \bigcup\{EqRej(p_c)(eq) \mid eq \in eqs\}\ \mathsf{in}$

.4 $\quad restrCIS(p_c \setminus rejected)$

12.3 The Set of Concept Algebras Satisfying a Set of Equations

Let *cset* and *aset* be the set of concepts and attributes and *bset* a set of basic concepts created from *cset* and *aset*. Let $pmax = MaxPO_{CA}(bset)(eqs)$. This is the greatest partial order satisfying the set *eqs* of equations. According to 250 all attribute consistent subsets of this partial order are also solutions to the set of equations. We define the corresponding set of concept algebra solutions

261.0 $C_{CA\text{-}eqs} = \{\mathcal{CA}(cset, aset, p) \mid p : PO \cdot p \subseteq pmax \wedge IsAttrConsistent(p)\}$

As in section 5.3 we have in $C_{CA\text{-}eqs}$ two solutions of special interest, namely the initial concept algebra $\mathcal{CA}(cset, aset, pmax)$ and the most disjoint concept algebra $\mathcal{CA}(cset, aset, pmdsj)$.

12.3.1 The Initial Concept Algebra

In the class $C_{CA\text{-}eqs}$ of Concept Algebras defined above $\mathcal{CA}(cset, aset, pmax)$ is the initial concept algebra or most general concept algebra. This is equivalent to saying that for each attribute consistent $p \subseteq pmax$ there is a unique homomorphism from $\mathcal{CA}(cset, aset, pmax)$ to $\mathcal{CA}(cset, aset, p)$. For a given $p \subseteq pmax$, that homomorphism is defined by the function h: $h : \mathcal{O}(pmax) \to \mathcal{O}(p)$ such that

$$h(cis) = cis \cap p, cis \in \mathcal{O}(pmax)$$

Now

$$h(Join_C(cis1, cis2)) = (cis1 \cup cis2) \cap p = (cis1 \cap p) \cup (cis2 \cap p) = Join_C(h(cis1), h(cis2))$$
$$h(Meet_C(cis1, cis2)) = (cis1 \cap cis2) \cap p = (cis1 \cap p) \cap (cis2 \cap p) = Meet_C(h(cis1), h(cis2))$$

According to the definition of constants (174.8, 168) we have

$$\begin{aligned}
&h(c_{pmax}()) = cValue_C(pmax)(c) \cap p = DownSet(pmax)(\{mk\text{-}CI(\{c\})\}) \cap p \\
&= \{mk\text{-}CI(cs) \mid mk\text{-}CI(cs) \in pmax \cdot c \in cs\} \cap p \\
&= \{mk\text{-}CI(cs) \mid mk\text{-}CI(cs) \in pmax \cap p \cdot c \in cs\} \\
&= cValue_C(p)(c) = c_p()
\end{aligned}$$

Similar for the constants TOP and BOTTOM.
Finally we must consider the attribution operation.

$$\begin{aligned}
&h(Attribution_{CA}(pmax)(\alpha)(cis)) = Attribution_{CA}(pmax)(\alpha)(cis) \cap p \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{from } 173.3 \\
&= Attribution_{CA}(p)(\alpha)(cis)
\end{aligned}$$

Consequently h is a homomorphism.

One should notice, that although $\mathcal{CA}(cset, aset, pmax)$ is initial in the class $C_{CA\text{-}eqs}$, $\mathcal{CA}(cset, aset, pmax)$ is certainly not a freely generated algebra. Due to the finite attribution approach we know that for some level of attribution nesting the term $a(a(\ldots a(c)\ldots))$ will evaluate to $\perp$. This equality in $\mathcal{CA}(cset, aset, pmax)$ is not derivable from the set of equations from which $pmax$ was computed.

13 The Concept Algebra of Anti-chains

In this section we define the concept algebra of anti-chains corresponding to the lattice of anti-chains defined in 4.1. Given a concept algebra $\mathcal{CA}(cset, aset, p)$ we can easily define the isomorphic lattice where the elements are the antichain-part of the elements in $\mathcal{O}(p)$. As in section 4.1, we denote the set of new lattice elements $\mathcal{N}(p)$. It is defined as in 43. The corresponding algebra is

262.0 $\mathcal{NCA}(cset, aset, p) =$
.1 $\quad < \mathcal{N}(p); Join_N, Meet_N, A_N(p)(aset), C_N(p)(cset), \text{TOP}_N, \text{BOTTOM}_N >$

where $Join_N$, $Meet_N$, $C_N(p)(cset)$, $\textsc{top}_N$, $\textsc{bottom}_N$ and also ISA_N are as defined in section 4.1. Compared to section 4.1 we now also have to redefine the unary operators corresponding to the set of attributes $aset$:

$$A_N(p)(aset) = \{Attribution_N(p)(\alpha) \mid \alpha \in aset\}$$

where

263.0 $Attribution_N : PO \rightarrow A \rightarrow CI\text{-set} \rightarrow CI\text{-set}$

.1 $Attribution_N\,(p)(\alpha)(ac) \triangleq AntiCh(Attribution_{CA}(p)(\alpha)(DownSet(p)(ac)))$

Here $Attribution_N$ is defined from $Attribution_{CA}$ (def. 169) by converting between down-sets and anti-chains using $DownSet$ and $AntiCh$ (in the same way as was done in section 4.1). Given the definitions of the operations in $\mathcal{NCA}(cset, aset, p)$ we can now define the evaluation of terms in the new algebra:

264.0 $Eval_{NCA} : PO \rightarrow Term \rightarrow CI\text{-set}$

.1 $Eval_{NCA}\,(p)(t) \triangleq$

.2 cases t :

.3 $mk\text{-}Join(t_1, t_2) \rightarrow Join_N(p)(Eval_{NCA}(p)(t_1), Eval_{NCA}(p)(t_2))$,

.4 $mk\text{-}Meet(t_1, t_2) \rightarrow Meet_N(p)(Eval_{NCA}(p)(t_1), Eval_{NCA}(p)(t_2))$,

.5 $mk\text{-}Attr(\alpha, t) \rightarrow Attribution_N(p)(\alpha)(Eval_{NCA}(p)(t))$,

.6 $(\textsc{top}) \rightarrow AntiCh(p)$,

.7 $(\textsc{bottom}) \rightarrow \{\}$,

.8 $c \rightarrow cValue_N(p)(c)$

.9 end

In section 4.1 it was shown how the operations $Join_N$, $Meet_N$, $cValue_N$ and ISA_N could be implemented directly as operations on anti-chains without having to convert between down-sets and anti-chains. We can do the same with $Attribution_N$. If we use the definition of $Attribution_{CA}$ we get

$$\begin{aligned}&Attribution_N(p)(\alpha)(ac)\\&= AntiCh(DownSetC(p)(\{attrCI(\alpha)(ci) \mid ci \in DownSet(p)(ac)\}))\\&= CISproj(p)(\{attrCI(\alpha)(ci) \mid ci \in ac\})\end{aligned}$$

where $attrCI$ is defined in 170. So together with the operations defined in section 4.1 we have an implementation of all the operations in $\mathcal{NCA}(cset, aset, p)$ working directly on the anti-chain values.

13.1 Evaluation in the Power Set Partial Order

We saw in section 7 that in order to make an efficient implementation of the most disjoint lattice we needed an efficient implementation of the evaluation function, which evaluates in the power-set partial order. So here we do the same again, so we have it ready for section 15. We define the specialized function $Eval_{NCAP}$ such that

265.0 $Eval_{NCAP}(bset)(t) = Eval_{NCA}(\mathcal{P}(bset))(t)$

266.0 $Eval_{NCAP} : Bset \to Term \to CI\text{-set}$
.1 $Eval_{NCAP}\,(bset)(t) \triangleq$
.2 $\quad$ cases t :
.3 $\quad\quad mk\text{-}Join(t_1, t_2) \to Join_{NCAP}(Eval_{NCAP}(bset)(t_1), Eval_{NCAP}(bset)(t_2)),$
.4 $\quad\quad mk\text{-}Meet(t_1, t_2) \to Meet_{NCAP}(Eval_{NCAP}(bset)(t_1), Eval_{NCAP}(bset)(t_2)),$
.5 $\quad\quad mk\text{-}Attr(\alpha, t) \to \{attrCI(\alpha)(ci) \mid ci \in Eval_{NCAP}(bset)(t)\},$
.6 $\quad\quad (\text{TOP}) \to \{mk\text{-}CI(\{b\}) \mid b \in bset\},$
.7 $\quad\quad (\text{BOTTOM}) \to \{\},$
.8 $\quad\quad c \to \{mk\text{-}CI(\{c\})\}$
.9 $\quad$ end

In order to have the full definition of $Eval_{NCAP}$ available here we repeat the definitions from section 7:

267.0 $Join_{NCAP} : CI\text{-set} \times CI\text{-set} \to CI\text{-set}$
.1 $Join_{NCAP}\,(ac_1, ac_2) \triangleq AntiCh(ac_1 \cup ac_2)$

268.0 $Meet_{NCAP} : CI\text{-set} \times CI\text{-set} \to CI\text{-set}$
.1 $Meet_{NCAP}\,(ac_1, ac_2) \triangleq$
.2 $\quad$ let $cis = \{mk\text{-}CI(bs_1 \cup bs_2) \mid mk\text{-}CI(bs_1) \in ac_1, mk\text{-}CI(bs_2) \in ac_2\}$ in
.3 $\quad AntiCh(cis)$

As can be seen from the definition of $Eval_{NCAP}$ above it is only the term TOP that actually uses the *bset*-argument. Later we will find it convenient to have a version of $Eval_{NCAP}$ which do not need the *bset* argument and consequently only can evaluate restricted terms not containing a TOP-subterm:

269.0 $Eval'_{NCAP} : Termr \to CI\text{-set}$
.1 $Eval'_{NCAP}\,(t) \triangleq$
.2 $\quad$ cases t :
.3 $\quad\quad mk\text{-}Join(t_1, t_2) \to Join_{NCAP}(Eval'_{NCAP}(t_1), Eval'_{NCAP}(t_2)),$
.4 $\quad\quad mk\text{-}Meet(t_1, t_2) \to Meet_{NCAP}(Eval'_{NCAP}(t_1), Eval'_{NCAP}(t_2)),$
.5 $\quad\quad mk\text{-}Attr(\alpha, t) \to \{attrCI(\alpha)(ci) \mid ci \in Eval'_{NCAP}(t)\},$
.6 $\quad\quad (\text{BOTTOM}) \to \{\},$
.7 $\quad\quad c \to \{mk\text{-}CI(\{c\})\}$
.8 $\quad$ end

14 The Most Disjoint Concept Algebra

In order to define the concept of a most disjoint concept algebra — corresponding to the most disjoint lattice defined in section 6 — we must prove lemmas and theorems corresponding to the lemmas and theorems in section 6.2. We will consider the special term value preserving properties in concept algebras, so compared to the proofs in section 6.2 we will assume that the partial order is attribute consistent.

Lemma

270.0 $\forall\, t : \mathit{Term}, pm : PO \cdot$
.1 $\quad \mathit{IsAttrConsistent}(pm) \;\Rightarrow$
.2 $\quad\quad \mathsf{let}\ ptn = \mathit{Eval}_{NCA}(pm)(t)\ \mathsf{in}$
.3 $\quad\quad \forall\, p : PO \cdot ptn \subseteq p \wedge p \subseteq pm \wedge \mathit{IsAttrConsistent}(p)$
.4 $\quad\quad\quad\quad \Rightarrow\; \mathit{Eval}_{NCA}(p)(t) = \mathit{Eval}_{NCA}(pm)(t) = \mathit{Eval}_{CA}(ptn)(t)$

Evaluating a term t (using Eval_{NCA}) in an attribute consistent partial order pm and in any attribute consistent subset partial order p, which includes the normal form value ptn of the term t in pm, yields the same normal form value.

Proof: Assume the left hand side

271.0 $ptn \subseteq p \wedge p \subseteq pm \wedge \mathit{IsAttrConsistent}(p)$

From the term-value property 242 and the two rightmost conjuncts in 271 we get

272.0 $\mathit{Eval}_{CA}(p)(t) \subseteq \mathit{Eval}_{CA}(pm)(t)$
.1 $\mathit{Eval}_{CA}(p)(t) = \mathit{Eval}_{CA}(pm)(t) \cap p$

From 270.2 we now get

$$ptn = \mathit{Eval}_{NCA}(pm)(t) = \mathit{AntiCh}(\mathit{Eval}_{CA}(pm)(t)) \subseteq \mathit{Eval}_{CA}(pm)(t)$$

because an anti-chain of a set ds is a subset of ds. Combining this subset inclusion with the left conjunct in 271 gives

$$\begin{aligned} & ptn \subseteq \mathit{Eval}_{CA}(pm)(t) \cap p \\ & \qquad\qquad\qquad\qquad \text{from } 272.1 \\ & = \mathit{Eval}_{CA}(p)(t) \end{aligned}$$

Again, combining this subset inclusion with the one in 272.0 gives

$$ptn = \mathit{AntiCh}(\mathit{Eval}_{CA}(pm)(t)) \subseteq \mathit{Eval}_{CA}(p)(t) \subseteq \mathit{Eval}_{CA}(pm)(t)$$

If we use the anti-chain property 21 to the above subset inclusion of an antichain we get

$$\mathit{AntiCh}(\mathit{Eval}_{CA}(p)(t)) = \mathit{AntiCh}(\mathit{Eval}_{CA}(pm)(t))$$

which is equivalent to

273.0 $\mathit{Eval}_{NCA}(p)(t) = \mathit{Eval}_{NCA}(pm)(t)$

Next we prove the equality to $\mathit{Eval}_{CA}(ptn)(t)$. From 175.1 we get

$$\mathit{Eval}_{CA}(ptn)(t) \subseteq ptn$$

ptn is an anti-chain and so are all of its subsets, so $\mathit{Eval}_{CA}(ptn)(t)$ is an anti-chain. From the anti-chain property 20 we then get

$$\mathit{Eval}_{NCA}(ptn)(t) = \mathit{AntiCh}(\mathit{Eval}_{CA}(ptn)(t)) = \mathit{Eval}_{CA}(ptn)(t)$$

Finally from 273 for $p = ptn$ and the equality above we get

$$\mathit{Eval}_{NCA}(pm)(t) = \mathit{Eval}_{NCA}(ptn)(t) = \mathit{Eval}_{CA}(ptn)(t)$$

which together with the equality in 273 gives the equalities in 270.4.

Lemma

274.0 $\forall\, t : \mathit{Term}, pm : \mathit{PO} \cdot$
.1 $\quad \mathit{IsAttrConsistent}(pm) \;\Rightarrow$
.2 $\quad\quad \mathsf{let}\; ptn = \mathit{Eval}_{NCA}(pm)(t)\; \mathsf{in}$
.3 $\quad\quad \forall\, p : \mathit{PO} \cdot \neg\, ptn \subseteq p \;\Rightarrow\; \mathit{Eval}_{NCA}(p)(t) \neq \mathit{Eval}_{NCA}(pm)(t)$

Evaluating a term t (using Eval_{NCA}) in an attribute consistent partial order pm and in any subset partial order p, which does not include the normal form value of the term t in pm yields different normal form values.

Proof: Assume the left hand side of the implication:

$$\neg\, ptn \subseteq p$$

so ptn has a non-empty subset not in p:

$$\{\} \subset ptn \setminus p \subseteq ptn = \mathit{Eval}_{NCA}(pm)(t)$$

So $\mathit{Eval}_{NCA}(pm)(t)$ has a nonempty subset, which is not in p. But for $\mathit{Eval}_{NCA}(p)(t)$ we have

$$\mathit{Eval}_{NCA}(p)(t) \subseteq \mathit{Eval}_{CA}(p)(t) \subseteq p$$

Hence

$$\mathit{Eval}_{NCA}(pm)(t) \neq \mathit{Eval}_{NCA}(p)(t)$$

The properties in lemma 270 and 274 can now be combined in the following theorem:

Theorem

275.0 $\forall\, t : \mathit{Term}, pm : \mathit{PO} \cdot$
.1 $\quad \mathit{IsAttrConsistent}(pm) \;\Rightarrow$
.2 $\quad\quad \mathsf{let}\; ptn = \mathit{Eval}_{NCA}(pm)(t)\; \mathsf{in}$
.3 $\quad\quad \forall\, p : \mathit{PO} \cdot p \subseteq pm \wedge \mathit{IsAttrConsistent}(p) \;\Rightarrow$
.4 $\quad\quad\quad ptn \subseteq p \;\Leftrightarrow\; \mathit{Eval}_{NCA}(p)(t) = \mathit{Eval}_{NCA}(pm)(t)$

The proof follows directly from the lemmas 270, and 274. The theorem above shows that if we want to have the term t evaluated to the same normal form value as in the given partial order pm, then we can use exactly all the subsets of pm, which are attribute consistent and contain the normal form value of the term in pm.

Next, we consider a *set* of terms and a partial order pm. In which sub partial orders will all the given terms have the same normal form value as in pm? So the first question is what to do if we want to preserve the value of *two* terms t_1 and t_2? For t_1 we can use all the attribute consistent partial orders between $\mathit{Eval}_{NCA}(pm)(t_1)$ and pm, and for t_2 we can use all the attribute consistent partial orders between $\mathit{Eval}_{NCA}(pm)(t_2)$ and pm. Actually, the smallest attribute consistent partial order between $\mathit{Eval}_{NCA}(pm)(t_1)$ and pm is $\mathit{extCIS}(\mathit{Eval}_{NCA}(pm)(t_1))$ and similar for the t_2 case. Consequently, to keep the value of both t_1 and t_2 we can use all the attribute consistent partial orders between $\mathit{extCIS}(\mathit{Eval}_{NCA}(pm)(t_1)) \cup \mathit{extCIS}(\mathit{Eval}_{NCA}(pm)(t_2))$ and pm. The next theorem generalizes this to an arbitrary set of terms: Given a *set* of terms and a partial order pm. In which sub partial orders will all the given terms have the same normal form value as in pm? The next theorem corresponds to theorem 81.

Theorem

276.0 $\forall\, tset : Term\text{-}\mathsf{set}, pm : PO \cdot$
.1 $IsAttrConsistent(pm) \;\Rightarrow$
.2 $\mathsf{let}\; pts = \bigcup\{extCIS(Eval_{NCA}(pm)(t)) \mid t \in tset\}\; \mathsf{in}$
.3 $\forall\, p : PO \cdot p \subseteq pm \wedge IsAttrConsistent(p) \;\Rightarrow$
.4 $(pts \subseteq p \;\Leftrightarrow\; \forall\, t \in tset \cdot Eval_{NCA}(p)(t) = Eval_{NCA}(pm)(t))$

Proof: Assume the left hand side of the two outermost implications above

277.0 $IsAttrConsistent(pm) \wedge IsAttrConsistent(p) \wedge p \subseteq pm$

Now, to prove the right hand side equivalence, we prove the left to right and right to left implications individually.
LEFT TO RIGHT: So we first assume the left hand side

278.0 $pts \subseteq p$

Next, let t be an arbitrary term in $tset$:

$$t \in tset$$

According to the theorem 275 and the assumptions in 277 we then have

279.0 $Eval_{NCA}(pm)(t) \subseteq p \;\Leftrightarrow\; Eval_{NCA}(p)(t) = Eval_{NCA}(pm)(t)$

Because $extCIS(Eval_{NCA}(pm)(t))$ is the smallest attribute consistent partial order that contains $Eval_{NCA}(pm)(t)$ and p is attribute consistent we have

280.0 $Eval_{NCA}(pm)(t) \subseteq p \;\Leftrightarrow\; extCIS(Eval_{NCA}(pm)(t)) \subseteq p$

so we may rewrite 279 above to

281.0 $extCIS(Eval_{NCA}(pm)(t)) \subseteq p \;\Leftrightarrow\; Eval_{NCA}(p)(t) = Eval_{NCA}(pm)(t)$

From 278 and the definition of pts the left hand side above is true and consequently also the right hand side:

$$Eval_{NCA}(p)(t) = Eval_{NCA}(pm)(t)$$

RIGHT TO LEFT: Next, we must prove the right to left implication in the equivalence:

$$(\forall\, t \in tset \cdot Eval_{NCA}(p)(t) = Eval_{NCA}(pm)(t)) \;\Rightarrow\; pts \subseteq p$$

so assume

$$\forall\, t \in tset \cdot Eval_{NCA}(p)(t) = Eval_{NCA}(pm)(t)$$

From 277 and theorem 275 this may be transformed to

$$\forall\, t \in tset \cdot Eval_{NCA}(pm)(t) \subseteq p$$

Again, if we use that p is attribute consistent and that $extCIS(Eval_{NCA}(pm)(t))$ is the smallest attribute consistent partial order containig $Eval_{NCA}(pm)(t)$ we get

$$\forall\, t \in tset \cdot extCIS(Eval_{NCA}(pm)(t)) \subseteq p$$

So

$$\bigcup\{Eval_{NCA}(pm)(t) \mid t \in tset\} \subseteq p$$

which is equivalente to

$$pts \subseteq p$$

14.1 The Most Disjoint Concept Algebra and its Properties

Given a set of basic concepts, a set of equations *eqs* and a set of user-specified inserted terms, the function below now finds the partial order for the concept algebra which we call the most disjoint concept algebra with respect to the given set of terms.

282.0 $\mathit{TMdisjPO}_{CA} : \mathit{Bset} \rightarrow \mathit{Eq}\text{-set} \rightarrow \mathit{Term}\text{-set} \rightarrow \mathit{PO}$

.1 $\mathit{TMdisjPO}_{CA}\,(\mathit{bset})(\mathit{eqs})(\mathit{insterms}) \triangleq$

.2 $\quad \text{let } \mathit{pmax} = \mathit{MaxPO}_{CA}(\mathit{bset})(\mathit{eqs}) \text{ in}$

.3 $\quad \bigcup\{\mathit{extCIS}(\mathit{Eval}_{NCA}(\mathit{pmax})(t)) \mid t \in \mathit{insterms}\}$

In 282.2 *pmax* is the maximal partial order satisfying the given set of equations *eqs*. In line 282.3 the most disjoint concept algebra is defined to be the set of concept-intersections which is the union of the extended normal form value in *pmax* of all the inserted terms.

From theorem 276 and the definition of the most disjoint concept algebra above we can easily derive the following property for most disjoint concept algebras:

Term-value Preserving Property of Most Disjoint Concept Algebra Let *bset* be a set of basic concepts, *eqs* a set of equations about these concepts and *insterms* a set of user specified inserted terms.

283.0 $\text{let } \mathit{pmax} = \mathit{MaxPO}_{CA}(\mathit{bset})(\mathit{eqs}),$

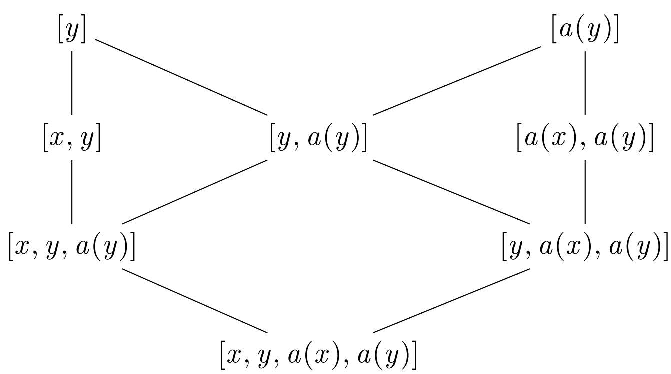
.1 $\quad \mathit{pmdsj} = \mathit{TMdisjPO}_{CA}(\mathit{bset})(\mathit{eqs})(\mathit{insterms}) \text{ in}$

.2 $\forall\, p : \mathit{PO} \cdot p \subseteq \mathit{pmax} \wedge \mathit{IsAttrConsistent}(p) \;\Rightarrow$

.3 $\qquad (\mathit{pmdsj} \subseteq p \;\Leftrightarrow\; \forall\, t \in \mathit{insterms} \cdot \mathit{Eval}_{NCA}(p)(t) = \mathit{Eval}_{NCA}(\mathit{pmax})(t))$

In the most disjoint concept algebra $\mathcal{NCA}(\mathit{cset}, \mathit{aset}, \mathit{pmdsj})$ all the inserted terms evaluate to the same normal form value as they do in the initial concept algebra $\mathcal{NCA}(\mathit{cset}, \mathit{aset}, \mathit{pmax})$. Furthermore, the most disjoint concept algebra is the smallest concept algebra having this property in the sense that all concept algebras based on a partial order not containing *pmdsj* do *not* have this property.

Example We continue with the example on page 64. With $\mathit{bset} = \{x, y, a(x), a(y)\}$ and the equation $x \leq y$ we found the maximal partial order *pmax*



In this partial order *pmax* the terms x, y, $a(x)$ and $a(y)$ evaluates as follows:

t	$Eval_{CA}(pmax)(t)$
x	$[x, y], [x, y, a(x)], [x, y, a(x), a(y)]$
y	$[y], [y, a(y)], [y, a(x), a(y)] \cup Eval_{CA}(pmax)(x)$
$a(x)$	$[a(x), a(y)], [y, a(x), a(y)], [x, y, a(x), a(y)]$
$a(y)$	$[a(y)], [y, a(y)], [x, y, a(y)] \cup Eval_{CA}(pmax)(a(x))$

t	$Eval_{NCA}(pmax)(t)$
x	$[x, y]$
y	$[y]$
$a(x)$	$[a(x), a(y)]$
$a(y)$	$[a(y)]$

Now, let the set of inserted terms be $terms = \{a(x), a(y)\}$. Following the definition of $TMdisjPO_{CA}$ (def. 282) we can now construct the corresponding most disjoint concept algebra as follows:

$insterm$	$Eval_{NCA}(pmax)(insterm)$	$extCIS(Eval_{NCA}(pmax)(insterm))$
$a(x)$	$[a(x), a(y)]$	$[x, y], [a(x), a(y)]$
$a(y)$	$[a(y)]$	$[y], [a(y)]$

So the partial order for the most disjoint concept algebra is the union of the extended normal form values in the right column:

$$pmdsj = \{[x, y], [a(x), a(y)]\} \cup \{[y], [a(y)]\}$$

$$\begin{tikzcd} {[y]} \arrow[d, no head] & {[a(y)]} \arrow[d, no head] \\ {[x, y]} & {[a(x), a(y)]} \end{tikzcd}$$

15 Implementation of the Most Disjoint Concept Algebra

In this section we show how to make an efficient implementation of the most disjoint concept algebra as defined in 282 section 14.1. We will try to proceed almost as in section 7. But of course there will be some important differences, which turn up in the way an equation accepts and rejects concept-intersections. Furthermore, in section 7 we gave a very open/general specification of the algorithm which allowed several concrete implementations. Here we will not try to specify such a general algorithm, but we prefer (for the moment/in this report) to specify a very concrete algorithm.

Notice that in order to avoid the explicit knowledge of the set *bset* of basic concepts we will assume that terms in equations and inserted terms are restricted to terms not containing a TOP-subterm.

Our starting point is the definition of $TMdisjPO_{CA}$ in 282, which we repeat here:

284.0 $TMdisjPO_{CA} : Bset \to Eq\text{-set} \to Termr\text{-set} \to PO$

.1 $TMdisjPO_{CA}\,(bset)(eqs)(insterms) \triangleq$

.2 $\quad \text{let } pmax = MaxPO_{CA}(bset)(eqs) \text{ in}$

.3 $\quad \bigcup\{extCIS(Eval_{NCA}(pmax)(t)) \mid t \in insterms\}$

As in section 7 the main task is to find a way to compute $Eval_{NCA}(pmax)(t)$ for each inserted term without having *pmax* available. Corresponding to 89 in section 7 we have a similar lemma:

Lemma: Term Evaluation using Projection

285.0 $Eval_{NCA}(pmax)(t) = CISproj(pmax)(Eval_{NCA}(\mathcal{P}(bset))(t))$

The proof is similar to the proof for 89.

15.1 Projection into *pmax*.

The equation 285 above shows that we must look for a projection function *cProj* such that

$$cProj(bset, neqs)(cis) = CISproj(pmax)(cis)$$

where *bset* is the set of basic concepts and *neqs* is the normalized equations corresponding to the equations used to compute *pmax*. Having such a function makes it easy to compute $Eval_{NCA}(pmax)(t)$ by combining the equation above with 285 and 265:

$$Eval_{NCA}(pmax)(t) = cProj(bset, neqs)(Eval_{NCAP}(bset)(t))$$

However, because we only allow restricted terms (witout TOP) in the specification of a lattice and we in the approach shown in the sequel never compute *pmax*, we will never need *bset*. The equations above may then be simplified to

286.0 $cProj(neqs)(cis) = CISproj(pmax)(cis)$

287.0 $Eval_{NCA}(pmax)(t) = cProj(neqs)(Eval'_{NCAP}(t))$, where $t : Termr$

The projection function must take the elements in $Eval_{NCA}(\mathcal{P}(bset))(t)$ which are not already in *pmax* (in figure 10 the black bullets outside *pmax*) and project them down into *pmax*. An element is outside *pmax* if and only if it is rejected by an equation or after rejection has been removed by restriction (according to 260). So next we must consider how to decide if a concept-intersection is rejected by an equation. From the description (in sect. 12.2) of the subtractive method to construct *pmax* we have the following: Let $t1_j = t2_j$ be an equation from the given set of equations, let $p_c = \mathcal{P}(bset)$ and let

$$cis1_j = Eval_{CA}(p_c)(t1_j) \text{ and } cis2_j = Eval_{CA}(p_c)(t2_j)$$

then $eqrej0_j = (cis1_j \cup cis2_j) \setminus (cis1_j \cap cis2_j)$ is a set of concept-intersections which must be rejected. Considering all the equations we have (according to 260)

$$pmax = restrCIS(p_c \setminus (eqrej0_1 \cup eqrej0_2 \cup \ldots \cup eqrej0_n))$$

The concept-intersection-set $eqrej0_j$ is called the equations reject-region. Because the set with the rejected elements is furthermore restricted to make it attribute consistent, we say that the concept-intersections in $eqrej0_j$ are *directly* rejected by the equation. Our next task is to understand how the elements that are removed by the restriction are related to the directly rejected elements.

We want to investigate if a concept-intersection *ci* is rejected by an equation *eq*. It is rejected if *ci* is directly rejected by the equation, but assume it is not. Furthermore assume $AttrsInCI(ci) \neq \{\}$ and let $\alpha \in AttrsInCI(ci)$. Then the α-attribution argument $ci' = AttrArgCI(\alpha)(ci)$ exists. If the α-attribution argument ci' is rejected by the equation by being directly rejected by *eq*, so $ci' \notin pmax$, then *ci* is missing its α-attribution argument. So if $ci \in pmax$ then *pmax* would not be attribute consistent. Consequently we must reject

ci as well. In the same way we may argue that if ci' has one of its α-attribution arguments rejected, then ci' must also be rejected and then also ci.

The table in figure 17 shows a concept-intersection ci at the top-line. Every line, except the top-line, contains all possible $AttrArgCI(\alpha)(ci)$ for all concept-intersections ci on the line above it. Consequently, if any concept-intersection ci' in this table is being directly rejected by an equation, then that ci' is rejected *and also* the concept-intersection above of which it is an α-attribution argument and so on upwards until ci at the top-line. For instance, if — in figure 17 — ci_3 is directly rejected by an equation, then also ci_1 and ci are rejected.

$$ci = [a, \underline{\alpha(b), \alpha(\beta(c)), \alpha(\beta(d))}, \underline{\gamma(e), \gamma(\beta(f))}]$$
$$ci_1 = [b, \underline{\beta(c), \beta(d)}], ci_2 = [e, \underline{\beta(f)}]$$
$$ci_3 = [c, d], ci_4 = [f]$$

Figure 17: The set of all *AttrArgCI* for the top-most concept-intersection

The function *AllAttrArgs* defined below finds for a set of concept-intersections *cis* the set of all possible α-attribution arguments as illustrated in the table in figure 17 above:

288.0 $AllAttrArgs : CI\text{-set} \rightarrow CI\text{-set}$
.1 $AllAttrArgs\,(cis) \triangleq$
.2 $\quad \text{let } attrArgs = \bigcup\{\{AttrArgCI(\alpha)(ci) \mid \alpha \in AttrsInCI(ci)\} \mid ci \in cis\} \text{ in}$
.3 $\quad cis \cup$
.4 $\quad \text{if } attrArgs = \{\} \text{ then } \{\} \text{ else } AllAttrArgs(attrArgs)$

A concept-intersection ci is rejected by an equation if any of the concept-intersections in $AllAttrArgs(\{ci\})$ are rejected by the equation. In order to treat these matters we introduce the normalized equations exactly as in section 7:

types

289.0 $\quad NEq :: \quad CI\text{-set} \times CI\text{-set}$

290.0 $EvalEq : Eq \rightarrow NEq$
.1 $EvalEq\,(mk\text{-}Eq(t_1, t_2)) \triangleq$
.2 $\quad \text{let } ac_1 = Eval'_{NCAP}(t_1),$
.3 $\qquad ac_2 = Eval'_{NCAP}(t_2) \text{ in}$
.4 $\quad mk\text{-}NEq(Join_{NCAP}(ac_1, ac_2), Meet_{NCAP}(ac_1, ac_2))$

Notice that we do not need the set *bset* of basic concepts to evaluate the equation terms, because these terms are restricted terms with no TOP-subterms. We will evaluate all the equations in this way:

291.0 $EvalEqs : Eq\text{-set} \rightarrow NEq\text{-set}$
.1 $EvalEqs\,(eqs) \triangleq \{EvalEq(eq) \mid eq \in eqs\}$

Let eqs be the given set of equations, then we will use the set of normalized equations $neqs = EvalEqs(eqs)$ to (implicitly) represent $pmax$.

The function *InEqDirectRej* defined below tells if a concept-intersection is directly rejected by an equation by being in the equations rejection region:

292.0 $\mathit{InEqDirectRej} : \mathit{NEq} \rightarrow \mathit{CI} \rightarrow \mathbb{B}$

.1 $\mathit{InEqDirectRej}\,(\mathit{mk\text{-}NEq}(ac_1, ac_2, m))(ci) \triangleq$

.2 $\quad \mathit{ISA}_N(\{ci\}, u) \wedge \neg\, \mathit{ISA}_N(\{ci\}, m)$

Now we can define a function *IsEqRejected* corresponding to the *InEqRej*-function defined in 99:

293.0 $\mathit{IsEqRejected} : \mathit{NEq} \rightarrow \mathit{CI} \rightarrow \mathbb{B}$

.1 $\mathit{IsEqRejected}\,(neq)(ci) \triangleq$

.2 $\quad \mathsf{let}\ \mathit{attrArgs} = \mathit{AllAttrArgs}(\{ci\})\ \mathsf{in}$

.3 $\quad \exists\, aci \in \mathit{attrArgs} \cdot \mathit{InEqDirectRej}(neq)(aci)$

In words, the concept-intersection ci is rejected if ci itself or any of its attribution arguments are directly rejected.

15.2 Computing Upper Bounds

If a concept-intersection ci is *directly* rejected by a normalized equation $\mathit{mk\text{-}NEq}(u, m)$, then we know that all concept-intersections below (ISA_P) ci, which are in $pmax$ must be below some concept-intersection in m, so we have:

294.0 $\mathit{InEqDirectRej}(\mathit{mk\text{-}NEq}(\text{-}, m))(ci) \;\Rightarrow\; \mathit{ISA}_N(\mathit{CISproj}(pmax)(\{ci\}), m)$

But what if a concept-intersection is *indirectly* rejected? To solve this problem we have the following lemma, which relates the projections of a concept-intersection ci and its attribution arguments:

Lemma:

295.0 $\forall\, eqs : Eq\text{-}\mathsf{set}, ci : CI, upl_{arg} : CI\text{-}\mathsf{set}, bset : Bset \cdot$

.1 $\qquad \forall\, \alpha \in \mathit{AttrsInCI}(ci) \cdot$

.2 $\qquad\qquad \mathsf{let}\ pmax = \mathit{MaxPO}_{CA}(bset)(eqs),$

.3 $\qquad\qquad\quad ci_{arg} = \mathit{AttrArgCI}(\alpha)(ci)\ \mathsf{in}$

.4 $\qquad\qquad (\mathit{ISA}_S(upl_{arg}, \{ci_{arg}\}) \wedge \mathit{ISA}_S(\mathit{CISproj}(pmax)(\{ci_{arg}\}), upl_{arg})$

.5 $\qquad\qquad\quad \Rightarrow$

.6 $\qquad\qquad \mathsf{let}\ upl = \mathit{Meet}_{NCAP}(\{ci\}, \mathit{atCIs}(\alpha)(upl_{arg}))\ \mathsf{in}$

.7 $\qquad\qquad \mathit{ISA}_S(upl, \{ci\}) \wedge$

.8 $\qquad\qquad \mathit{ISA}_S(\mathit{CISproj}(pmax)(\{ci\}), upl))$

In words, if ci_{arg} is the α-attribution argument of ci, and upl_{arg} an upper-bound for $\mathit{CISproj}(pmax)(\{ci_{arg}\})$, then $\mathit{Meet}_{NCAP}(\{ci\}, \mathit{atCIs}(\alpha)(upl_{arg}))$ will be an upper-bound for $\mathit{CISproj}(pmax)(\{ci\})$. Of course, this property is a consequence of $pmax$ being attribute consistent. The lemma and its proof is illustrated in figure 18.

In order to prove 295 above we need some auxiliary lemmas:

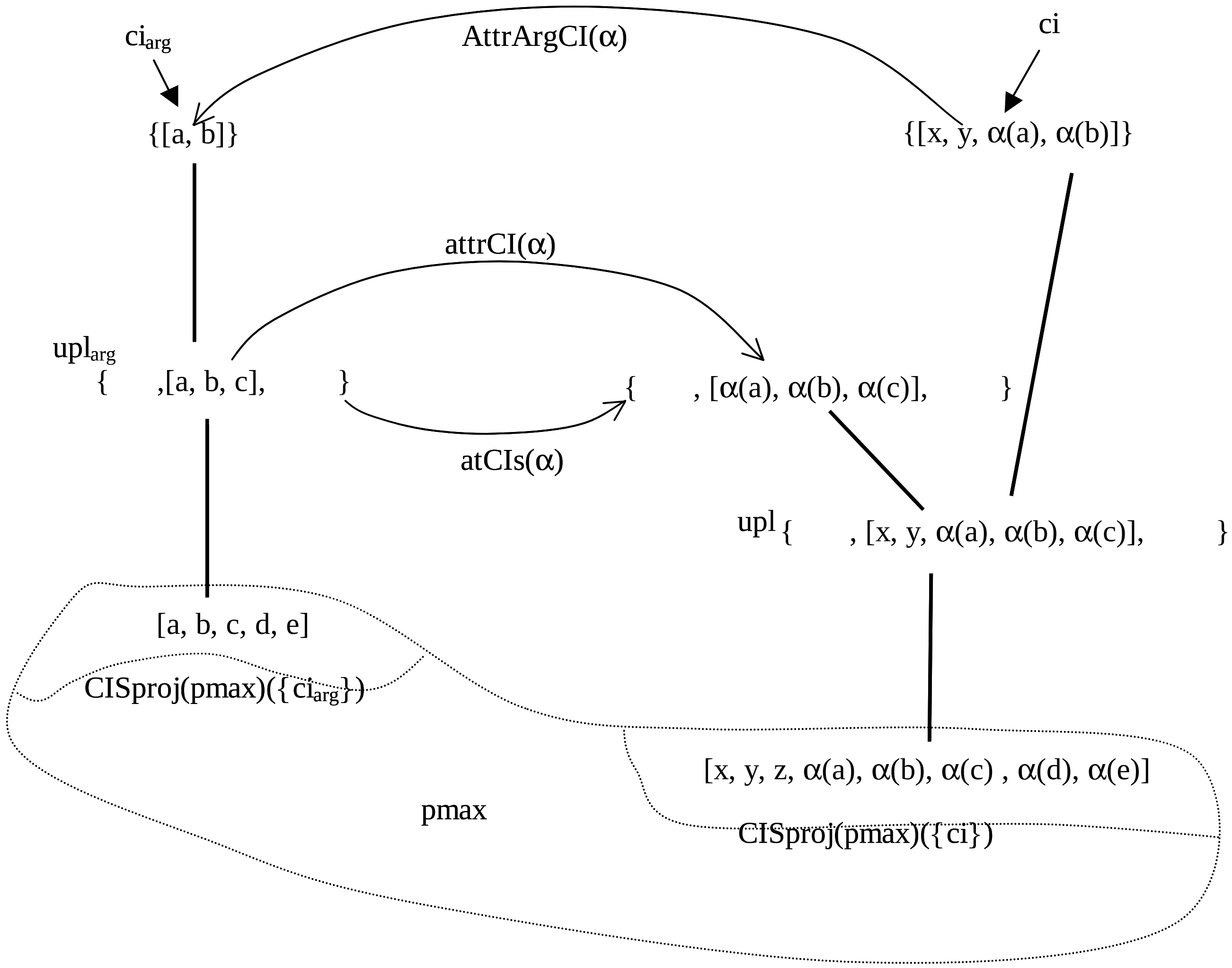


Figure 18: Illustation of proof for lemma 295

AuxLemma:

296.0 $ISA_S(cis, cis_1) \wedge ISA_S(cis, cis_2) \;\Rightarrow\; ISA_S(cis, Meet_{NCAP}(cis_1, cis_2))$

Proof: Assume the left hand side of the implication above:

$$ISA_S(cis, cis_1) \wedge ISA_S(cis, cis_2)$$

If we use the definition of ISA_S (104) we get

$$\forall\, mk\text{-}CI(bs) \in cis \cdot \exists\, mk\text{-}CI(bs_1) \in cis_1 \cdot bs_1 \subseteq bs \wedge$$
$$\forall\, mk\text{-}CI(bs) \in cis \cdot \exists\, mk\text{-}CI(bs_2) \in cis_2 \cdot bs_2 \subseteq bs$$

which can be rewritten to

297.0 $\forall\, mk\text{-}CI(bs) \in cis \cdot$

.1 $\quad \exists\, mk\text{-}CI(bs_1) \in cis_1, mk\text{-}CI(bs_2) \in cis_2 \cdot bs_1 \cup bs_2 \subseteq bs$

Again, using the definition of ISA_S (104) gives

$$\text{let } mcis = \{mk\text{-}CI(bs_1 \cup bs_2) \mid mk\text{-}CI(bs_1) \in cis_1, mk\text{-}CI(bs_2) \in cis_2\} \text{ in}$$
$$ISA_S(cis, mcis)$$

Next, a little bit informally, as $mk\text{-}CI(bs_i) \in cis_i, i = 1, 2$, that exists in 297 satisfies $bs_1 \cup bs_2 \subseteq bs$, then every concept-intersection above it in cis_i will also satisfy this subset inclusion so we may actually conclude that

$$\text{let } mcis = AntiCh\{mk\text{-}CI(bs_1 \cup bs_2) \mid mk\text{-}CI(bs_1) \in cis_1, mk\text{-}CI(bs_2) \in cis_2\} \text{ in}$$
$$ISA_S(cis, mcis)$$

Finally, from the definition of $Meet_{NCAP}$ (268) we then get

$$ISA_S(cis, Meet_{NCAP}(cis_1, cis_2))$$

□

AuxLemma:

298.0 $\forall\, upl_{arg} : CI\text{-set}, ciprj : CI \cdot$

.1 $\quad ISA_S(\{AttrArgCI(\alpha)(ciprj)\}, upl_{arg}) \;\Rightarrow\; ISA_S(\{ciprj\}, atCIs(\alpha)(upl_{arg}))$

The auxiliary lemma and its proof is illustrated in figure 19.

Proof: Below, according to the universal quantifier above let $uplarg, ciprj$ be arbitrary values such that

$$upl_{arg} : CI\text{-set}, ciprj : CI$$

From the defiition of ISA_S we can rewrite 298 above to

299.0 $\exists\, ciuplarg \in upl_{arg} \cdot ISA_P(AttrArgCI(\alpha)(ciprj), ciuplarg) \;\Rightarrow$

.1 $\exists\, cia \in atCIs(\alpha)(upl_{arg}) \cdot ISA_P(ciprj, cia)$

To prove the implication in 299 we assume the left hand side 299.0 and prove the right hand side. So let $ciuplarg$ be the concept-intersection that exists according to 299.0. We then get

300.0 $ciuplarg \in upl_{arg}$,

.1 $ISA_P(AttrArgCI(\alpha)(ciprj), ciuplarg)$

Applying 180 to $ciprj$ gives

301.0 $ISA_P(ciprj, attrCI(\alpha)(AttrArgCI(\alpha)(ciprj)))$

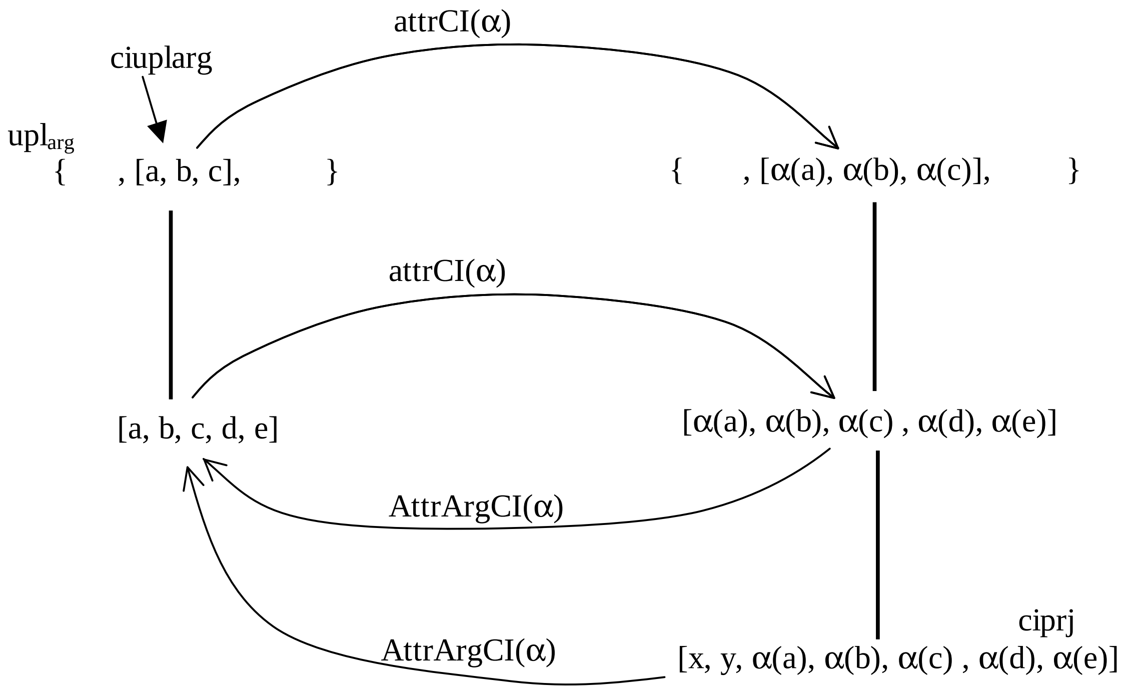


Figure 19: Illustation of proof for lemma 298

From the definition of the function *attrCI* (170) we can easily conclude that

$$ISA_P(ci_1, ci_2) \;\Rightarrow\; ISA_P(attrCI(\alpha)(ci_1), attrCI(\alpha)(ci_2))$$

Applying this to 300.1 gives

$$ISA_P(attrCI(\alpha)(AtrArgCI(\alpha)(ciprj)), attrCI(\alpha)(ciuplarg))$$

Combining this with 301 gives

302.0 $ISA_P(ciprj, attrCI(\alpha)(ciuplarg))$

From 300.0 and the definition of *atCIs* (171) we get

$$attrCI(\alpha)(ciuplarg) \in \{attrCI(\alpha)(ci) \mid ci \in uplarg\} = atCIs(\alpha)(uplarg)$$

Finally, combining this with 302 gives

$$\exists\, cia \in atCIs(\alpha)(upl_{arg}) \cdot ISA_P(ciprj, cia)$$

which is the right hand side 299.0 of 299.

□

We are now ready for a proof of 295.

Proof of 295: In the sequel, according to the universal quantifier in 295, let $eqs, ci, upl_{arg}, bset, \alpha$ be arbitrary values such that

$$eqs : Eq\text{-set}, ci : CI, upl_{arg} : CI\text{-set}, bset : Bset, \alpha : A, \alpha \in AttrsInCI(ci)$$

We can easily prove the conjunct in 295.7. Let

303.0 $mk\text{-}CI(bci) = ci$

Then

304.0 $upl = Meet_{NCAP}(\{ci\}, atCIs(\alpha)(upl_{arg}))$

from 268 we get

.1 $\subseteq \{mk\text{-}CI(bs1 \cup bs2) \mid mk\text{-}CI(bs1) = ci, mk\text{-}CI(bs2) \in atCIs(\alpha)(upl_{arg})\}$

.2 $= \{mk\text{-}CI(bci \cup bs2) \mid mk\text{-}CI(bs2) \in atCIs(\alpha)(upl_{arg})\}$

From 304 we get

305.0 $\forall\, ciu \in upl \cdot \exists\, mk\text{-}CI(bs_2) \in atCIs(\alpha)(upl_{arg}) \cdot ciu = mk\text{-}CI(bci \cup bs2)$

Finally 303, 305 and the definition of ISA_S and ISA_P give

306.0 $ISA_S(upl, \{ci\})$

So the conjunct in 295.7 does not depend on 295.4 at all.

Next we prove the conjunct in 295.8. According to 296 we just have to prove

307.0 $ISA_S(CISproj(pmax)(\{ci\}), \{ci\})$,

.1 $ISA_S(CISproj(pmax)(\{ci\}), atCIs(\alpha)(upl_{arg}))$

The proof of 307.0 follows immediately from the definition of *CISproj* (26). So below we finally consider the proof of 307.1. According to 295.2, 295.3 let

308.0 $pmax = MaxPO_{CA}(bset)(eqs)$,

.1 $ci_{arg} = AttrArgCI(\alpha)(ci)$

Furthermore, assume the left hand side (295.4) of the implication. So we have

309.0 $ISA_S(upl_{arg}, \{ci_{arg}\})$,

.1 $ISA_S(CISproj(pmax)(\{ci_{arg}\}), upl_{arg})$

We prove 307.1 by contradiction. So we now also assume

$$\neg\, ISA_S(CISproj(pmax)(\{ci\}), atCIs(\alpha)(upl_{arg}))$$

If we use the definition of ISA_S we get

$$\exists\, ciprj \in CISproj(pmax)(\{ci\}) \cdot \forall\, cia \in atCIs(\alpha)(uplarg) \cdot \neg\, ISA_P(ciprj, ci)$$

So let *ciprj* be the concept-intersection that exists. We then have

310.0 $ciprj \in CISproj(pmax)(\{ci\})$,

.1 $\forall\, cia \in atCIs(\alpha)(uplarg) \cdot \neg\, ISA_P(ciprj, cia)$

From 298 and $a \;\Rightarrow\; b \;\Leftrightarrow\; \neg\, b \;\Rightarrow\; \neg\, a$ we get

$$\forall\, cia \in atCIs(\alpha)(uplarg) \cdot \neg\, ISA_P(ciprj, cia) \;\Rightarrow$$
$$\forall\, ciuplarg \in upl_{arg} \cdot \neg\, ISA_P(AttrArgCI(\alpha)(ciprj), ciuplarg)$$

Combining this with 310.1 gives

311.0 $\forall\, ciuplarg \in uplarg \cdot \neg\, ISA_P(AttrArgCI(\alpha)(ciprj), ciuplarg)$

From 310.0 and the definition of *CISproj* we have

$$ISA_P(ciprj, ci)$$

If we apply this to the implication in 183 we get

$$ISA_P(AttrArgCI(\alpha)(ciprj), AttrArgCI(\alpha)(ci))$$

which, by use of 308.1, may be rewritten to

312.0 $ISA_P(AttrArgCI(\alpha)(ciprj), ci_{arg})$

From 309.0 and 309.1 we know that every ci in $pmax$ and below ci_{arg} must also be below upl_{arg}. From 312 we know that $AttrArgCI(\alpha)(ciprj)$ is below ci_{arg}, but from 311 we know that it is not below upl_{arg} and then consequently can not be in $pmax$. So although $ciprj \in pmax$ we have $AttrArgCI(\alpha)(ciprj) \notin pmax$ and consequently $pmax$ can not be attribut consistent. But this is a contradiction as $pmax$ as defined in 308.0 is known to be attribute consistent. □

Using lemma 295 to compute upper bounds. Lemma 295 above may help us compute upper bounds for the projection of indirectly rejected concept-intersections. As an example, assume that ci_3 in figure 17 is directly rejected by the normalized equation $neq = mk\text{-}NEq(u, m)$. Then, according to 294 above, we have

$$ISA_N(CISproj(pmax)(\{ci_3\}), m)$$

We also have

$$ISA_N(CISproj(pmax)(\{ci_3\}), \{ci_3\})$$

Put together we get

$$upl_3 = Meet_{NCAP}(m, \{ci_3\})$$
$$ISA_N(CISproj(pmax)(\{ci_3\}), upl_3)$$

As $ci_3 = AttrArgCI(\beta)(ci_1)$ we may now conclude from 295 that

$$upl_1 = Meet_{NCAP}(\{ci_1\}, atCIs(\beta)(upl_3)),$$
$$ISA_S(CISproj(pmax)(\{ci_1\}), upl_1) \wedge ISA_S(upl_1, \{ci_1\})$$

In the same way, as $ci_1 = AttrArgCI(\alpha)(ci)$, we may also conclude that

$$upl = Meet_{NCAP}(\{ci\}, atCIs(\alpha)(upl_1)),$$
$$ISA_S(CISproj(pmax)(\{ci\}), upl) \wedge ISA_S(upl, \{ci\})$$

So upl is the upper limit for $CISproj(pmax)(\{ci\})$ we are looking for.

Before we proceed with a detailed algorithm for this new way to compute upper bounds we will start with the main algorithm for term evaluation using projection.

15.3 The Algorithm for Computing $Eval_{NCA}$

The top functions are almost identical to the functons defined in section 7. First the function which computes $Eval_{NCA}$ as described in 287:

313.0 $cEvalNCA : NEq\text{-set} \rightarrow Term \rightarrow CI\text{-set}$

.1 $cEvalNCA(neqs)(t) \triangleq cProj(neqs)(Eval'_{NCAP}(t))$

The projection of the set of concept-intersections — obtained from the evaluation of the term t — into $pmax$ is done by $cProj$:

314.0 $cProj : NEq\text{-set} \rightarrow CI\text{-set} \rightarrow CI\text{-set}$

.1 $cProj(neqs)(cis) \triangleq$

.2 $\quad$ let $mk\text{-}(\text{-}, pcis) = cIProj(neqs)(\mathsf{true})(cis, \{\})(\{\})$ in

.3 $\quad pcis$

The only task for $cProj$ is to start the iterated projection of cis. We will understand the new parameters later.

The Iterated Projection The repeated execution of the projection step is done by the function *cIProj* defined below:

315.0 $cIProj : NEq\text{-set} \to \mathbb{B} \to CI\text{-set} \times CI\text{-set} \to CI\text{-set} \to \mathbb{B} \times CI\text{-set}$

.1 $cIProj\,(neqs)(accUnch)(upl, InP)(pLevCis) \triangleq$

.2 $\quad \text{let } mk\text{-}(newunch, newupl, newInP) = cProjStep(neqs)(upl, InP)(pLevCis) \text{ in}$

.3 $\quad \text{if } newunch$

.4 $\quad \text{then } mk\text{-}(accUnch, AntiCh(newInP))$

.5 $\quad \text{else } cIProj(neqs)(\text{false})(newupl, newInP)(pLevCis)$

In each (tail recursive) call of *cIProj* the function *cProjStep* takes the current partial projection $upl \cup Inp$ and yields the next (partial) projection $newupl \cup newInp$. The projection stops when the new projection has not changed (indicated by *newunch*); the projection is then in *newInP*. The parameter *accUnch* keeps track of changes. If a projection step changes the partial projection then *accUnch* becomes false, otherwise *accUnch* is returned with its initial value true.

The Projection Step. As we saw above, the single projection step in the iterated projection is done by the function *cProjStep*:

316.0 $cProjStep : NEq\text{-set} \to CI\text{-set} \times CI\text{-set} \to CI\text{-set} \to \mathbb{B} \times CI\text{-set} \times CI\text{-set}$

.1 $cProjStep\,(neqs)(upl, InP)(pLevCis) \triangleq$

.2 $\quad \text{let } cupls = \{cEqsUpl(neqs)(pLevCis)(ci) \mid ci \in upl\} \text{ in}$

.3 $\quad \text{let } InP1 = \{ci \mid mk\text{-}((\text{true}), \{ci\}) \in cupls\},$

.4 $\qquad notInP = \{upl' \mid mk\text{-}((\text{false}), upl') \in cupls\},$

.5 $\qquad newupl = \bigcup notInP \text{ in}$

.6 $\quad mk\text{-}(notInP = \{\}, newupl, InP \cup InP1)$

The concept-intersections which are not yet known to be in *pmax* are in *upl*. In order to make the next step in the projection of $upl \cup InP$, *cProjStep* makes a separate projection of each $ci \in upl$ using *cEqsUpl* (316.2). In 316.3 $InP1$ is the subset of *upl* that turned out not to be rejected, neither directly nor indirectly, and consequently is in *pmax*, and in 316.5 *newupl* is the union of the new upper-limits for the remainig concept-intersections.

15.4 Projecting a Single concept-intersection.

In this section we now consider the algorithms based on the theory outlined in sections 15.1 and 15.2. First we need an auxiliary function. $CisMeet_{NCAP}(ciss)$ defines the greatest lower bound of a set of anti-chains *ciss*:

317.0 $CisMeet_{NCAP} : CI\text{-set-set} \to CI\text{-set}$

.1 $CisMeet_{NCAP}\,(ciss) \triangleq$

.2 $\quad \text{cases } ciss :$

.3 $\qquad \{cis\} \to cis,$

.4 $\qquad \{cis\} \cup ciss' \to Meet_{NCAP}(cis, CisMeet_{NCAP}(ciss'))$

.5 $\quad \text{end}$

.6 $\text{pre } \; ciss \neq \{\}$

The computation of an upper-limit for $\mathit{CISproj}(\mathit{pmax})(\{ci\})$ as described in 15.1 and 15.2, is done by $\mathit{cEqsUpl}(\mathit{neqs})(ci)$, defined below. The computation of upper limits is based on *indirect* as well as direct rejection:

318.0 $\mathit{cEqsUpl} : \mathit{NEq}\text{-set} \rightarrow \mathit{CI}\text{-set} \rightarrow \mathit{CI} \rightarrow \mathbb{B} \times \mathit{CI}\text{-set}$

.1 $\mathit{cEqsUpl}\,(\mathit{neqs})(\mathit{pLevCis})(ci) \triangleq$

.2 $\quad \text{let } \mathit{uls}1 = \{m \mid \mathit{mk}\text{-}\mathit{NEq}(u,m) \in \mathit{neqs} \cdot \mathit{InEqDirectRej}(\mathit{mk}\text{-}\mathit{NEq}(u,m))(ci)\},$

.3 $\quad\quad \mathit{uls}2 = \{\mathit{iupl} \mid \alpha \in \mathit{AttrsInCI}(ci), \mathit{iupl} : \mathit{CI}\text{-set} \cdot$

.4 $\quad\quad\quad\quad \mathit{mk}\text{-}(\mathsf{false}, \mathit{iupl}) = \mathit{cIUpl}(\mathit{neqs})(\alpha)(\mathit{pLevCis})(ci)\} \text{ in}$

.5 $\quad \text{let } \mathit{uls} = \mathit{uls}1 \cup \mathit{uls}2 \text{ in}$

.6 $\quad \text{cases } \mathit{uls} :$

.7 $\quad\quad \{\} \rightarrow \mathit{mk}\text{-}(\mathsf{true}, \{ci\}),$

.8 $\quad\quad \text{-} \rightarrow \mathit{mk}\text{-}(\mathsf{false}, \mathit{CisMeet}_{NCAP}(\{\{ci\}\} \cup \mathit{uls}))$

.9 $\quad \text{end}$

In line 318.2 $\mathit{uls}1$ is the set of upper limits coming from the equations that *directly* rejected ci. Next, in line 318.4 $\mathit{uls}2$ is the set of upper limits coming from ci being *indirectly* rejected. We will explain this in more detail below and so also the parameter $\mathit{pLevCis}$. If the union uls of these sets is empty (318.7) then ci is not rejected, neither direcly nor indirectly, so it is in pmax (indicated by the first component, true). If uls is not empty (318.8) then the greatest lower bound of $\{\{ci\}\} \cup \mathit{uls}$ is the new upper limit (se fig. 18 and 295.6). However, we don't know if all the concept-intersections in this new upper limit are in pmax (indicated by false).

When ci is *indirectly* rejected the set of upper limits is computed by the function cIUpl defined below (319). If $\mathit{cIUpl}(\mathit{neqs})(\alpha)(\mathit{pLevCis})(ci)$ finds that ci is indirectly rejected then it returns with $\mathit{mk}\text{-}(\mathsf{false}, \mathit{iupl})$, where iupl is the found upper limit. If ci is not indirectly rejected it returns with $\mathit{mk}\text{-}(\mathsf{true}, \{ci\})$.

319.0 $\mathit{cIUpl} : \mathit{NEq}\text{-set} \rightarrow A \rightarrow \mathit{CI}\text{-set} \rightarrow \mathit{CI} \rightarrow \mathbb{B} \times \mathit{CI}\text{-set}$

.1 $\mathit{cIUpl}\,(\mathit{neqs})(\alpha)(\mathit{pLevCis})(ci) \triangleq$

.2 $\quad \text{let } ci_{arg} = \mathit{AttrArgCI}(\alpha)(ci) \text{ in}$

.3 $\quad \text{let } \mathit{mk}\text{-}(\mathit{unch}, \mathit{proj}) =$

.4 $\quad\quad\quad \text{if } ci_{arg} \in \mathit{pLevCis} \text{ then } \mathit{mk}\text{-}(\mathsf{false}, \{\})$

.5 $\quad\quad\quad \text{else } \mathit{cIProj}(\mathit{neqs})(\mathsf{true})(\{ci_{arg}\}, \{\})(\mathit{pLevCis} \cup \{ci_{arg}\}) \text{ in}$

.6 $\quad \text{cases } \mathit{unch} :$

.7 $\quad\quad \mathsf{false} \rightarrow \mathit{mk}\text{-}(\mathsf{false}, \mathit{atCIs}(\alpha)(\mathit{proj})),$

.8 $\quad\quad \mathsf{true} \rightarrow \mathit{mk}\text{-}(\mathsf{true}, \{ci\})$

.9 $\quad \text{end}$

The function cIUpl (compute indirect upper limit) tests if ci is indirectly rejected, and if it is, computes the upper bound according to lemma 295. As illustrated in figure 18, given ci we must first apply $\mathit{AttrArgCI}(\alpha)$ to get the attribution argument ci_{arg} (line 319.2). Next (line 319.4–319.5) ci_{arg} must be projected into pmax. First assume $ci_{arg} \notin \mathit{pLevCis}$ (to be explained below); then ci_{arg} is projected into pmax using the function cIProj for iterated projection (defined in 315).

If the iterated projection returns with $\mathit{mk}\text{-}(\mathsf{false}, \mathit{proj})$ then proj is the projection of ci_{arg} (ci_{arg} is changed), so proj is the best possible upper-limit for the projection; it corresponds to upl_{arg} in figure 18. Finally, according to 295.6 (and again figure 18) we must apply $\mathit{atCIs}(\alpha)$

to upl_{arg} to deliver the contribution to the upper bound for the indirectly rejected ci. This is returned together with false to indicate that ci actually *was* indirectly rejected.

If the iterated projection returned with the first component equal to true, then ci was *not* indirectly rejected so *cIUpl* returns the unchanged ci (indicated by true).

Notice that *cIUpl* after having constructed ci_{arg} starts a *new projection* using *cIProj* so we may have an iterated projection running at several levels simultaneously.

Example Consider the example specification

```
equations
   c * a(c) = a(c)
terms a(a(c))
```

The equation corresponds to the *ISA*-relation $a(c) \leq c$ and gives the normalized equation $mk\text{-}NEq(\{[a(c)]\}, \{[c, a(c)]\})$.

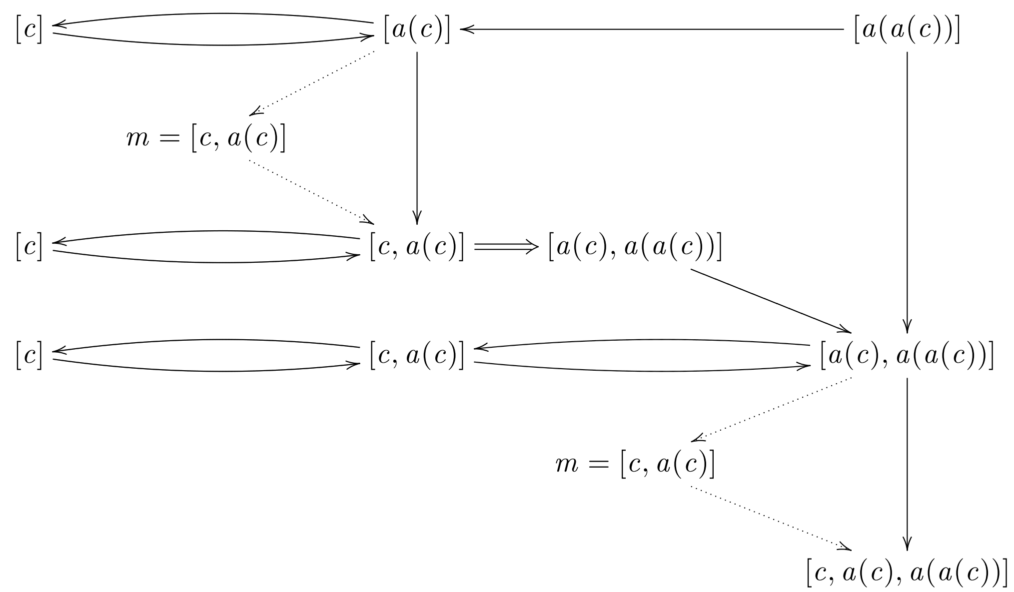


Figure 20: Inserting $a(a(c))$, equation: `c * a(c) = a(c)`

The computation of $cProj(neqs)(\{[a(a(c))]\})$ is illustrated in figure 20. The upper bounds in the example contains only one concept-intersection. Nodes on vertical lines represent (tail recursive) calls of *cIProj* and the resulting call of *cProjStep* and *cEqsUpl*. The concept-intersections along a vertical line are partial projections with the final projection at the bottom of the line.

A horizontal left arrowed line represents a call of *cIUpl* with the argument ci at the right end and ci_{arg} at the left end. A right arrowed line represent return from *cIUpl*. A single lined right arrow means that ci is not indirectly rejected so *cIUpl* returns ci unchanged. A double lined right arrow indicates that ci *is* indirectly rejected and the concept-intersection at the right end of the arrow is the found upper bound.

The *direct* rejection of a concept-intersection ci is indicated by a dotted line from ci down to the normalized equations m-part and further down to the new upper bound.

In the example the concept-intersection ci to be projected is at the top right corner, and the final projection at the right bottom corner. After extension the partial order becomes $\{[c, a(c), a(a(c))], [c, a(c)], [c]\}$.

□

Example Next consider the example specification

```
equations
   c * a(c) = c
terms a(a(c))
```

The equation corresponds to the *ISA*-relation $c \leq a(c)$. From the monotonicity rule for attribution we also have $a(c) \leq a(a(c))$, $a(a(c)) \leq a(a(a(c)))$, etc. Due to the finite attribution approach, for some level of attribution nesting the term $a(a(\ldots a(c)\ldots))$ will evaluate to $\perp$ and then according to the derived equations above also all terms of the form $c, a(c), a(a(c)), \ldots$. How will this turn up when inserting the term $a(a(c))$?

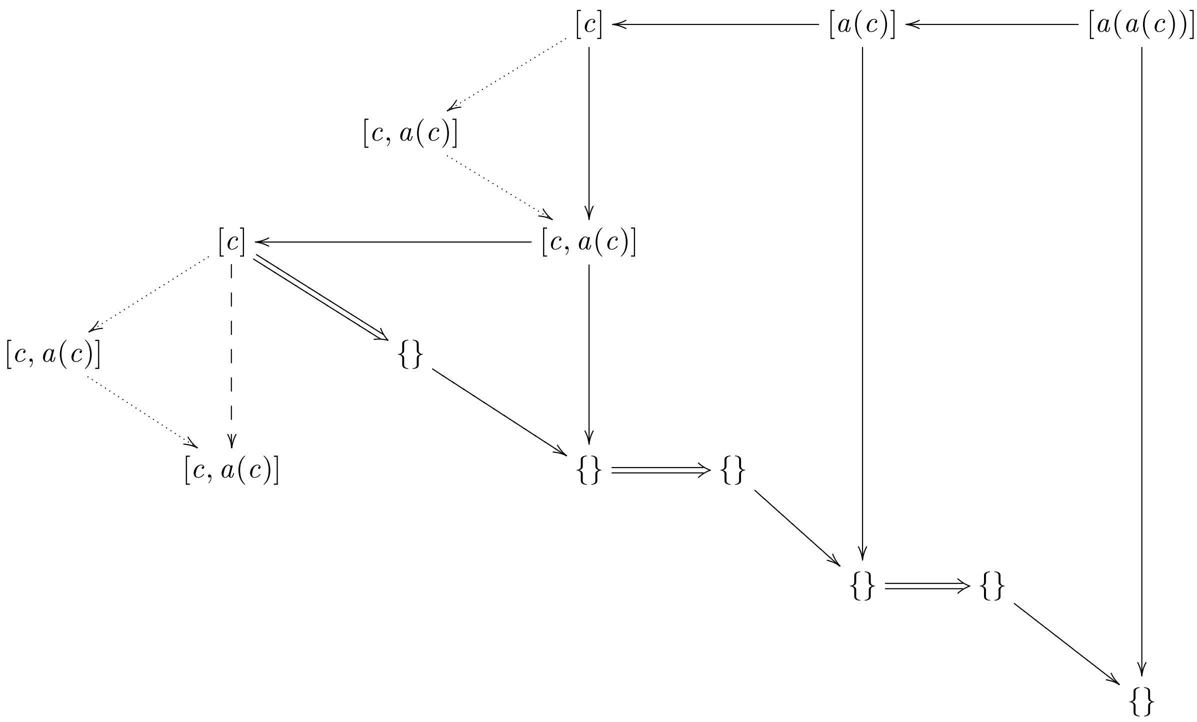


Figure 21: Inserting a(a(c)), equation: `c * a(c) = c`

The normalized equation for `c*a(c)=c` is $\textit{mk-NEq}(\{[c]\}, \{[c, a(c)]\})$. The computation of $\textit{cProj}(\textit{neqs})(\{[a(a(c))]\})$ is illustrated in figure 21. The upper bounds in the example contains at most one concept-intersection. We use the same notation as in the previous example.

The upper most horizontal line indicates the same sequence of calls of *cIUpl* as in the previous example, but in this example the concept-intersection $[c]$ (at level 3) is directly rejected and in the first projection step projected down to $[c, a(c)]$. Here *cEqsUpl* now looks for direct as well as *indirect* rejection of $[c, a(c)]$. The call of *cIUpl* with $[c, a(c)]$ then computes $ci_{arg} = [c]$ and consequently now (at level 4) looks for a projection of $[c]$, exactly as at the previous level 3. The start of this projection is indicated by dashed/dotted arrows to the left. As can be seen, this search for an indirect upper bound will go on for ever. If, hypothetically, an upper bound was found, then an infinitely nested attribution would be returned, which according to the finite attribution approach should be $\perp$.

As can be seen from figure 21, the algorithm will detect that it is starting to look for a projection of a concept-intersection, which it is already looking for at another level and consequently it returns $\{\}$ $(\perp)$ as the upper bound. □

The situation illustrated in the previous example is handled in *cIUpl* in lines 319.4-319.5. The parameter *pLevCis* (previous levels *ci*'s) holds all ci_{arg}'s for which *cIUpl* has started a computation of the projection (319.5) leading to the current call of *cIUpl*. If the current *ci* is in *pLevCis* then $\{\}$ is returned (319.4).

15.5 Implementing $cTMdisjPO$.

Finally, from the definition of $TMdisjPO_{CA}$ (282) we can now define the function that computes the most disjoint lattice with respect to a given set of terms :

320.0 $cTMdisjPO : Eq\text{-set} \rightarrow Termr\text{-set} \rightarrow PO$

.1 $cTMdisjPO\,(eqs)(terms) \triangleq$

.2 $\quad$ let $neqs = EvalEqs(eqs)$ in

.3 $\quad\quad extCIS(\bigcup\{(cEvalNCA(neqs)(t)) \mid t \in terms\})$

16 Querying the Concept Algebra

In this section we consider how to extract information from a concept algebra. The simplest way to extract information is simply to evaluate terms in the algebra. So assume we from a given lattice specification $mk\text{-}LatSpec(eqs, terms)$ – consisting of the equations and the terms to be inserted – have constructed the corresponding partial order *pmdsj* for the most disjoint lattice. We can now extract information from the lattice/database by evaluating a term t in the lattice:

$$Eval_{NCA}(pmdsj)(t)$$

The result will be the corresponding anti-chain in the form of a set of concept-intersections. The introduction (section 1) showed several examples of this kind of lattice/database construction and subsequent querying.

Evaluation of a term in a given algebra can only yield values from the carrier of the algebra, but can not construct *new* values obtained by combining already existing values.

Example In the specification below the equations specifies two relational database tables and the terms part insert rows in the two tables.

```
equations
```

```
  db1<= A(a) * B(b) * C(c),
  db2<= C(c)* D(d),

  c>= r + s + t,
  r>= r1+r2+r3,
  s>= s1+s2,
  t>= t1+t2

terms
  db1*A(a1)*B(b1)*C(r1),
  db1*A(a2)*B(b2)*C(s1),
  db1*A(a3)*B(b1)*C(t),
  db1*A(a4)*B(b3)*C(r2),
  db1*A(a5)*B(b2)*C(s2),

  db2*C(r2)*D(d1),
  db2*C(r1)*D(d2),
  db2*C(r3)*D(d3),
  db2*C(s) *D(d4),
  db2*C(t2)*D(d5)

queries
 db1 * db2,
```

As explained in [3] applying the lattice meet operation to the tables above should yield a result corresponding to the database natural join operation. The result of the query may be a little bit disappointing:

```
Queries:
  db1 * db2: < >
```

i.e. the term $db1 * db2$ evaluates to $\perp$. The reason is that the expected joined rows do not exist as values in the constructed concept algebra. □

16.1 Constructing Data Base Natural Joins

We need a special kind of query to be able to construct new values that do not already exist in the database. If we relate our lattice with a relational database, concept-intersections correspond to rows and attributes to columns/attributes. Basic concepts in a concept-intersection that is not an attribute, does not have a counterpart in relational database theory. We will use the operator `&` to indicate this new database natural join query. So if we now ask the query:

```
queries
  db1 & db2
```

we would like to see the following answer:

```
db1 & db2:
<[db1, db2, A(a), A(a1), B(b), B(b1), C(c), C(r), C(r1), D(d), D(d2)]
 [db1, db2, A(a), A(a2), B(b), B(b2), C(c), C(s), C(s1), D(d), D(d4)]
 [db1, db2, A(a), A(a3), B(b), B(b1), C(c), C(t), C(t2), D(d), D(d5)]
 [db1, db2, A(a), A(a4), B(b), B(b3), C(c), C(r), C(r2), D(d), D(d1)]
 [db1, db2, A(a), A(a5), B(b), B(b2), C(c), C(s), C(s2), D(d), D(d4)]>
```

i.e. we have combined those rows from $db1$ and $db2$ for which the common C-attribute arguments have a meet different from $\perp$ when evaluated in the actual database. To define such a database join query we first define what corresponds to a database cartesian product:

321.0 $DBCartPrd : PO \to CI\text{-set} \times CI\text{-set} \to CI\text{-set}$
.1 $DBCartPrd\,(p)(ac_1, ac_2) \triangleq$
.2 $\quad \{mk\text{-}CI(cs_1 \cup cs_2) \mid mk\text{-}CI(cs_1) \in ac_1, mk\text{-}CI(cs_2) \in ac_2\}$

The function $DBCartProd$ (321) defined above makes a combination of all possible pairs of concept-intersections. Notice, that this is close to the $Meet_N$ operation (51), but the result is not projected into p. To get a database join we use the following function:

322.0 $DBJoin : NEq\text{-set} \to PO \to CI\text{-set} \times CI\text{-set} \to CI\text{-set}$
.1 $DBJoin\,(p)(neqs)(ac_1, ac_2) \triangleq$
.2 $\quad \text{let } jcis = \bigcup\{JoinCI(p)(ci_1, ci_2) \mid ci_1 \in ac_1, ci_2 \in ac_2\} \text{ in}$
.3 $\quad cProj(neqs)(jcis)$

Here the function $JoinCI$ does the joining for each pair of ci's (or rows). Finally, the joined rows are projected into $pmax$ to make sure that the result satisfies the given equations. The next function does the joining of each ci-pair.

323.0 $JoinCI : PO \to CI \times CI \to CI\text{-set}$
.1 $JoinCI\,(p)(ci_1, ci_2) \triangleq$
.2 $\quad \text{let } mk\text{-}(attrNs1, attrNs2) = mk\text{-}(AttrsInCI(ci_1), AttrsInCI(ci_2)),$
.3 $\qquad mk\text{-}(mk\text{-}CI(bs_1), mk\text{-}CI(bs_2)) = mk\text{-}(ci_1, ci_2) \text{ in}$
.4 $\quad \text{let } cattrNs = attrNs1 \cap attrNs2,$
.5 $\qquad bs12 = bs_1 \cup bs_2 \text{ in}$
.6 $\quad \text{let } cattrs = \{mk\text{-}At(a, b) \mid mk\text{-}At(a, b) \in bs12 \cdot a \in cattrNs\} \text{ in}$
.7 $\quad \text{let } jbs1 = bs12 \setminus cattrs,$
.8 $\qquad joinedAttrs = \{EvalAttrArg(p)(\alpha)(mk\text{-}CI(cattrs)) \mid \alpha \in cattrNs\} \text{ in}$
.9 $\quad CisMeet_{NCAP}(\{mk\text{-}CI(jbs1)\} \cup joinedAttrs)$

where

324.0 $EvalAttrArg : PO \to A \times CI \to CI\text{-set}$
.1 $EvalAttrArg\,(p)(\alpha)(ci) \triangleq$
.2 $\quad \text{let } attrarg = AttrArgCI(\alpha)(ci) \text{ in}$
.3 $\quad \text{let } attrval = CISproj(p)(\{attrarg\}) \text{ in}$
.4 $\quad \{attrCI(\alpha)(cix) \mid cix \in attrval\}$

In $JoinCI$ the two concept-intersection's ci_1 and ci_2 should be joined iff the common attributes have overlapping arguments. The resulting join will be constructed from the set of basic concepts ($bs12$) from ci_1 and ci_1. *cattrsNs* is the set of common attribute names and *cattrs* is the set of basic concepts from $bs12$ constituting the common attributes. The resulting join will first of all consist of the set of basic concepts $jbs1$ that consist of the original set $bs12$ minus those from the common attributes. Concerning the common attributes the arguments should be evaluated in the actual database and the original arguments replaced by these new values; the result is *joinedAttrs*. Notice that this evaluated argument part is now a *CI*-set. Finally the first part ($jbs1$) – converted to a *CI*-set – and the *CI*-set's in *joinedAttrs* must be *Meet*'ed together.

Example The examples below illustrate the evaluation of the attribute arguments

```
terms
  db1 * A(a)*R(r),
  db2* R(s)* B(b),
  r*s*U
queries
  db1 & db2;
```

We get the answer

```
query: db1 & db2
  <[db1,db2,A(a),R(r),R(s),R(U),B(b)]>
```

Example Here we add an equation to the above example

```
equations
  r*s*U= X + Y
terms
  db1 * A(a)*R(r*s),
  db2* R(s*t)* B(b),
  r*s*t*U
queries
  db1 & db2;
```

We now get the answer

```
query: db1 & db2
< [db1,db2,A(a),R(r),R(s),R(U),R(Y),B(b)],
  [db1,db2,A(a),R(r),R(s),R(U),R(X),B(b)] >
```

17 Conclusion

In this project we have investigated the possibilities to make a system based on the concept algebra described in [3], [4] and [5]. One of the main ideas in this work has been to use Birkhoff's representation theorem, so we have represented distributive lattices using its dual representation: the partial order of join irreducibles. We have given solutions to the questions: how do we construct a concept algebra/distributive lattice satisfying a given set of equations?, and among all the possible solutions which one to choose? Here the most important contribution seems to be the idea of inserting terms in the lattice and the answer to the question: what does it mean to insert a term in a lattice? To solve this we invented the concept of the most disjoint lattice with respect to a given set of inserted terms, that is the smallest lattice where the inserted terms preserve their value compared to the value in the initial algebra/lattice. And that is how the database turned up. The partial order corresponding to the most disjoint lattice is the database; it grows when new terms are inserted and always contains just the information necessary to give the value of (or explain) the inserted terms.

We attacked the problem in two steps. First we considered the concept algebra without attributes. Here we managed to prove the correctness of the algorithms used to construct the database. Next we considered the full concept algebra. The introduction of attributes increased the complexity of the problem considerably. We proved several fundamental properties that made it possible to construct the algorithm used to construct the database. We did not manage to give a mathematical proof for the final algorithm.

The specification of the database (see appendix A) was used to construct a prototype for the database system (called LatBase). This system has been used with several examples and seems to yield the expected results. Thus, the main algorithm for construction of the database seems to have been proved “by experiment”.

There are several problems to consider in the future. First of all much of the mathematics – especially in the second part – became rather complex and ad hoc and unfinished. This should be reconsidered, but we need some more expert knowledge in advanced lattice theory and universal algebra to do that job. Next, the LatBase system should be implemented as a real database system and this also includes to make efficient versions of the algorithms for constructing and querying the database.

Acknowledgment

I would like to thank prof. Jørgen Fischer Nilsson for suggesting this project for me several years ago and his continued encouragement to work with the project.

Appendix

A The Final Database System

This section contains all the specifications needed to implement the final LatBase database system. So it is essentially a collection of some of the specifications from the previous sections extended with a few new features that will be useful in a practical database system. First of all, in the real database system we want to be able to associate information to the inserted terms. Information that are inserted together with a term will be associated to the resulting concept-intersections. Here we will not decide on the nature of the associated information, but it could for instance just be a unique name.

types

325.0 $Info = \mathsf{token}$ — Information associated with inserted term

Secondly we will add two predefined equations in order to make integers and strings available in the inserted values:

```
NUMBER >= 0 + 1 + 2 + 3 + ...
STRING >= ‘‘’’ + ‘‘a’’ + ‘‘b’’ + ...+ ‘‘aa’’ + ‘‘ab’’ + ...
```

Finally we also add (a huge amount of) equations of the form

```
BOTTOM= literal1 * literal2
```

in order to specify that all the predefined integer and string literals are disjoint.

A.1 Concept Intersections with Associated Information

types

326.0 $CN = \mathsf{token} \mid \mathtt{STRING} \mid \mathtt{NAT};$ — The type of named concept constants

327.0 $C = CN \mid \mathbb{N} \mid \mathbf{String};$ — The type of concept constants

A concept constant (C) is either an arbitrary name including one of the predefined names `STRING` and `NAT` or an integer or string literal like `23` and `"abc"`.

types

328.0 $A = \mathsf{token}$ — The type of attributes, $\alpha, \beta, \ldots : A$

329.0 $B = C \mid At;$ — The type of basic concepts, $a, b, \alpha(a), \alpha(\beta(a)), \ldots : B$

330.0 $At :: \ A \times B;$ — The type of attributions, $\alpha(a), \alpha(\beta(a)), \ldots : At$

331.0 $Bset = B\text{-}\mathsf{set}$

.1 $\mathsf{inv}\ (bset) \triangleq bset \neq \{\};$

332.0 $CI :: \ Bset;$ — The type of concept intersections

333.0 $CII :: \ CI \times Info\text{-}\mathsf{set};$ — concept-intersection with associated set of Infos

334.0 $PO = CI\text{-set};$

335.0 $POI = CII\text{-set}$ — the partial order with Info

In the new partial order we need to redefine some of the basic operations on the partial order:

336.0 $ISA_{PI} : CII \times CII \to \mathbb{B}$ — $\sim$ 151

.1 $ISA_{PI}\,(mk\text{-}CII(CI(cs_1), \text{-}), mk\text{-}CII(CI(cs_2), \text{-}) \triangleq cs_2 \subseteq cs_1$

337.0 $IsAntiChainI : CII\text{-set} \to \mathbb{B}$ — $\sim$ 8

.1 $IsAntiChainI\,(ac) \triangleq$

.2 $\quad \forall\, ci_1 \in ac, ci_2 \in ac \cdot ci_1 \neq ci_2 \;\Rightarrow\; \neg\, ISA_{PI}(ci_1, ci_2) \wedge \neg\, ISA_{PI}(ci_2, ci_1)$

338.0 $AntiCh\,(cis : CI\text{-set})\;\, ac : CI\text{-set}$ — $\sim$ 17

.1 post $ac \subseteq cis \wedge IsAntiChain(ac) \wedge \forall\, ci_1 \in cis \cdot \exists\, ci_2 \in ac \cdot ISA_P(ci_1, ci_2)$

339.0 $AntiChI\,(cis : CII\text{-set})\;\, ac : CII\text{-set}$

.1 post $ac \subseteq cis \wedge IsAntiChainI(ac) \wedge \forall\, ci_1 \in cis \cdot \exists\, ci_2 \in ac \cdot ISA_{PI}(ci_1, ci_2)$

340.0 $DownSetIC : POI \to CI\text{-set} \to CII\text{-set}$ — $\sim$ 23

.1 $DownSetIC\,(p)(ciset) \triangleq$

.2 $\quad \{mk\text{-}CII(ci, infs) \mid mk\text{-}CII(ci, infs) \in p \cdot \exists\, ci' \in ciset \cdot ISA_P(ci, ci')\}$

341.0 $stripInfo : CII\text{-set} \to CI\text{-set}$

.1 $stripInfo\,(cis) \triangleq \{ci \mid mk\text{-}CII(ci, \text{-}) \in cis\}$

342.0 $CISprojI\,(p : POI)(cis : CI\text{-set})\;\, ac : CII\text{-set}$ — $\sim$ 26

.1 post $ac \subseteq p \wedge$

.2 $\quad IsAntiChainI(ac) \wedge$

.3 $\quad DownSetIC(p)(stripInfo(ac)) = DownSetC(p)(cis)$

A.2 Terms

types

343.0 $BasicTerm = C \mid \text{TOP} \mid \text{BOTTOM};$

344.0 $Term = Join \mid Meet \mid Attr \mid BasicTerm;$

345.0 $Join :: \;\; Term \times Term$ — Join-term ;

346.0 $Meet :: \;\; Term \times Term$ — Meet-term ;

347.0 $Attr :: \;\; A \times Term$ — Attribute-term

These general terms will only be used when making queries to the lattice database. When inserting terms in the lattice database we will only allow terms not containing the TOP-term, and finally in the equations we will only allow terms without the TOP-term and literals. Below we define these two kinds of restricted terms

types

348.0 $termIr = Term$

.1 inv $tir \triangleq$ TOP $\notin BTerms(tir)$

;

349.0 $termEr = Term$

.1 inv $ter \triangleq$ TOP $\notin BTerms(ter) \wedge \neg \exists\, e \in BTerms(ter) \cdot is\text{-}\mathbb{N}(e) \vee is\text{-}\mathbf{String}(e)$;

350.0 $Eq :: \quad termEr \times termEr$ — term-equation

where

351.0 $BTerms : Term \to BasicTerm\text{-set}$ — $\sim$ 141

.1 $BTerms\,(t) \triangleq$

.2 cases t :

.3 $mk\text{-}Join(t_1, t_2) \to BTerms(t_1) \cup BTerms(t_2)$,

.4 $mk\text{-}Meet(t_1, t_2) \to BTerms(t_1) \cup BTerms(t_2)$,

.5 $mk\text{-}Attr(\alpha, t) \to BTerms(t)$,

.6 $bt \to \{bt\}$

.7 end

A.3 Evaluation of Terms

352.0 $EvalI_{NCA} : POI \to Term \to CII\text{-set}$ — $\sim$ 264 — $\sim$ 264

.1 $EvalI_{NCA}\,(p)(t) \triangleq$

.2 cases t :

.3 $mk\text{-}Join(t_1, t_2) \to JoinI_N(p)(EvalI_{NCA}(p)(t_1), EvalI_{NCA}(p)(t_2))$,

.4 $mk\text{-}Meet(t_1, t_2) \to MeetI_N(p)(EvalI_{NCA}(p)(t_1), EvalI_{NCA}(p)(t_2))$,

.5 $mk\text{-}Attr(\alpha, t) \to AttributionI_N(p)(\alpha)(EvalI_{NCA}(p)(t))$,

.6 (TOP) $\to AntiChI(p)$,

.7 (BOTTOM) $\to \{\}$,

.8 $c \to cEvalI_N(p)(c)$

.9 end

353.0 $JoinI_N : POI \to CII\text{-set} \times CII\text{-set} \to CII\text{-set}$ — $\sim$ 50

.1 $JoinI_N\,(p)(ac_1, ac_2) \triangleq AntiChI(ac_1 \cup ac_2)$

354.0 $MeetI_N : POI \rightarrow CII\text{-set} \times CII\text{-set} \rightarrow CII\text{-set}$ — $\sim$ 51

.1 $MeetI_N\ (p)(ac_1, ac_2) \triangleq$
.2 $\quad$ let $cis = \{mk\text{-}CI(cs_1 \cup cs_2) \mid$
.3 $\qquad\qquad mk\text{-}CII(mk\text{-}CI(cs_1), \text{-}) \in ac_1, mk\text{-}CII(mk\text{-}CI(cs_2), \text{-}) \in ac_2\}$ in
.4 $\quad CISprojI(p)(cis)$

355.0 $AttributionI_N : POI \rightarrow A \rightarrow CII\text{-set} \rightarrow CII\text{-set}$ — $\sim$ 263

.1 $AttributionI_N\ (p)(\alpha)(ac) \triangleq CISprojI(p)(\{attrCI(\alpha)(ci) \mid mk\text{-}CII(ci, \text{-}) \in ac\})$

356.0 $attrCI : A \rightarrow CI \rightarrow CI$ — $\sim$ 170

.1 $attrCI\ (\alpha)(mk\text{-}CI(bs)) \triangleq mk\text{-}CI(\{mk\text{-}At(\alpha, b) \mid b \in bs\})$

357.0 $cEvalI_N : POI \rightarrow C \rightarrow CII\text{-set}$ — $\sim$ 53

.1 $cEvalI_N\ (p)(c) \triangleq CISproj(p)(\{mk\text{-}CI(\{c\})\})$

A.4 Evaluation in the Power Set Partial Order

358.0 $Eval'_{NCAP} : termEr \rightarrow CI\text{-set}$ — $\sim$ 269

.1 $Eval'_{NCAP}\ (t) \triangleq$
.2 $\quad$ cases t :
.3 $\qquad mk\text{-}Join(t_1, t_2) \rightarrow Join_{NCAP}(Eval'_{NCAP}(t_1), Eval'_{NCAP}(t_2)),$
.4 $\qquad mk\text{-}Meet(t_1, t_2) \rightarrow Meet_{NCAP}(Eval'_{NCAP}(t_1), Eval'_{NCAP}(t_2)),$
.5 $\qquad mk\text{-}Attr(\alpha, t) \rightarrow \{attrCI(\alpha)(ci) \mid ci \in Eval'_{NCAP}(t)\},$
.6 $\qquad (\textsc{bottom}) \rightarrow \{\},$
.7 $\qquad c \rightarrow \{mk\text{-}CI(\{c\})\}$
.8 $\quad$ end

359.0 $Join_{NCAP} : CI\text{-set} \times CI\text{-set} \rightarrow CI\text{-set}$ — $\sim$ 267

.1 $Join_{NCAP}\ (ac_1, ac_2) \triangleq AntiCh(ac_1 \cup ac_2)$

360.0 $Meet_{NCAP} : CI\text{-set} \times CI\text{-set} \rightarrow CI\text{-set}$ — $\sim$ 268

.1 $Meet_{NCAP}\ (ac_1, ac_2) \triangleq$
.2 $\quad$ let $cis = \{mk\text{-}CI(bs_1 \cup bs_2) \mid mk\text{-}CI(bs_1) \in ac_1, mk\text{-}CI(bs_2) \in ac_2\}$ in
.3 $\quad AntiCh(cis)$

A.5 Constructing the Database

types

361.0 $NEq :: \ CI\text{-set} \times CI\text{-set}$;

362.0 $DB :: \ NEq\text{-set} \times POI$

Create a new empty database with a given set of equations:

363.0 $crDataBase : Eq\text{-set} \rightarrow DB$ — $\sim$ part of 320

.1 $crDataBase\,(eqs) \triangleq mk\text{-}DB(EvalEqs(eqs), \{\})$

where

364.0 $EvalEq : Eq \rightarrow NEq$ — $\sim$ 290

.1 $EvalEq\,(mk\text{-}Eq(t_1, t_2)) \triangleq$

.2 $\quad$ let $ac_1 = Eval'_{NCAP}(t_1)$,

.3 $\qquad ac_2 = Eval'_{NCAP}(t_2)$ in

.4 $\quad mk\text{-}NEq(Join_{NCAP}(ac_1, ac_2), Meet_{NCAP}(ac_1, ac_2))$

365.0 $EvalEqs : Eq\text{-set} \rightarrow NEq\text{-set}$ — $\sim$ 291

.1 $EvalEqs\,(eqs) \triangleq \{EvalEq(eq) \mid eq \in eqs\}$

Insert a term with information in the database:

366.0 $insertTerm : DB \rightarrow termIr \times Info \rightarrow DB$ — $\sim$ part of 320

.1 $insertTerm\,(mk\text{-}DB(neqs, p))(term, info) \triangleq$

.2 $\quad$ let $cis = extCIS(cEvalNCA(neqs)(term))$ in

.3 $\quad$ let $newpoi = \{mkCII(ci, infs) \mid mk\text{-}CII(ci, infs) \in p \cdot ci \notin cis\} \cup$

.4 $\qquad\qquad \{mk\text{-}CII(ci, infs \cup \{info\}) \mid mk\text{-}CII(ci, infs) \in p \cdot ci \in cis\} \cup$

.5 $\qquad\qquad \{mk\text{-}CII(ci, \{info\}) \mid ci \in cis \cdot \neg\exists\, mk\text{-}CII(ci', \text{-}) \in p \cdot ci = ci'\}$ in

.6 $\quad mk\text{-}DB(neqs, newpoi)$

A.5.1 The Algorithm for Computing $Eval_{NCA}$

367.0 $cEvalNCA : NEq\text{-set} \rightarrow termIr \rightarrow CI\text{-set}$ — $\sim$ 313

.1 $cEvalNCA\,(neqs)(t) \triangleq cProj(neqs)(Eval'_{NCAP}(t))$

368.0 $cProj : NEq\text{-set} \rightarrow CI\text{-set} \rightarrow CI\text{-set}$ — $\sim$ 314

.1 $cProj\,(neqs)(cis) \triangleq$

.2 $\quad$ let $mk\text{-}(\text{-}, pcis) = cIProj(neqs)(\mathsf{true})(cis, \{\})(\{\})$ in

.3 $\quad pcis$

The Iterated Projection

369.0 $cIProj : NEq\text{-set} \rightarrow \mathbb{B} \rightarrow CI\text{-set} \times CI\text{-set} \rightarrow CI\text{-set} \rightarrow \mathbb{B} \times CI\text{-set}$ — $\sim$ 315

.1 $cIProj\,(neqs)(accUnch)(upl, InP)(pLevCis) \triangleq$

.2 $\quad$ let $mk\text{-}(newunch, newupl, newInP) = cProjStep(neqs)(upl, InP)(pLevCis)$ in

.3 $\quad$ if $newunch$

.4 $\quad$ then $mk\text{-}(accUnch, AntiCh(newInP))$

.5 $\quad$ else $cIProj(neqs)(\mathsf{false})(newupl, newInP)(pLevCis)$

The Projection Step.

370.0 $cProjStep : NEq\text{-set} \rightarrow CI\text{-set} \times CI\text{-set} \rightarrow CI\text{-set} \rightarrow \mathbb{B} \times CI\text{-set} \times CI\text{-set}$ — $\sim$ 316
.1 $cProjStep\,(neqs)(upl, InP)(pLevCis) \triangleq$
.2 $\quad$ let $cupls = \{cEqsUpl(neqs)(pLevCis)(ci) \mid ci \in upl\}$ in
.3 $\quad$ let $InP1 = \{ci \mid mk\text{-}((\mathsf{true}), \{ci\}) \in cupls\}$,
.4 $\qquad notInP = \{upl' \mid mk\text{-}((\mathsf{false}), upl') \in cupls\}$,
.5 $\qquad newupl = \bigcup notInP$ in
.6 $\quad mk\text{-}(notInP = \{\}, newupl, InP \cup InP1)$

Projecting a Single concept-intersection.

371.0 $CisMeet_{NCAP} : CI\text{-set-set} \rightarrow CI\text{-set}$ — $\sim$ 317
.1 $CisMeet_{NCAP}\,(ciss) \triangleq$
.2 $\quad$ cases $ciss$:
.3 $\qquad \{cis\} \rightarrow cis$,
.4 $\qquad \{cis\} \cup ciss' \rightarrow Meet_{NCAP}(cis, CisMeet_{NCAP}(ciss'))$
.5 $\quad$ end
.6 pre $\;ciss \neq \{\}$

372.0 $cEqsUpl : NEq\text{-set} \rightarrow CI\text{-set} \rightarrow CI \rightarrow \mathbb{B} \times CI\text{-set}$ — $\sim$ 318
.1 $cEqsUpl\,(neqs)(pLevCis)(ci) \triangleq$
.2 $\quad$ let $uls1 = cEqsDirectUpl(neqs)(ci)$,
.3 $\qquad uls2 = \{iupl \mid \alpha \in AttrsInCI(ci), iupl : CI\text{-set} \cdot$
.4 $\qquad\qquad mk\text{-}(\mathsf{false}, iupl) = cIUpl(neqs)(\alpha)(pLevCis)(ci)\}$ in
.5 $\quad$ let $uls = uls1 \cup uls2$ in
.6 $\quad$ cases uls :
.7 $\qquad \{\} \rightarrow mk\text{-}(\mathsf{true}, \{ci\})$,
.8 $\qquad \text{-} \rightarrow mk\text{-}(\mathsf{false}, CisMeet_{NCAP}(\{\{ci\}\} \cup uls))$
.9 $\quad$ end

373.0 $cEqsDirectUpl : NEq\text{-set} \rightarrow CI \rightarrow CI\text{-set-set}$
.1 $cEqsDirectUpl\,(neqs)(ci) \triangleq$
.2 $\quad$ let $uls1 = \{m \mid mk\text{-}NEq(u, m) \in neqs \cdot InEqDirectRej(mk\text{-}NEq(u, m))(ci)\}$,
.3 $\qquad uls2 = cBuildInEqsDirectUpl(ci)$ in
.4 $\quad uls1 \cup uls2$

374.0 $cBuildInEqsDirectUpl : CI \rightarrow CI\text{-set-set}$
.1 $cBuildInEqsDirectUpl\,(mk\text{-}CI(bs)) \triangleq$
.2 $\quad \{mk\text{-}CI(\{\mathtt{NUMBER}, c\}) \mid c \in bs \cdot is\text{-}Integer(c) \wedge \mathtt{NUMBER} \notin bs\} \cup$
.3 $\quad \{mk\text{-}CI(\{\mathtt{STRING}, c\}) \mid c \in bs \cdot is\text{-}String(c) \wedge \mathtt{STRING} \notin bs\} \cup$
.4 $\quad$ (if $\exists\, b_1, b_2 \in bs \cdot b_1 \neq b_2 \wedge \forall\, b \in \{b_1, b_2\} \cdot is\text{-}Integer(b) \vee is\text{-}String(c)$
.5 $\quad$ then $\{\{\}\}$
.6 $\quad$ else $\{\}$)

Lines 2 and3 above correspond to the equations

```
INTEGER >= 0 + 1 + 2 + 3 + ...
STRING >= ‘‘’’ + ‘‘a’’ + ‘‘b’’ + ...+ ‘‘aa’’ + ‘‘ab’’ + ...
```

and line 4 corresponds to equations of the form

```
litteral1 * literal2 = BOTTOM
```

i.e. if an inserted concept-intersection contains two different string or number literals it will have `BOTTOM` as an upper bound.

375.0 $cIUpl : NEq\text{-set} \rightarrow A \rightarrow CI\text{-set} \rightarrow CI \rightarrow \mathbb{B} \times CI\text{-set}$ — $\sim$ 319

.1 $cIUpl\,(neqs)(\alpha)(pLevCis)(ci) \triangleq$

.2 $\quad$ let $ci_{arg} = AttrArgCI(\alpha)(ci)$ in

.3 $\quad$ let $mk\text{-}(unch, proj) =$

.4 $\qquad\qquad$ if $ci_{arg} \in pLevCis$ then $mk\text{-}(\mathsf{false}, \{\})$

.5 $\qquad\qquad$ else $cIProj(neqs)(\mathsf{true})(\{ci_{arg}\}, \{\})(pLevCis \cup \{ci_{arg}\})$ in

.6 $\quad$ cases $unch$:

.7 $\qquad$ $\mathsf{false} \rightarrow mk\text{-}(\mathsf{false}, atCIs(\alpha)(proj))$,

.8 $\qquad$ $\mathsf{true} \rightarrow mk\text{-}(\mathsf{true}, \{ci\})$

.9 $\quad$ end

Equation Reject The function *InEqDirectRej* defined below tells if a concept-intersection is directly rejected by an equation by being in the equations rejection region:

376.0 $InEqDirectRej : NEq \rightarrow CI \rightarrow \mathbb{B}$ — $\sim$ 292

.1 $InEqDirectRej\,(mk\text{-}NEq(u, m))(ci) \triangleq$

.2 $\quad ISA_N(\{ci\}, u) \wedge \neg ISA_N(\{ci\}, m)$

Extracting Attributes

377.0 $AttrsInCI : CI \rightarrow A\text{-set}$ — $\sim$ 162

.1 $AttrsInCI\,(mk\text{-}CI(bs)) \triangleq \{a \mid mk\text{-}At(a, \text{-}) \in bs\}$

378.0 $AttrArgCI : A \rightarrow CI \rightarrow CI$ — $\sim$ 178

.1 $AttrArgCI\,(\alpha)(ci) \triangleq mk\text{-}CI(\{b \mid mk\text{-}At(\text{-}, b) \in CbattrsCI(\alpha)(ci)\})$

.2 pre $\alpha \in AttrsInCI(ci)$

Constructing Attribute Consistent Partial Orders

379.0 $extCIS : CI\text{-set} \rightarrow CI\text{-set}$ — $\sim$ 379

.1 $extCIS\,(cis) \triangleq$

.2 $\quad$ let $attrArg = \{AttrArgCI(\alpha)(ci) \mid ci \in cis, \alpha \in AttrsInCI(ci)\}$ in

.3 $\quad$ if $attrArg = \{\}$ then cis else $cis \cup extCIS(attrArg)$

A.6 Querying the Database

380.0 $Query = TermQ \mid DownsetQ \mid DBnJoinQ$

381.0 $TermQ :: \ \ Term$ — term query

382.0 $DownsetQ :: \ \ Term$ — downset query

383.0 $DBnJoinQ :: \ \ Query \times Query$ — database natural join query

We consider just three query constructs. A basic term query is just term-evaluation in the actual partial order resulting in an antichain with an information set associated to each concept-intersection. The downset query is almost like the term query, but results in a downset in the actual partial order. Finally we have the data base natural join as discussed in section 16.1.

384.0 $EvalQuery : DB \rightarrow Query \rightarrow CII\text{-set}$
.1 $EvalQuery\,(db)(q) \triangleq$
.2 let $mk\text{-}DB(neqs, p) = db$ in
.3 cases q :
.4 $mk\text{-}TermQ(t) \rightarrow EvalI_{NCA}(p)(t)$,
.5 $mk\text{-}DownsetQ(t) \rightarrow DownSetIC(p)(stripInfo(EvalI_{NCA}(p)(t)))$,
.6 $mk\text{-}DBnJoinQ(q_1, q_2) \rightarrow$
.7 let $ciis_1 = EvalQuery(db)(q_1)$,
.8 $ciis_2 = EvalQuery(db)(q_2)$ in
.9 $DBJoinI(p)(neqs)(ciis_1, ciis_2)$
.10 end

385.0 $DBJoinI : POI \rightarrow NEq\text{-set} \rightarrow CII\text{-set} \times CII\text{-set} \rightarrow CII\text{-set}$ — $\sim$ 322
.1 $DBJoinI\,(p)(neqs)(ciis_1, ciis_2) \triangleq$
.2 let $jciis = \bigcup\{JoinCII(p)(cii_1, cii_2) \mid cii_1 \in ciis_1, cii_2 \in ciis_2\}$ in
.3 $cProjI(neqs)(jciis)$

In *JoinCII* the two concept-intersection's with information cii_1 and cii_2 will be joined iff the common attributes have overlapping arguments. When two rows/CII's are joined we must consider what to do with the associated information. A solution would be to associate to the new joined row the union of the information sets from the original rows. Thus *JoinCII* will correspond to *JoinCI* in 323 but with some trivial complications. We will however not elaborate further on that topic in this report.

B Proofs

B.1 Proof of Term Value Property 42.1

Below is a proof of the term value property 42.1 repeated here:

386.0 $Eval_L(p \setminus cis)(t) = Eval_L(p)(t) \setminus cis$

The proof uses structural induction on the term structure. First 386 must be proved correct for basic terms i.e. terms without sub-terms:

BOTTOM:

$$\begin{aligned}&\mathit{Eval}_L(p \setminus \mathit{cis})(\text{BOTTOM}) = \{\} = \{\} \setminus \mathit{cis}\\ &= \mathit{Eval}_L(p)(\text{BOTTOM}) \setminus \mathit{cis}\end{aligned}$$

TOP:

$$\begin{aligned}&\mathit{Eval}_L(p \setminus \mathit{cis})(\text{TOP}) = p \setminus \mathit{cis}\\ &= \mathit{Eval}_L(p)(\text{TOP}) \setminus \mathit{cis}\end{aligned}$$

Named concept c:

$$\begin{aligned}&\mathit{Eval}_L(p \setminus \mathit{cis})(c) = c\mathit{Value}_L(p \setminus \mathit{cis})(c)\\ &= \mathit{DownSet}(p \setminus \mathit{cis})(\{\textit{mk-CI}(\{c\})\})\\ &\qquad\qquad\text{from 14}\\ &= \mathit{DownSet}(\mathcal{P}(\mathit{cset}))(\{\textit{mk-CI}(\{c\})\}) \cap (p \setminus \mathit{cis})\\ &= (\mathit{DownSet}(\mathcal{P}(\mathit{cset}))(\{\textit{mk-CI}(\{c\})\}) \cap p) \setminus \mathit{cis}\\ &\qquad\qquad\text{from 14}\\ &= \mathit{DownSet}(p)(\{\textit{mk-CI}(\{c\})\}) \setminus \mathit{cis}\\ &= c\mathit{Value}_L(p)(c) \setminus \mathit{cis}\\ &= \mathit{Eval}_L(p)(c) \setminus \mathit{cis}\end{aligned}$$

Next, in the induction steps, when considering compound terms, assume 386 is true for the sub-terms (the induction hypothesis):

$\textit{mk-Join}(t_1, t_2)$:

$$\begin{aligned}&\mathit{Eval}_L(p \setminus \mathit{cis})(\textit{mk-Join}(t_1, t_2))\\ &\qquad\qquad\text{from 41}\\ &= \mathit{Join}_L(\mathit{Eval}_L(p \setminus \mathit{cis})(t_1), \mathit{Eval}_L(p \setminus \mathit{cis})(t_2))\\ &\qquad\qquad\text{from 30}\\ &= \mathit{Eval}_L(p \setminus \mathit{cis})(t_1) \cup \mathit{Eval}_L(p \setminus \mathit{cis})(t_2)\\ &\qquad\qquad\text{from induction hypothesis}\\ &= (\mathit{Eval}_L(p)(t_1) \setminus \mathit{cis}) \cup (\mathit{Eval}_L(p)(t_2) \setminus \mathit{cis})\\ &= (\mathit{Eval}_L(p)(t_1) \cup \mathit{Eval}_L(p)(t_2)) \setminus \mathit{cis}\\ &= \mathit{Join}_L(\mathit{Eval}_L(p)(t_1), \mathit{Eval}_L(p)(t_2)) \setminus \mathit{cis}\\ &\qquad\qquad\text{from 41}\\ &= \mathit{Eval}_L(p)(\textit{mk-Join}(t_1, t_2)) \setminus \mathit{cis}\end{aligned}$$

$\textit{mk-Meet}(t_1, t_2)$: Similar to the proof for Join.

B.2 Proof of Term Value Property 42.2

Below is a proof of the term value property 42.2:

387.0 $\quad p_1 \subseteq p_2 \;\Rightarrow\; \mathit{Eval}_L(p_1)(t) = \mathit{Eval}_L(p_2)(t) \cap p_1$

The proof is based on the properties 42.0 and 42.1.

$$\begin{aligned}&\mathit{Eval}_L(p_2 \setminus \mathit{cs})(t)\\ &\qquad\qquad\text{from 42.1}\\ &= \mathit{Eval}_L(p_2)(t) \setminus \mathit{cs}\end{aligned}$$

from 42.0

$= (Eval_L(p_2)(t) \cap p_2) \setminus cs$

$= Eval_L(p_2)(t) \cap (p_2 \setminus cs)$

Hence, if $p_2 \setminus cs$ is replaced by p_1 we get

388.0 $p_1 \subseteq p_2 \;\Rightarrow\; Eval_L(p_1)(t) = Eval_L(p_2)(t) \cap p_1$

B.3 Proof of Term Value Properties 42.3 and 42.4

Below is a proof of the term value property 42.3:

389.0 $Eval_L(p_1 \cup p_2)(t) = Eval_L(p_1)(t) \cup Eval_L(p_2)(t)$

The proof is based on the equality in 42.2. We instantiate that equality to the two equations below:

390.0 $Eval_L(p_1)(t) = Eval_L(p_1 \cup p_2)(t) \cap p_1$

.1 $Eval_L(p_2)(t) = Eval_L(p_1 \cup p_2)(t) \cap p_2$

By making the union of the two equations in 390 above we get the first line below

$Eval_L(p_1)(t) \cup Eval_L(p_1)(t) = Eval_L(p_1 \cup p_2)(t) \cap p_1 \cup Eval_L(p_1 \cup p_2)(t) \cap p_2$

$= Eval_L(p_1 \cup p_2)(t) \cap (p_1 \cup p_2)$

from $Eval_L(p_1 \cup p_2)(t) \subseteq p_1 \cup p_2$, (42.0)

$= Eval_L(p_1 \cup p_2)(t)$

Next, a proof of the term value property 42.4:

391.0 $Eval_L(p_1 \cap p_2)(t) = Eval_L(p_1)(t) \cap Eval_L(p_2)(t)$

The proof is again based on the equality in 42.2. We instantiate that equality to the two equations below:

392.0 $Eval_L(p_1 \cap p_2)(t) = Eval_L(p_1)(t) \cap (p_1 \cap p_2)$

.1 $Eval_L(p_1 \cap p_2)(t) = Eval_L(p_2)(t) \cap (p_1 \cap p_2)$

By making the intersection of the two equations in 392 above we get the equation below

$Eval_L(p_1 \cap p_2)(t) \cap Eval_L(p_1 \cap p_2)(t)$

$= Eval_L(p_1)(t) \cap (p_1 \cap p_2) \cap Eval_L(p_2)(t) \cap (p_1 \cap p_2)$

By reducing and rearranging we get

$Eval_L(p_1 \cap p_2)(t) = (Eval_L(p_1)(t) \cap p_1) \cap (Eval_L(p_2)(t) \cap p_2)$

which according to 42.0 is equivalent to

$Eval_L(p_1 \cap p_2)(t) = Eval_L(p_1)(t) \cap Eval_L(p_2)(t)$

B.4 Proof of $Meet_N$ Correctness 52

In this section we prove

393.0 $DownSet(AntiCh(DownSet(p)(ac_1) \cap DownSet(p)(ac_2))) =$ — $\sim$ 52

.1 $DownSet$

.2 (let $cis = \{mk\text{-}CI(cs_1 \cup cs_2) \mid mk\text{-}CI(cs_1) \in ac_1, mk\text{-}CI(cs_2) \in ac_2\}$ in

.3 $CISproj(p)(cis))$

Let the downset defined in 393.0 be $ds1$. It is seen to be the set of concept-intersections in p which are below ac_1 and ac_2:

$$ds1 = \{ci \mid ci \in p \cdot \exists\, ci_1 \in ac_1, ci_2 \in ac_2 \cdot ISA_P(ci, ci_1) \wedge ISA_P(ci, ci_2)\}$$

Let the downset defined in 393.1–393.3 be $ds2$. To show that $ds1 = ds2$ we need an auxiliary lemma:

394.0 $ISA_P(mk\text{-}CI(cs), mk\text{-}CI(cs_1 \cup cs_2))$
from 6
.1 $= cs_1 \cup cs_2 \subseteq cs$
.2 $= cs_1 \subseteq cs \wedge cs_2 \subseteq cs$
.3 $= ISA_P(mk\text{-}CI(cs), mk\text{-}CI(cs_1)) \wedge ISA_P(mk\text{-}CI(cs), mk\text{-}CI(cs_2))$

According to the definition of *CISproj* (in 26) $CISproj(p)(cis)$ is the anti-chain in p which has the same downset in p as cis has. Consequently, the downset of this anti-chain is the set of all concept-intersections in p which are below some element in cis:

$$\begin{aligned}
ds2 &= \{ci \mid ci \in p \cdot \\
&\qquad \exists\, ci' \in \{mk\text{-}CI(cs_1 \cup cs_2) \mid mk\text{-}CI(cs_1) \in ac_1, mk\text{-}CI(cs_2) \in ac_2\} \cdot ISA_P(ci, ci')\}\\
&= \{mk\text{-}CI(cs) \mid mk\text{-}CI(cs) \in p \cdot \\
&\qquad \exists\, mk\text{-}CI(cs_1) \in ac_1, mk\text{-}CI(cs_2) \in ac_2 \cdot ISA_P(mk\text{-}CI(cs), mk\text{-}CI(cs_1 \cup cs_2))\}\\
&\qquad\qquad \text{from } 394\\
&= \{mk\text{-}CI(cs) \mid mk\text{-}CI(cs) \in p \cdot \exists\, mk\text{-}CI(cs_1) \in ac_1, mk\text{-}CI(cs_2) \in ac_2 \cdot \\
&\qquad ISA_P(mk\text{-}CI(cs), mk\text{-}CI(cs_1)) \wedge ISA_P(mk\text{-}CI(cs), mk\text{-}CI(cs_2))\}\\
&= ds1
\end{aligned}$$

where the last step was obtained by a little renaming.

B.5 Proof of Attribution Property 173.4

Below is a proof of the attribution property 173.4:

395.0 $Attribution_{CA}(p_1)(\alpha)(cis) \cup Attribution_{CA}(p_2)(\alpha)(cis) = Attribution_{CA}(p_1 \cup p_2)(\alpha)(cis)$

The proof is based on 173.0 and 173.2. First we need an auxiliary result

$$\begin{aligned}
&Attribution_{CA}(p \setminus d)(\alpha)(cis)\\
&\qquad \text{from } 173.2\\
&= Attribution_{CA}(p)(\alpha)(cis) \setminus d\\
&\qquad \text{from } 173.0\\
&= (Attribution_{CA}(p)(\alpha)(cis) \cap p) \setminus d\\
&= Attribution_{CA}(p)(\alpha)(cis) \cap (p \setminus d)
\end{aligned}$$

Hence, if $p \setminus d$ is replaced by p_1 we get

396.0 $p_1 \subseteq p \;\Rightarrow\; Attribution_{CA}(p_1)(\alpha)(cis) = Attribution_{CA}(p)(\alpha)(cis) \cap p_1$

The equality 396 above may be instantiated to the two equations below:

397.0 $Attribution_{CA}(p_1)(\alpha)(cis) = Attribution_{CA}(p_1 \cup p_2)(\alpha)(cis) \cap p_1$
.1 $Attribution_{CA}(p_2)(\alpha)(cis) = Attribution_{CA}(p_1 \cup p_2)(\alpha)(cis) \cap p_2$

By making the union of the two equations in 397 above we get the first line below

$$\begin{aligned}
&Attribution_{CA}(p_1)(\alpha)(cis) \cup Attribution_{CA}(p_1)(\alpha)(cis)\\
&= Attribution_{CA}(p_1 \cup p_2)(\alpha)(cis) \cap p_1 \cup Attribution_{CA}(p_1 \cup p_2)(\alpha)(cis) \cap p_2\\
&= Attribution_{CA}(p_1 \cup p_2)(\alpha)(cis) \cap (p_1 \cup p_2)
\end{aligned}$$

from $Attribution_{CA}(p_1 \cup p_2)(\alpha)(cis) \subseteq p_1 \cup p_2$, (173.0)

$= Attribution_{CA}(p_1 \cup p_2)(\alpha)(cis)$

B.6 Proof of Downset Intersection Property 210

We prove the general downset intersection property 210 mentioned in section 11.1. It is repeated below:

398.0 $\forall\, cis_1, cis_2 : CI\text{-set} \cdot$

.1 $DownSetC(p)(cis_1 \cap cis_2) \subseteq DownSetC(p)(cis_1) \cap DownSetC(p)(cis_2)$

If $DownSetC(p)(cis_1 \cap cis_2) = \{\}$ then the inclusion is obvious. So below assume $DownSetC(p)(cis_1 \cap cis_2) \neq \{\}$. We prove every element in the left-hand side set is also in the right-hand side set. So assume $ci : CI$ is an arbitrary concept-intersection such that

$ci \in DownSetC(p)(cis_1 \cap cis_2)$

Then, according to the definition of $DownSetC$ (23) we have

$\exists\, ci' \in p \cdot ci' \in cis_1 \cap cis_2 \wedge ISA_P(ci, ci')$

which can easily be rearranged to

$\exists\, ci' \in p \cdot (ci' \in cis_1 \wedge ISA_P(ci, ci')) \wedge \exists\, ci' \in p \cdot (ci' \in cis_2 \wedge ISA_P(ci, ci'))$

According to the definition of $DownSetC$ this is equivalente to

$ci \in DownSetC(p)(cis_1) \wedge ci \in DownSetC(p)(cis_2)$

which is trivially equivalent to

$ci \in DownSetC(p)(cis_1) \cap DownSetC(p)(cis_2)$

B.7 Proof of Term Value Property 175.0

Below is a proof of the term value property 175.0 repeated here:

399.0 $Eval_{CA}(p)(t) \subseteq p$

The proof uses structural induction on the term structure. First 399 must be proved correct for basic terms i.e. terms without sub-terms:

BOTTOM:

$Eval_{CA}(p)(\text{BOTTOM}) = \{\} \subseteq p$

TOP:

$Eval_{CA}(p_1)(\text{TOP}) = p \subseteq p$

Named concept c:

$Eval_{CA}(p)(c) = cValue_L(p)(c)$

$= DownSet(p)(\{mk\text{-}CI(\{c\})\})$

from 14

$\subseteq p$

Next, in the induction steps, when considering compound terms, assume 399 is true for the sub-terms (the induction hypothesis):

$mk\text{-}Join(t_1, t_2)$:

$$\begin{aligned} & Eval_{CA}(p)(mk\text{-}Join(t_1, t_2)) && \text{from 174} \\ & = Join_L(Eval_{CA}(p)(t_1), Eval_{CA}(p)(t_2)) && \text{from 30} \\ & = Eval_{CA}(p)(t_1) \cup Eval_{CA}(p)(t_2) && \text{from induction hypothesis} \\ & \subseteq p \cup p = p \end{aligned}$$

$mk\text{-}Meet(t_1, t_2)$: Similar to the proof for Join.

$Attr(\alpha, t)$:

$$\begin{aligned} & Eval_{CA}(p)(Attr(\alpha, t)) && \text{from 174} \\ & = Attribution_{CA}(p)(\alpha)(Eval_{CA}(p)(t)) && \text{173.0} \\ & \subseteq p \end{aligned}$$

B.8 Proof of Term Value Property 175.1

Below is a proof of the term value property 175.1 repeated here:

400.0 $p_1 \subseteq p_2 \;\Rightarrow\; Eval_{CA}(p_1)(t) \subseteq Eval_{CA}(p_2)(t)$

To make the proof, let $p_1 : PO$ and $p_2 : PO$ be two arbitrary partial orders, and assume that the lefthand side of the implication above is true, i.e.

401.0 $p_1 \subseteq p_2$

First 400 must be proved correct for basic terms i.e. terms without sub-terms:

BOTTOM:

$$Eval_{CA}(p_1)(\text{BOTTOM}) = \{\} = Eval_{CA}(p_1)(\text{BOTTOM})$$

TOP:

$$Eval_{CA}(p_1)(\text{TOP}) = p_1 \subseteq p_2 = Eval_{CA}(p)(\text{TOP})$$

Named concept c:

$$\begin{aligned} & Eval_{CA}(p_1)(c) = cValue_L(p_1)(c) \\ & = DownSet(p_1)(\{mk\text{-}CI(\{c\})\}) && \text{from 14} \\ & \subseteq DownSet(p_2)(\{mk\text{-}CI(\{c\})\}) \\ & = Eval_{CA}(p_2)(c) \end{aligned}$$

Next, in the induction steps, when considering compound terms, assume 400 is true for the sub-terms (the induction hypothesis):

$mk\text{-}Join(t_1, t_2)$:

$$\begin{aligned} & Eval_{CA}(p_1)(mk\text{-}Join(t_1, t_2)) && \text{from 174} \\ & = Join_L(Eval_{CA}(p_1)(t_1), Eval_{CA}(p_1)(t_2)) \end{aligned}$$

$= Eval_{CA}(p_1)(t_1) \cup Eval_{CA}(p_1)(t_2)$ from 30

$\subseteq Eval_{CA}(p_2)(t_1) \cup Eval_{CA}(p_2)(t_2)$ from induction hypothesis
$= Join_L(Eval_{CA}(p_2)(t_1), Eval_{CA}(p_2)(t_2))$
$= Eval_{CA}(p_2)(mk\text{-}Join(t_1, t_2))$

$mk\text{-}Meet(t_1, t_2)$: Similar to the proof for Join.

$Attr(\alpha, t)$:

$Eval_{CA}(p_1)(Attr(\alpha, t))$

$= Attribution_{CA}(p_1)(\alpha)(Eval_{CA}(p_1)(t))$ from 174

$\subseteq Attribution_{CA}(p_1)(\alpha)(Eval_{CA}(p_2)(t))$ from induction hypothesis and 173.1

$\subseteq Attribution_{CA}(p_2)(\alpha)(Eval_{CA}(p_2)(t))$ from 173.2
$= Eval_{CA}(p_2)(Attr(\alpha, t))$

B.9 Proof of *CISproj*-property 106

Below is a proof of the property of *CISproj* in 106 repeated here:

402.0 $ISA_S(CISproj(pmax)(cis), cis1) \wedge ISA_S(cis1, cis)$
.1 $\Rightarrow CISproj(pmax)(cis1) = CISproj(pmax)(cis)$

proof: Let $ac1 = CISproj(pmax)(cis1)$ and $ac2 = CISproj(pmax)(cis)$. From the definition of *CISproj* (26) we know that $ac1$ and $ac2$ are both anti-chains in *pmax*. To see that $ac1 = ac2$ we show that they have the same down-set in *pmax*. From the definition of *CISproj* 26.3 we get

403.0 $DownSetC(pmax)(ac1) = DownSetC(pmax)(cis1)$,
.1 $DownSetC(pmax)(ac2) = DownSetC(pmax)(cis)$

Assume the left-hand side of the implication

$ISA_S(CISproj(pmax)(cis), cis1) \wedge ISA_S(cis1, cis)$

which is equivalent to

404.0 $ISA_S(ac2, cis1) \wedge ISA_S(cis1, cis)$

To prove the equality we make two sub-proofs:

$\underline{DownSetC(pmax)(cis1) \subseteq DownSetC(pmax)(cis)}$: So assume

$cip \in DownSetC(pmax)(cis1)$

which, according to the definition of *DownSetC* (23) is equivalent to

405.0 $\exists\, ci1 \in cis1 \cdot ISA_P(cip, ci1)$

Applying the the definition of *DownSetC*(23) to the right conjunct in 404 gives

$\forall\, ci1 \in cis1 \cdot \exists\, ci \in cis \cdot ISA_P(ci1, ci)$

Applying this to 405 gives

$\exists\, ci1 \in cis1 \cdot \exists\, ci \in cis \cdot ISA_P(ci1, ci) \wedge ISA_P(cip, ci1)$

Using the trasitivity of ISA_P gives

$$\exists\, ci \in cis \cdot ISA_P(cip, ci)$$

which is equivalent to

$$cip \in DownSetC(pmax)(cis)$$

$\underline{DownSetC(pmax)(cis) \subseteq DownSetC(pmax)(cis1)}$: We prove the equivalent

$$DownSetC(pmax)(ac2) \subseteq DownSetC(pmax)(cis1)$$

So assume

$$cip \in DownSetC(pmax)(ac2)$$

which, according to the definition of *DownSetC* is equivalent to

406.0 $\exists\, ci2 \in ac2 \cdot ISA_P(cip, ci2)$

Using the definition of ISA_S to the left conjunct in 404 gives:

$$\forall\, ci2 \in ac2 \cdot \exists\, ci1 \in cis1 \cdot ISA_P(ci2, ci1)$$

Applying this to 406 gives

$$\exists\, ci2 \in ac2 \cdot \exists\, ci1 \in cis1 \cdot ISA_P(ci2, ci1) \wedge ISA_P(cip, ci2)$$

Using the trasitivity of ISA_P gives

$$\exists\, ci1 \in cis1 \cdot ISA_P(cip, ci1)$$

from which we get

$$cip \in DownSetC(pmax)(cis1)$$

□

B.10 Proof of *CISproj*-property 107

Below is a proof of the property of *CISproj* in 107 repeated here:

407.0 $CISproj(p)(cis_1 \cup cis_2) = AntiCh(CISproj(p)(cis_1) \cup CISproj(p)(cis_2))$

proof: The equality in 407 is between two antichains in p. Let the left- and right-hand side antichains be $ac1$ and $ac2$. We show they are identical by showing that they have the same down-set in p. For the left-hand side anti-chain $ac1$ we have:

$$\begin{aligned}
&DownSetC(p)(ac1) \\
&= DownSetC(p)(cis_1 \cup cis_2) && \text{from 26.3} \\
&= DownSetC(p)(cis_1) \cup DownSetC(p)(cis_2) && \text{from 12}
\end{aligned}$$

For the right-hand side anti-chain $ac2$ we have:

$$\begin{aligned}
&DownSetC(p)(ac2) \\
&= DownSetC(p)(AntiCh(CISproj(p)(cis_1) \cup CISproj(p)(cis_2))) \\
&= DownSetC(p)(CISproj(p)(cis_1) \cup CISproj(p)(cis_2)) && \text{from 22} \\
&= DownSetC(p)(CISproj(p)(cis_1)) \cup DownSetC(p)(CISproj(p)(cis_2)) && \text{from 12} \\
&= DownSetC(p)(cis_1) \cup DownSetC(p)(cis_2) && \text{from 26.3}
\end{aligned}$$

References

[1] B.A.Davey, H.A.Priestley. *Introduction to Lattices and Order, Second Edition* Cambridge University Press, 2002.

[2] John Dawes. *The VDM-SL Reference Guide.* London: Pitman, 1991

[3] J. Fischer Nilsson. *An Algebraic Logic for Concept Structures.* Information Modelling and Knowledge Bases V, H.Jaakkola et all.(Eds.). IOS Press, 1994

[4] J.Fischer Nilsson. *A Logico-Algebraic Framework for Ontologies (ONTOLOG)*, in Procs. from Int. OntoQuery Workshop on Ontology-based interpretation of NP's, Kolding, January 17-18, 2000

[5] J. Fischer Nilsson, Hele-Mai Haav. *Inducing Queries from Examples as Concept Formation.* Information Modeling and Knowledge Bases X, H.Jaakkola et all.(Eds.). IOS Press, 1999

[6] Frank J. Oles. *An application of lattice theory to knowledge representation*, Theoretical Computer Science 249 (2000) 163-196