ISIMA

Institut Supérieur d'Informatique, de Modélisation et de leurs Applications

1 rue de la Chebarde
Aubières, 63178, France

NATIONAL RESEARCH UNIVERSITY

Higher School of Economics

Kochnovskiy Proyezd, 3
Moscow, 125319, Russia

Master Thesis report:
Data science and 3rd year of computer science engineering

Horn Minimization: an overview of existing algorithms

Author : Simon Vilmin

Academic Supervisor : Sergei A. Obiedkov

Held : June, 26, 2018

Acknowledgement

List of Figures

List of Algorithms

Abstract

Contents

Introduction

Chapter 1

Introduction to implications through closure systems

In this first chapter, we will be involved in presenting our topic of minimization. For this ground to be understandable by as much readers as possible, we will heavily rely on toy examples to illustrate and provide intuition on the various notions we will introduce. To be more precise on the path we are about to follow in this chapter, we are first to expose an informal small example of the task we want to achieve. Then, we shall investigate the history of research on our topic, to act as an exposition of the actual knowledge on the question and to give a context to our study. For the rest of this chapter we will get familiar with mathematical objects called *closure operators* and *closure systems* modelling our problem. As we shall observe, the topic of minimization can be described in several mathematical frameworks. However, even if we describe briefly other objects in next chapters, we will stick to our closure framework in all the report in order to have a leading light among various different terminologies.

1.1 Implications and minimization: first meeting

Let us imagine we are some specialist of flowers and plants in general. As such, we are interested in studying *correlations* between plant characteristics. Some possible traits are: *colourful, bloom, wither, aquatic, seasonal, climbing, scented, flower, perennial* and so forth. Having observed countless plants during our studies, we are able to draw relations among all those *attributes*. For instance, we know that a plant having the attribute *flower* is likely to have traits *scent, bloom, wither* while a plant being *perennial* (i.e: does not need a lot of water to survive, like a cactus) is not likely to be *aquatic*.

Those relations *"if we have some attributes, we get those ones too"* depict correlation between attributes (not cause/consequence!). It is important to stress on the knowledge those relations bring. They just indicate that whenever we have say *flower*, we have also *colourful*. This is very different from saying that *because* some plant is a flower, it will be colourful. We call those correlation relations *implication* and use $flower \longrightarrow colourful$ to denote *"if we have the attribute flower, then we have colourful"*. Now let us give

some implications:

$$(colourful,\ bloom \longrightarrow seasonal),\ (colourful,\ wither \longrightarrow seasonal),\ (bloom \longrightarrow wither)$$

All those implications represent a certain amount of knowledge. While in our example they are not numerous we could imagine having tons of them. Hence we would wonder whether there is a way to reduce the number of implications while keeping all the knowledge they represent. This question is *minimization*. Actually, in our small example we can reduce the number of implications. Take *(colourful, bloom* $\longrightarrow$ *seasonal)*. We can derive this implication relation only with the two other ones. Indeed, because a plant *blooming* is likely to *wither* (3rd implication), we have *(colourful, bloom* $\longrightarrow$ *wither)*, but since we now have *wither* and *colourful* we also have *seasonal* (2nd implication). That is, the implication *(colourful, bloom* $\longrightarrow$ *seasonal)* is useless (or *redundant*) in our context and can be removed. Our set of implications will then be smaller, but pointing out the same relations as before.

To summarize, we have seen that out of a set of *attributes* we can draw several relations called *implications* providing some knowledge. We also realized that sometimes, some implications are not necessary. Consequently, the set of implications we are given can be *minimized* without altering the information it contains. This is the topic we were interested during this master thesis. In the next section, we will trace back the overhaul knowledge on this question.

1.2 Research on implications theories minimization

This section is intended to supply the reader with a general overview of the minimization topic. After a short contextual information, we focus on some relevant results on the question by providing references to algorithms and properties dedicated to our problem. Eventually, we situate our work within this context.

The question of minimization has been discussed and developed through various frameworks, and several computer scientists communities. Notice that in order not to make this synthesis too long, we will stay within the context of minimization and will not trace the field of implication theories in general. For a survey of this domain anyway, the reader should refer to [29]. Also, note that minimality in general terms is not unique. Indeed, one can define several type of minimality among implication systems. For instance, not only we can define minimality with respect to the number of implication within a system (which is our interest) but also with respect to the number of attributes in each implications. The former one is called *canonical* in relational database field, and *hyperarc minimum* within the graph context. Especially in the graph-theoretic and boolean logic settings, one can derive more types of minimality. For general introduction to boolean logic notations, we invite the reader to see [15]. In terms of propositional logic, implications are represented through Horn formulae. Interestingly, the minimization problem we are going to consider is the only one being polynomial time solvable. Other problems are proved to be NP-Complete or NP-Hard. For more discussion on other minimality definitions and their computational complexity, the reader should refer to [13, 6, 7, 5, 29, 11]. In particular for NP-Completeness in the canonical case, one can see [23]. In subsequent explanations, we will refer to minimization with respect to the number of implications.

To the best of our knowledge, the two first fields in which algorithms and properties of minimality arose are Formal Concept Analysis (FCA) (see [21, 20] for an introduction) and Database Theory (DB) (see [24]). Both sides were developed independently in the early 80's. For the first domain, characterization of minimality goes to Duquenne and Guigues [22], in which they describe the so-called *canonical basis* (also called *Duquenne-Guigues basis* after its authors) relying on the notion of pseudo-closed sets. For the database part, study of implications is made by Maier through FD's ([24, 17]). The polynomial time algorithm he gives for minimization heavily relies on a fast subroutine discovered by Beeri and Bernstein in [9], 1979.

From then on, knowledge increased over years and spread out over domains. Another algorithm based on a minimality theorem is given by Shock in 1986 ([25]). Unfortunately, as we shall see and as already discussed by Wild in [28] the algorithm may not be correct in general, even though the underlying theorem is. During the same period, Ausiello and al. brought the problem to graph-theoretic ground, and provided new structure known as *FD-Graph* and algorithm to represent and work on implication systems in [6, 4, 5]. This approach has been seen in graph theory as an extension of the transitive closure in graphs ([1]), but no consideration equivalent to minimization task seems to have been taken beforehand, as far as we know. Still in the 1980 decade, Ganter expressed the canonical basis formalized by Duquenne and Guigues in his paper related to algorithms in FCA, [20] through closure systems, pseudo-closed and quasi-closed sets. Next, Wild ([26, 27, 28]) linked within this set-theoretic framework both the relational databases, formal concept analysis and lattice-theoretic approach. In relating those fields, he describes an algorithm for minimizing a basis, similar to algorithms of Day and, somehow, Shock (resp. [18], [25]). This framework is the one we will use for our study, and can be found in more recent work by Ganter & Obiedkov in [7]. Also, the works of Maier and Duquenne-Guigues have been used in the lattice-theoretic context by Day in [18] to derive an algorithm based on congruence relations. For in-depth knowledge of implication system within lattice terminology, we can see [16] as an introduction and [10] for a survey. Later, Duquenne proposed some variations in Day's work with another algorithm in [19]. More recently, Boròs and al. by working in a boolean logic framework, exhibited a theorem on the size of canonical basis [12, 13]. They also gave a general theoretic approach that algorithm should do one way or another on reduction purpose. Out of these papers, Berczi & al. derived a new minimization procedure based on hypergraphs in [14]. Furthermore, an algorithm for computing the canonical basis starting from any system is given in [7].

Even though the work we are going to cite is not designed to answer this question of minimization, it must also be exposed as the algorithm is intimately related to DG basis and can be used for base reduction. The paper of Angluin and al. in query learning, see [2], provides an algorithm for learning a Horn representation of an unknown initial formula. It has been shown later by Ariàs and Alcazar ([3]) that the output of Angluin algorithm was always the Duquennes-Guigues basis.

Our purpose with this master thesis is to review and implement as much as possible the algorithms we exposed to provide a comparison. This comparison shall act as both theoretical and experimental statement of algorithm efficiency. As we already mentioned we will focus on closure theory framework. The reason for this choice is our starting point. Because we start from the algorithms provided by Wild and

because the closure framework is the one we are the most familiar with, we focus on clearly explain this terminology with examples. However, once we will be comfortable with those definitions, we will relate other frameworks to our main approach in the next chapter, to explain and draw parallels with other algorithms. In the next section we will focus on theoretical definitions we shall need to understand the algorithms we have implemented.

1.3 Implications and minimization: theoretic approach

Here we will dive into mathematical representation of the task we gave in the first section of this chapter. For the recall, our aim here is to get familiar with the representation being closest from closure systems. Most of the notions initially come from [22, 20, 27, 21] but the reader can also find more than sufficient explanations in [7, 29]. Readers with knowledge in relational databases will recognize most of functional dependency notations. The reason is close vicinity between implications and functional dependencies. Talking about our needs, we can consider them as equivalent notations. Actually, the real-life application our set up will be the closest from is FCA ([21]) as we shall see in the last chapter.

1.3.1 Implications and closure systems

The easiest object to project onto mathematical definitions is our attribute set. For all the report, we fix Σ to be a set of *attributes*. Usually, we will denote attributes by small letters: *a, b, c, ...* and subsets of Σ (groups of attributes) will be denoted by capital letters: *A, B, C, ...* We assume the reader to have few background in elementary set-theoretic notations.

Definition 1 (*Implication, implication system*)**.** *An implication over Σ is a pair (A, B) with $A, B \subseteq \Sigma$. It is usually denoted by $A \longrightarrow B$. A set $\mathcal{L}$ of implications is called an implication system, implication theory or implication(al) base(is).*

Note that given as is, this definition seems to lose the semantic relation we depicted earlier. But we should keep in mind that in our set up, we will be given implications more than an attribute set. Hence, implications will make sense on their own, independently from the attribute set they are drawn from. Quickly, remark that implications in logical terms are expressed as *Horn formulae* giving another of its names to implication theories. Also, in $A \longrightarrow B$, A is said to be the *premise* (or *body*) and B the *conclusion* (*head*).

Definition 2 (*Model*)**.** *Let $\mathcal{L}$ be an implication system over Σ, and $M \subseteq \Sigma$. Then:*

(i) M is a model of an implication $A \longrightarrow B$, written $M \models A \longrightarrow B$, if $B \subseteq M$ or $A \nsubseteq M$,

(ii) M is a model of $\mathcal{L}$ if $M \models A \longrightarrow B$ for all $A \longrightarrow B \in \mathcal{L}$.

The notion of model may seem disarming at first sight. But M being a model of $A \longrightarrow B$ simply means that, if A is included in M, then for the implication $A \longrightarrow B$ to hold in M, we must have B in M too. This still suits the intuitive notion of premise/conclusion. Placed in the context of M, $A \longrightarrow B$ says *"whenever we have A, we must also have B"*. Reader with some background in mathematical logic should be familiar with the notation $\models$, denoting semantic entailment, as opposed to $\vdash$ for syntactic deduction (see [15]). By a fortunate twist of fate, semantic entailment is our next step:

Definition 3 (*Semantic entailment*)**.** *We say that an implication $A \longrightarrow B$ semantically follows from $\mathcal{L}$, denoted $\mathcal{L} \models A \longrightarrow B$, if all models M of $\mathcal{L}$ are models of $A \longrightarrow B$.*

Because next definitions are going to be on a slightly different structure, even though closely related to implication systems of course, let us rest for a while and illustrate our definitions with an example.

Example Consider again our plant properties. Let $\Sigma = \{$*colourful, bloom, wither, seasonal, aquatic, perennial, flower, scented*$\}$. An implication could be *flower* $\longrightarrow$ *scented*, or even *(bloom, aquatic)* $\longrightarrow$ *colourful* if we get rid off semantic interpretations. An implication basis $\mathcal{L}$ is for instance:

(colourful, bloom $\longrightarrow$ *seasonal), (colourful, wither* $\longrightarrow$ *seasonal), (bloom* $\longrightarrow$ *wither)*

and $M =$*(colourful, bloom, seasonal)* is a model of *colourful, bloom* $\longrightarrow$ *seasonal* because both the head and the body of the implication belong to M. Also, M is not a model of $\mathcal{L}$ because it is not a model of *bloom* $\longrightarrow$ *wither*. A model of $\mathcal{L}$ could be *(bloom, wither)* or even the empty set $\emptyset$.

Next definitions are about closure operators, and closure systems. We need to ground ourselves in those definitions before returning to implications. 2^Σ is the set of all subsets of Σ, also named the *power set* of Σ.

Definition 4 (*Closure operator*)**.** *Let Σ be a set and $\phi : 2^\Sigma \longrightarrow 2^\Sigma$ an application on the power set of Σ. ϕ is a closure operator if $\forall X, Y \subseteq \Sigma$:*

(i) $X \subseteq \phi(X)$ *(extensive),*

(ii) $X \subseteq Y \longrightarrow \phi(X) \subseteq \phi(Y)$ *(monotone),*

(iii) $\phi(X) = \phi(\phi(X))$ *(idempotent).*

$X \subseteq \Sigma$ is called closed if $X = \phi(X)$.

Definition 5 (*Closure system*)**.** *Let Σ be a set, and $\Sigma^\phi \subseteq 2^\Sigma$. Σ^ϕ is called a closure system if:*

(i) $\Sigma \in \Sigma^\phi$,

(ii) if $\mathcal{S} \subseteq \Sigma^\phi$, *then* $\bigcap \mathcal{S} \in \Sigma^\phi$ *(closed under intersection).*

In the second definition, it is worth stressing on the fact that Σ^ϕ is a set of sets. Also, the notation Σ^ϕ may seem surprising, but it has been chosen purposefully. Indeed, to each closure system Σ^ϕ over Σ, we can associate a closure operator ϕ and vice-versa:

- from ϕ to Σ^ϕ: compute all closed sets of ϕ to obtain Σ^ϕ,
- from Σ^ϕ to ϕ: define $\phi(X)$ as the smallest element of Σ^ϕ (inclusion-wise) containing X. Observe that such a set always exists in Σ^ϕ because $\Sigma \in \Sigma^\phi$.

In any case, this notation used for clear exposition of the link between closure systems and closure operators will be adapted to our context of implication systems as we shall see later on. Notice that one can encounter another object, *closure space*, being a pair (Σ, ϕ) where Σ is a set and ϕ a closure operator over Σ. We are likely to find this notation notably in [26, 27] where a general theory of closure spaces is addressed.

Example Let us imagine we have four people: *Jezabel, Neige, Seraphin* and *Narcisse*. Let us assume they all know each other and then define a relation *"like"* between them. For instance, say *Séraphin likes Jezabel*. this relation is a *binary relation*: it relates pairs of elements. We can represent this relation by a graph where nodes are people and edges represent relations:

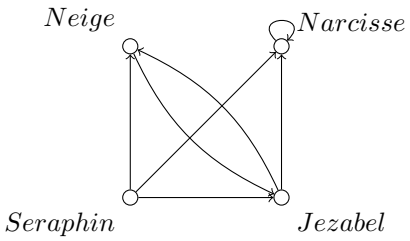


Figure 1.1: Graph of "like" relation

The arrow from *Seraphin* to *Jezabel* stands for *"Seraphin likes Jezabel"* and the arrow from *Narcisse* to itself means equivalently *"Narcisse likes Narcisse"*. With this clear, let us introduce an operation of gathering people. Starting from any group A of persons presented here, let's add to A every person liked by at least one element of A, until we can no more add people. For instance:

- if we start from *Neige*, because *Neige* likes *Jezabel* and *Jezabel* likes *Narcisse* we will add both of them to the group of *Neige*,
- because *Narcisse* only likes himself, we have no people to add in his group.

Now observe that this operation of gathering people is in fact a closure operator:

(i) it is *extensive*: starting from any group of people, we can only add new ones, hence either the group does not change (e.g: *Narcisse*) or it grows,

(ii) it is *monotone*: if we start from a group A containing a group B, it is clear that we will at least gather in A all the people we would add with B,

(iii) *idempotency*: once we added all the people we had to reach, then trying to find new people is useless by definition. Hence the group will remain the same if we apply our operation once more.

We are going to get back to our main implication purpose to illustrate the notion of closure in our context. It turns out that given a basis $\mathcal{L}$ over some set Σ, the set of models of $\mathcal{L}$, $\Sigma^{\mathcal{L}}$, is a closure system. Moreover, the operator $\mathcal{L} : 2^{\Sigma} \longrightarrow 2^{\Sigma}$ associating to a subset X of Σ the smallest model (inclusion wise) containing X is a closure operator. Furthermore, the closure system it defines is $\Sigma^{\mathcal{L}}$. An interesting point is the mathematical computation of $\mathcal{L}(X)$ given $\mathcal{L}$ as a set of implications. We rely on [26, 7] to this end. Let us define a temporary operation $\circ : 2^{\Sigma} \longrightarrow 2^{\Sigma}$ as follows:

$$X^{\circ} = X \cup \bigcup \{B \mid A \longrightarrow B \in \mathcal{L},\ A \subseteq X\}$$

Applying this operator up to stability provides $\mathcal{L}(X)$. In other words $\mathcal{L}(X) = X^{\circ\circ\cdots}$. It is clear that we have a finite amount of iterations since X cannot grow more than Σ. Readers with background in logic (see [13]) or graph theory ([14]) might see this operation as the marking or forward chaining procedure.

Example Let's stick to our vegetable example, but reducing Σ to {*bloom, flower, colourful* } (abbreviated *b, f, c*) for the sake of simplicity. Furthermore, let $\mathcal{L} = \{$*((colourful, bloom)* $\longrightarrow$ *flower), (flower* $\longrightarrow$ *bloom)*$\}$, abbreviated then $cb \longrightarrow f$, $f \longrightarrow b$. For instance, because $f \longrightarrow b \in \mathcal{L}$, the smallest model of $\mathcal{L}$ containing f is bf, and bf is closed. More precisely, the set of closed sets is the following:

$$\Sigma^{\mathcal{L}} = \{\emptyset,\ b,\ c,\ bf,\ bcf\}$$

Having presented the main definitions we shall need, we are to investigate practical computation of closures and more elaborated structures like the canonical basis (or Duquenne-Guigues basis) in the next section.

1.3.2 Canonical Basis, Closure Algorithm

Before giving the definition of canonical basis, we should consider special kind of sets given $\mathcal{L}$ over Σ. Also, we will need to expose particular implications. First of all, let us introduce a property through a proposition we will assume (we redirect the reader to [7] for another proof). When not introduced, we consider a system $\mathcal{L}$ of implications, over some attribute set Σ.

Proposition 1. *Let $A \longrightarrow B$ be an implication. $\mathcal{L} \models A \longrightarrow B$ if and only if $B \subseteq \mathcal{L}(A)$.*

Proof. $\mathcal{L} \models A \longrightarrow B \implies B \subseteq \mathcal{L}(A)$. Every model of $\mathcal{L}$ models $A \longrightarrow B$, hence for each closed set X of $\mathcal{L}$, either $A \subseteq X$ and $B \subseteq X$, or $A \nsubseteq X$. Consider all closed X for which $A \subseteq X$. By definition $\mathcal{L}(A) = \bigcap\{X \in \Sigma^{\mathcal{L}},\ A \subseteq X\}$ and $B \subseteq \mathcal{L}(A)$.

$B \subseteq \mathcal{L}(A) \implies \mathcal{L} \models A \longrightarrow B$. By contraposition suppose $\mathcal{L} \not\models A \longrightarrow B$. Then there must exist at least one model X of $\mathcal{L}$ such that $A \subseteq X$ and $B \nsubseteq X$. Because $A \subseteq X$, $\mathcal{L}(A) \subseteq X$ hence $B \nsubseteq \mathcal{L}(A)$.

□

Definition 6 (*Redundancy*)**.** *An implication $A \longrightarrow B$ of $\mathcal{L}$ is redundant if $\mathcal{L} - \{A \longrightarrow B\} \models A \longrightarrow B$. If $\mathcal{L}$ contains no redundant implications, it is non-redundant.*

Our definition of redundancy models the notion of *"useless"* we were talking about in our toy example: if an implication is true in some $\mathcal{L}$ even if we remove it, it brings no knowledge. In practice, redundancy can be checked as follows: put $\mathcal{L}^-$ as $\mathcal{L}$ without $A \longrightarrow B$ and compute $\mathcal{L}^-(A)$. If $\mathcal{L}^-(A) = \mathcal{L}(A)$ or equivalently, if $B \subseteq \mathcal{L}^-(A)$, then $A \longrightarrow B$ is redundant. Moreover, it is worth commenting that in FCA or DB fields (see [21, 24]), implications (or FD's) are deduced from data presented as contexts or relation schemes. Hence, we usually introduce notions of soundness and completeness ensuring that implications we are working on are meaningful with respect to the knowledge we are dealing with. More precisely, *soundness* ensures that $\mathcal{L}$ does not contain any implication not holding in the dataset. *Completeness* says that all true implications

in the data context are true in $\mathcal{L}$. Because we work directly on implications, $\mathcal{L}$ is by definition sound and complete with respect to the models it defines. Next, we set up minimality.

Definition 7 (Minimality)**.** *$\mathcal{L}$ is minimal if removing one of its implication alters $\Sigma^{\mathcal{L}}$.*

Example We consider our canonical plant example. Take

$$\mathcal{L} = \{((colourful,\ bloom) \longrightarrow seasonal),\ ((colourful,\ wither) \longrightarrow seasonal),\ (bloom \longrightarrow wither)\ \}$$

as we explained in first section, the first implication can be removed. In particular, it is redundant. Hence $\mathcal{L}$ is not minimal. If we get rid of $(colourful,\ bloom) \longrightarrow seasonal)$, $\mathcal{L}$ will be minimal.

Interestingly, depending on the implications we get, non-redundancy is not a sufficient criterion for minimality as we shall see in Maier algorithm. As an example for now, consider $\Sigma = \{a,\ b,\ c,\ d,\ e,\ f\}$ and $\mathcal{L} = \{ab \longrightarrow cde,\ c \longrightarrow a,\ d \longrightarrow b,\ cd \longrightarrow f\}$. $\mathcal{L}$ is not redundant, but is not minimal either. In fact, $\mathcal{L}_m = \{ab \longrightarrow cdef,\ c \longrightarrow a,\ d \longrightarrow b\}$ is equivalent to $\mathcal{L}$ but with one implication less.

For now, we defined what are implication theories, redundancy and minimality. One could expect our next step to be the exposition of some minimal basis. Unfortunately, we need to make a detour to visit some set and order definitions before getting back to our main purpose. Those notions not only deserve to explain minimal basis but also to settle some landmarks for further discussions in the next chapter.

Recall that in our example of closure operator we briefly approached binary relations. To be more formal, let E, F be two sets. A *binary relation* $\mathfrak{R}$ is a set of pairs (e, f) (sometimes denoted $e\mathfrak{R}f$) with $e \in E$, $f \in F$, or equivalently $\mathfrak{R} \subseteq E \times F$. We will assume $\mathfrak{R} \subseteq E^2$. Actually, $\mathfrak{R}$ can present some properties:

(i) *reflexivity*: $\forall x \in E, x\mathfrak{R}x$,

(ii) *irreflexivity*: $\forall x \in E, \neg(x\mathfrak{R}x)$,

(iii) *symmetry*: $\forall x, y \in E, x\mathfrak{R}y \longrightarrow y\mathfrak{R}x$

(iv) *antisymmetry*: $\forall x, y \in E, x\mathfrak{R}y \wedge y\mathfrak{R}x \longrightarrow x = y$,

(v) *asymmetry*: $\forall x, y \in E, x\mathfrak{R}y \longrightarrow \neg(y\mathfrak{R}x)$,

(vi) *transitivity*: $\forall x, y, z \in E, x\mathfrak{R}y \wedge y\mathfrak{R}z \longrightarrow x\mathfrak{R}z$

All possible properties are not given here, see [15] for more. With those properties anyway, we can define several types of relations:

Definition 8. *Let E be a set and $\mathfrak{R}$ a binary relation on E:*

(i) $\mathfrak{R}$ is an equivalence relation (denoted by $=$) if it is reflexive, transitive and symmetric,

(ii) $\mathfrak{R}$ is a (partial) order ($\leq$) if reflexive, transitive and antisymmetric,

(iii) $\mathfrak{R}$ is a strict order ($<$) if irreflexive, transitive and asymmetric.

Example Time has come for some illustrations. First, let us imagine we are looking at some tree in a meadow. Because the season is spring, this tree has branches and leaves. We are interested in the set of all leaves, and we would like to relate them by the branch they are one. Hence define $\mathfrak{R}$ as *"is on the same branch as"*, being a binary relation. It turns out that $\mathfrak{R}$ is an equivalence relation:

- *reflexivity*: every leaf is on the same branch as itself;
- *transitivity*: if a leaf l_1 is on the same branch as a leaf l_2, and l_2 is on the same branch as l_3, then it is clear that l_1 is on the same branch as l_3;
- *symmetry*: l_1 being on the same branch as l_2 clearly implies that l_2 is on the same branch as l_1.

For partial and strict ordering, we will go back to more mathematical examples, in order to slowly go back to our main purpose. Consider the set $\mathbb{N}$ $(= \mathbb{N}_0)$ of positive integers, including 0. The natural relation $\leq$ is an order, and the pair $(\mathbb{N}, \leq)$ is an ordered set. In particular it is a *totally ordered set* or *chain* because every pair of integers can be compared. $<$ is a strict total ordering on $\mathbb{N}$. Another example,let $\Sigma = \{a, b, c\}$ be a set of attributes and consider $\subseteq$ as a binary relation on 2^Σ. Again, $(2^\Sigma, \subseteq)$ is a *partially ordered set* (or *poset* under abbreviation):

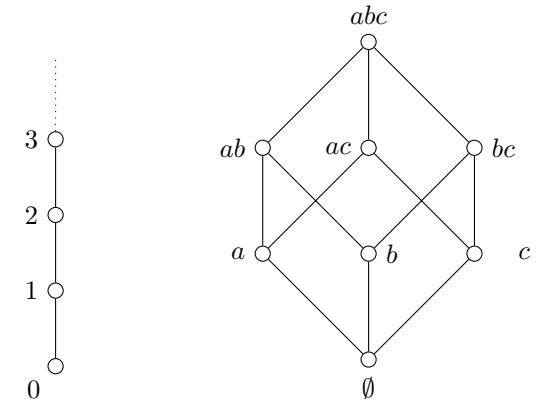
- every subset X of Σ is included in itself, for instance $\{a, b\}$ is a subset or equal to $\{a, b\}$, whence *reflexivity*,
- if $X \subseteq Y$ and $Y \subseteq X$ then necessarily, $X = Y$ (*antisymmetry*),
- if $X \subseteq Y \subseteq Z$, then clearly $X \subseteq Z$ (*transitivity*)

There is a convenient way to represent posets. At least when they are not to heavy. It is sometimes called *Hasse diagram* (see [16]) and relies on the *cover* relation of a partially ordered set. Take any poset $(P, \leq)$ and define the cover relation as $x \prec y$ if $x < y$ and $x \leq z < y \longrightarrow x = z$. In other word, $x \prec y$ says *"there is no element between x and y"*. The example of $\mathbb{N}$ is appealing. For instance, $4 \prec 5$ because there is no integer between 4 and 5, but $4 \not\prec 7$ since we can find 5 and 6 as intermediary elements. Now the Hasse diagram of $(P, \leq)$ is a graph drawn as follows:

1. there is a point for each $x \in P$,
2. if $x \leq y$, then y is placed above x,
3. we draw an arc between x and y if and only if $x \prec y$ in P.

As examples, one can observe the diagrams of $(\mathbb{N}, \leq)$ and $(2^\Sigma, \subseteq)$ described previously in figure 1.2. On the right-hand side we wrote a subset of Σ by a concatenation of its element for readability purpose.

Now equipped with orders and equivalence relation, we can go a bit further in the study of implications and sets. For instance, as exposed in the previous example, we can consider our attribute set Σ (more precisely its power set) equipped with $\subseteq$ as an ordering. Furthermore, recall that $\mathcal{L}$ is a set of implications and hence provide a closure operators. Because every subset of Σ has only one closure in $\mathcal{L}$, we can define an equivalence relation $\equiv_\mathcal{L}$ on 2^Σ as follows:



(a) $(\mathbb{N}, \leq)$ (b) $(2^{\Sigma}, \subseteq)$

Figure 1.2: Hasse diagrams of two ordered sets

$$\forall X, Y \subseteq \Sigma, X \equiv_{\mathcal{L}} Y \text{ if and only if } \mathcal{L}(X) = \mathcal{L}(Y)$$

Let us go one step beyond. With $\equiv_{\mathcal{L}}$, we can set up *equivalence classes* on $(2^{\Sigma}, \equiv_{\mathcal{L}})$. An equivalence class can be defined with respect to an element X as follows:

$$[X]_{\mathcal{L}} = \{Y \subseteq \Sigma \mid X \equiv_{\mathcal{L}} Y\}$$

Those equivalence classes are a partition of 2^{Σ} with respect to the closed sets of $\mathcal{L}$: every model of $\Sigma^{\mathcal{L}}$ defines an equivalence class.

Example Because all this discussion on equivalence class may have been a bit troubling, let us rest for a while before eventually reaching minimal basis definitions. Remind our plant example:

- $\Sigma = \{$*bloom, flower, colourful*$\}$ (abbreviated $\{b, f, c\}$),
- $\mathcal{L} = \{$*((colourful, bloom)* $\longrightarrow$ *flower), (flower* $\longrightarrow$ *bloom)*$\}$ (abbreviated $cb \longrightarrow f$, $f \longrightarrow b$)
- the models (closed sets of $\mathcal{L}$) are: $\Sigma^{\mathcal{L}} = \{\emptyset,\ b,\ c,\ bf,\ bcf\}$

In details, because of the implication $f \longrightarrow b$, we can observe that f and bf belong to the equivalence class defined by bf. The same goes for bc, cf and bcf, describing the class given associated to bcf. In order to make it clear, we give a graphical representation of those classes using orders in figure 1.3.

On the left side of the picture, we drew $(2^{\Sigma}, \subseteq)$. On the right-hand side: $(\Sigma^{\mathcal{L}}, \subseteq)$. Clusters on the left diagram are the equivalence classes, associated (dotted arrows) to their closed representative. If a cluster contains only one element, this element is closed. This drawing shows the relation between a closure operator and its associated system, in particular in implication basis context, where the closure describes models. Finally, one can graphically note that the set of models is indeed closed under intersection. While

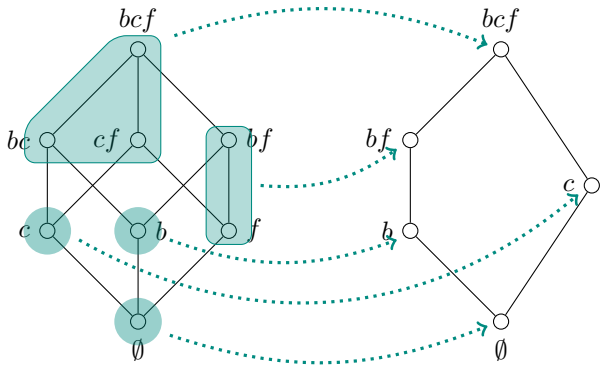


Figure 1.3: Closure operation and equivalence classes

this representation is graphically appealing, it is clearly not tractable for larger attribute set: we have to draw a diagram with an exponential number of elements (one for all $X \in 2^{\Sigma}$). Thus, all Hasse diagrams we are going to draw only aim at providing some intuition of the various notions and not as an efficient representation.

With this detour in order theory made, even though closely related to our topic as we have seen, we can now go back to our main goal: the canonical basis. It relies on some particular sets in the closure systems.

Definition 9 (*Pseudo-closed set*)**.** *Given $\mathcal{L}$ over Σ, we say that $P \subseteq \Sigma$ is pseudo-closed if:*

(i) $P \neq \mathcal{L}(P)$,

(ii) $Q \subset P$ *and* Q *pseudo-closed, implies* $\mathcal{L}(Q) \subseteq P$.

The idea of pseudo-closed sets goes back to Guigues and Duquenne in [22], but the name comes from Ganter in [20]. We can also find explanations in following research [20, 18] and in [7]. It turns out that we can explain pseudo-closure by using so called *quasi-closed sets* (see [26, 20, 22]).

Definition 10 (*Quasi-closed set*)**.** *a set $Q \subseteq \Sigma$ is quasi-closed with respect to $\mathcal{L}$ if:*

(i) $Q \neq \mathcal{L}(Q)$,

(ii) $\forall A \subseteq Q$, $\mathcal{L}(A) \subseteq Q$ *or* $\mathcal{L}(A) = \mathcal{L}(Q)$.

The recursive definition of pseudo-closed sets may seem complicated, and it is somehow since the problem of determining whether a set is pseudo-closed or not has been proven to be NP-Hard (see [7]). Fortunately, quasi-closed sets and equivalence classes give another definition to pseudo-closeness: a set is pseudo-closed if it is quasi-closed and minimal (inclusion-wise) in its equivalence class. Let us illustrate those notions with some diagrams.

Example Let us consider the following case:

- $\Sigma = \{a,\ b,\ c\}$,

- $\mathcal{L} = \{a \longrightarrow b,\ b \longrightarrow a,\ c \longrightarrow ab\}$.

As in 1.3, we will represent the power set of Σ and equivalence classes of $\mathcal{L}$. Two subsets of Σ are in the same class if they have the same closure in $\mathcal{L}$. First, one can observe the effective class representation in figure 1.4. Models of $\mathcal{L}$ are indeed $\emptyset$, ab and abc. For instance $\mathcal{L}(ac) = abc$.

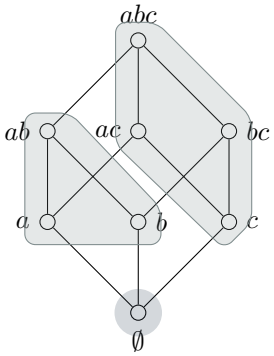


Figure 1.4: Equivalence classes representation of $\mathcal{L} = \{\ a \longleftrightarrow b,\ c \longrightarrow ab\ \}$

Next, we can observe figure 1.5 in which we somehow represented the definition of quasi-closure. We still represent equivalence classes. On the left-hand side figure, we consider the subset c. For c to be quasi-closed, we must look at all of its subsets, and see whether the closure of each subset is either smaller than c or in the same equivalence class as c. The dashed line shows which elements of the diagram we have to consider. In fact it represents what we call in lattice and order theories the *ideal* or *down-set* generated by c:

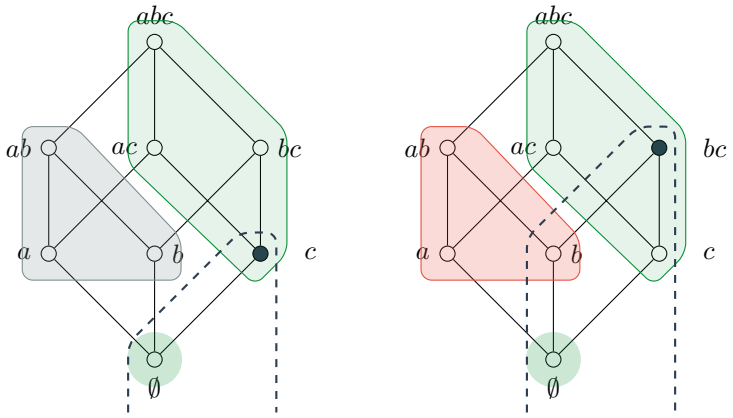
$$\downarrow c = \{X \subseteq \Sigma \,|\, X \subseteq c\}$$

It appears that the only distinct subset of c is $\emptyset$, which is closed. c itself is also not closed. Hence c is indeed quasi-closed.

On the right-hand side we consider the subset bc. As shown in the picture (under the dashed line), there are 3 elements to consider: $\emptyset$, c, b. For the same reason as for c, $\emptyset$ is not a problem. the closure of c is not included in bc, but equals the closure of bc. Hence c is not an issue either. However, b is included in bc, but its closure is ab, neither subset of bc nor equal to abc. Therefore, bc is not quasi-closed. Actually, one could informally say that a set Q is quasi-closed if the following is true:

$$(\forall P \in 2^{\Sigma})\,[(P \in \downarrow Q) \longrightarrow (\mathcal{L}(P) \in \downarrow Q \cup \{\mathcal{L}(Q)\})]$$

where $\downarrow Q$ is the ideal generated by Q (see dashed line in figure 1.5).



(a) Quasi-closed set: c (b) Non quasi-closed set: bc

Figure 1.5: Example of quasi-closeness in small implication system

Example Now let us take

- $\Sigma = \{a,\ b,\ c\}$,
- $\mathcal{L} = \{c \longrightarrow ab,\ b \longrightarrow ab\}$.

Again, we will use equivalence classes and Hasse diagram to represent the closure system of $\mathcal{L}$, see figure 1.6. In this representation we coloured all quasi-closed sets at least in grey. Red (or lighter) nodes are precisely pseudo-closed sets: they are the minimal quasi-closed sets among the equivalence class defined by abc.

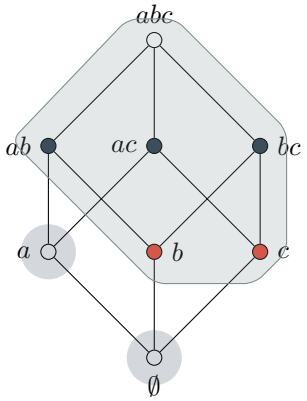


Figure 1.6: Pseudo-closed sets of $\mathcal{L} = \{b \longrightarrow ac,\ c \longrightarrow ab\ \}$

Note that in particular, minimal premises of $\mathcal{L}$ inclusion wise are pseudo-closed. Furthermore, we should be aware that an equivalence class may not contain pseudo-closed set, or more generally, quasi-closed sets. As such, we cannot consider that minimal elements of equivalence classes are quasi-closed. Take for example

$\mathcal{L} = \{\emptyset \longrightarrow a,\ b \longrightarrow a\ \}$. b and ab define a class, but b is not even quasi-closed. With these notions, we can move on and define the canonical basis.

Definition 11 (*Duquenne-Guigues basis*)**.** *The basis $\mathcal{L}$ defined by*

$$\mathcal{L} = \{P \longrightarrow \mathcal{L}(P) \mid P \text{ is pseudo-closed } \}$$

is called the Duquenne-Guigues or canonical basis. It is minimal.

This definition does not say that the canonical basis is the only one being minimal. Actually, it says that every minimal basis should have the same number of implications than this one. We can find a deeper argument in [7] on links between any minimal basis and the canonical one.

So far we discussed several notions: implications, pseudo-closed set, quasi-closed set, canonical basis and so forth. Most of them relies heavily on computing the closure of sets with respect to $\mathcal{L}$. Hence, to have practical efficiency, we must be able to compute closures as fast as possible. Fortunately, several algorithms can be found. Among them, there is a naïve procedure based on the operation $\circ$ we described earlier. Furthermore the algorithm by Beeri and Bernstein in [9] called LinClosure addresses this question. LinClosure as previously mentioned has been widely used, notably in [24, 17, 7, 25, 18]. Before describing those procedures, let us introduce our complexity notations:

- $|\,\Sigma\,|$ will denote the size of the attribute set Σ,
- $|\,\mathcal{B}\,|$ will be the number of implications in $\mathcal{L}$ ($\mathcal{B}$ stands for body),
- $|\,\mathcal{L}\,|$ is the number of symbols used to represent $\mathcal{L}$.

We consider $|\,\mathcal{L}\,|$ to be in reduced form for complexity results. By "reduced" we mean that we do not have distinct implications with same bodies. Indeed, if say, $a \longrightarrow b$ and $a \longrightarrow c$ holds in some $\Sigma^{\mathcal{L}}$ then we can replace those two implications by $a \longrightarrow bc$. Moreover, we shall not explain in details O notation for complexity since we do not need in-depth knowledge within this field. For us, it is enough to say that O is the asymptotically worst case complexity (in time or space). For instance, in the worst case, $|\,\mathcal{L}\,| = |\,\mathcal{B}\,| \times |\,\Sigma\,|$, thus $|\,\mathcal{L}\,| = O(|\,\mathcal{B}\,| \times |\,\Sigma\,|)$. Closure and LinClosure are algorithms 1, 2 (resp.).

As we already mentioned, the algorithm Closure relies on the $\circ$ operation. The principle is to re-roll over the set of implications $\mathcal{L}$ to see whether there exists an implication $A \longrightarrow B$ in $\mathcal{L}$ such that $\mathcal{L}(X) \not\models A \longrightarrow B$ up to stability. Asymptotically, we will need $O(|\,\mathcal{B}\,|^2 \times |\,\Sigma\,|)$ if we remove only one implication per loop. the $|\,\Sigma\,|$ cost comes from the set union.

LinClosure has $O(|\,\mathcal{L}\,|)$ time complexity. The main idea is to use counters. Starting from X, if we reach for a given $A \longrightarrow B$ as many elements as $|A|$, then $A \subseteq \mathcal{L}(X)$ and we must also add B. Because the closure in itself is not the main point of our topic, we will not study LinClosure in depth. Furthermore, there exists other linear time algorithm for computing closure. For more complete theoretical and practical comparisons of closure algorithms, we redirect the reader to [8]. In this paper, LinClosure is shown maybe not to be the most efficient algorithm in practice when used in other algorithms, especially when

Algorithm 1: CLOSURE

Input: A base $\mathcal{L}$, $X \subseteq \Sigma$

Output: The closure $\mathcal{L}(X)$ of X under $\mathcal{L}$

```
closed := ⊥ ;
L(X) := X ;
while ¬closed do
    closed := ⊤ ;
    foreach A ⟶ B ∈ L do
        if A ⊆ L(X) then
            L(X) := L(X) ∪ B ;
            L := L − {A ⟶ B} ;
            closed := ⊥ ;
return L(X);
```

compared with CLOSURE. Anyway, because of its theoretical complexity and use in all algorithms we will review, we will still consider LINCLOSURE, notably because we can separate the initialization step from the computation one in some cases on optimization purpose.

In this last section, we got a step further in building ground for understanding the implication theory structure. We gave definitions of minimality and visual examples of particular sets called pseudo-closed. With the support of those sets, we defined the canonical basis known to be minimal. Finally, algorithms for efficiently computing closures have been presented.

Conclusion In this chapter we first gave a soft introduction to our task with a somehow "physical" example. Then we described briefly advances starting from the first properties found independently in Concept Analysis and Relational Databases fields. We have seen that the question of Horn minimization has been studied in various fields such as graphs, closure spaces, logic (where the name Horn comes from), functional dependencies, lattices. Then, we placed our study within this context. The aim of this study has been exposed as providing a review of some algorithms we talked about, and comparing them under implementation. The last part of the chapter was dedicated to a more formal and theoretical ground necessary for a good understanding of subsequent parts. In the next chapter, we will theoretically discuss in details several algorithms.

Algorithm 2: LinClosure

Input: A base $\mathcal{L}$, $X \subseteq \Sigma$

Output: The closure $\mathcal{L}(X)$ of X under $\mathcal{L}$

```
foreach A ⟶ B ∈ ℒ do
    count[A ⟶ B] := |A| ;
    if |A| = 0 then
        X := X ∪ B ;
    foreach a ∈ A do
        list[a] = list[a] ∪ {A ⟶ B} ;
update := X ;
while update ≠ ∅ do
    choose m ∈ update ;
    update := update − {m} ;
    foreach A ⟶ B ∈ list[m] do
        count[A ⟶ B] := count[A ⟶ B] − 1 ;
        if count[A ⟶ B] = 0 then
            add := B − X ;
            X := X ∪ add ;
            update := update ∪ add ;
return X ;
```

Chapter 2

Review of existing algorithms

In the first chapter we introduced the aim of our study, our working procedure and the main tools for us to understand the next explanations. In this part, we will focus on reviewing the existing algorithms for the Horn minimization task.

2.1 Minimization by reduction

2.1.1 Obiedkov and Ganter algorithm: use of saturation

Compute the canonical (or Duquenne-Guigues) basis $\mathcal{L}_c$ given $\mathcal{L}$. [**?**, 7]. This algorithm performs so called operations of *left-saturation*, *right-saturation* and *redundancy elimination*.

Algorithm 3: Minimal Cover

Input: $\mathcal{L}$ an implication basis

Output: $\mathcal{L}_c$ the canonical basis derived from $\mathcal{L}$

foreach $A \longrightarrow B$ *in* $\mathcal{L}$ **do**
    $\mathcal{L} = \mathcal{L} - \{A \longrightarrow B\}$;
    $B = \mathcal{L}(A \cup B)$;
    $\mathcal{L} = \mathcal{L} \cup \{A \longrightarrow B\}$;

foreach $A \longrightarrow B$ **do**
    $\mathcal{L} = \mathcal{L} - \{A \longrightarrow B\}$;
    $A = \mathcal{L}(A)$;
    if $A \neq B$ **then**
        $\mathcal{L} = \mathcal{L} \cup \{A \longrightarrow B\}$;

The first loop maximizes each heads, that is $A \longrightarrow B$ becomes $A \longrightarrow \mathcal{L}(A)$. We summarize the knowledge. The second loop maximizes left side of implications with $\mathcal{L}^{-}(A)$. The second step kills redundancy.

Notes: few tips to see that the resulting basis is equivalent to the input. By the end of the first loop, since we replaced $A \longrightarrow B$ by $A \longrightarrow \mathcal{L}(A)$, and by definition of $A \longrightarrow B$, $A \longrightarrow B$ still holds for all B ($B \subseteq \mathcal{L}(A)$). For the second loop, observe that we omit only redundant implications from the basis, because if $\mathcal{L}^-(A) = \mathcal{L}(A)$ it means that all the implications $A \longrightarrow B$ true in $\mathcal{L}$ are true in $\mathcal{L}^-$.

Complexity the two loops have the same complexity: $O(|\mathcal{B}(\mathcal{L})| \times |\mathcal{L}|)$ because we run LinClosure for all implications of $\mathcal{L}$. Thus, the complexity of the algorithm is $O(|\mathcal{B}(\mathcal{L})| \times |\mathcal{L}|)$.

2.1.2 Maier Algorithm: using equivalence classes

This part relies mainly on [17], [24] in which an algorithm for minimizing set of functional dependencies (FD). Because FDs behave as implications in implicational basis, we will adopt the latter terminology.

Theoretical set up and definitions

We are going to work in the FD framework, that is first notations we defined. Before diving into the algorithm, we may need some definitions. Indeed, the minimization procedure in this case relies on a criterion for minimality using specific objects.

Definition 12 (Equivalence classes)**.** *Let $X \subseteq \Sigma$. The set*

$$E_{\mathcal{L}}(X) = \{A \longrightarrow B | A \longrightarrow B \in \mathcal{L}, A \equiv X\}$$

is the equivalence class of X under $\mathcal{L}$. In fact, it is the set of implications of $\mathcal{L}$ with body equivalent to X. The set of all non-empty such classes is denoted $\bar{E}_{\mathcal{L}}$.

One should note in order to get $\bar{E}_{\mathcal{L}}$, there is no need to check all the subsets of Σ. Because $\bar{E}_{\mathcal{L}}$ defines a partition of $\mathcal{L}$ based on implication bodies, it is sufficient to go over implications of $\mathcal{L}$.

The second tool used by Maier in his algorithm is *direct determination*. This notion is extensively discussed in [17, 24]. Thus, the definition we are going to define is more the result of a very useful necessary and sufficient condition than the definition as given at the beginning. Our aim is not to (re)build all the material, but to explain the algorithm.

Definition 13 (Direct Determination)**.** *Given a basis $\mathcal{L}$, we say that A directly determines B under $\mathcal{L}$, denoted $A \xrightarrow{\mathrm{d}} B$, if $\mathcal{L} - \bar{E}_{\mathcal{L}} \models A \longrightarrow B$.*

More intuitively, we say that A directly determines B if we can reach B without using any implications with body equivalent to A (including A). Those definitions are sufficient to understand the algorithm.

Algorithm

In this section we will investigate the Maier Algorithm to minimize a basis $\mathcal{L}$. The principle is short enough to describe it as follows:

1. remove redundant implications

2. remove direct determinations in this sens: if $A \xrightarrow{d} B$ with $A \longrightarrow C$ and $B \longrightarrow D$ implications of $\mathcal{L}$, remove $A \longrightarrow C$ and replace by $B \longrightarrow D$ by $B \longrightarrow D \cup C$.

Of course, proof of correctness has been given in cite [17, 24]. Nevertheless, we shall give later some different elements for proving the algorithm ends with the right result. We do not give them here, because those elements also provide a proof for an another algorithm (namely Ausiello and al. algorithm) through equivalence properties. For now, we send the reader to cited papers. The procedure is presented in algorithm 4.

Algorithm 4: Maier minimization algorithm

Input:
Output:

// redundancy elimination
foreach $A \longrightarrow B \in \mathcal{L}$ **do**
  $\mathcal{L}^- := \mathcal{L} - \{A \longrightarrow B\}$;
  if $\mathcal{L}^-(A) = \mathcal{L}(A)$ **then**
    $\mathcal{L} := \mathcal{L} - \{A \longrightarrow B\}$;
$\bar{E}_{\mathcal{L}} := EquivClasses(\mathcal{L})$;
// direct determination elimination
foreach $E_{\mathcal{L}} \in \bar{E}_{\mathcal{L}}$ **do**
  foreach $A \longrightarrow C \in E_{\mathcal{L}}$ **do**
    if $\exists B \longrightarrow D \in E_{\mathcal{L}}$ *such that* $A \xrightarrow{d} B$ **then**
      remove $A \longrightarrow B$ from $\mathcal{L}$;
      replace $B \longrightarrow D$ by $B \longrightarrow C \cup D$;

Observations There are some observations and questions to answer about procedure 4:

- How do we compute $\bar{E}_{\mathcal{L}}$?
- How do we check for direct determination?
- Does the operation of remove and replace alter an equivalence class more than removing one of its implications?

Let us take those questions in order. To find $\bar{E}_{\mathcal{L}}$, we will consider applying a variation of LinClosure to each implication of $\mathcal{L}$. Thus for each $A \longrightarrow B$, instead of returning closure of A under $\mathcal{L}$, we will a bit vector of size $|\mathcal{B}(\mathcal{L})|$ with the i-th entry being 1 if the body of this implication is in the closure of A, and 0 otherwise. the i-th bit of the vector is set if $count[i\text{-th implication}] = 0$ in the procedure. Running this modified version of LinClosure over all implications produces a $|\mathcal{B}(\mathcal{L})| \times |\mathcal{B}(\mathcal{L})|$ matrix such that the entry

$[A \longrightarrow C, B \longrightarrow D]$ is 1 if $\mathcal{L} \models A \longrightarrow B$. Consequently, finding equivalence classes shall take no more than a run over this matrix to be done (after having computed closures).

Direct determination can be considered as well with a modification of LinClosure on $\mathcal{L} - E_{\mathcal{L}}(A)$. In this version, for a given $A \longrightarrow C$, we also have to be provided with $E_{\mathcal{L}}(A)$. Then, direct determination will be exposed the first time we reach $count[B \longrightarrow D] = 0$ in the computation, for $B \longrightarrow D$ different from $A \longrightarrow C$ and belonging to $E_{\mathcal{L}}(A)$. Because we stop to the first time this will ensure that we did not reach any other body of this equivalence class, otherwise we would have stopped before. Because equivalence classes partition the implications of $\mathcal{L}$ we will proceed to this operation at most once for each implication.

Finally, we consider the question of whether equivalence classes are altered more than removal during computations. Of course we remove direct determination is found. Does replacing $B \longrightarrow D$ by $B \longrightarrow C \cup D$ has any chances to add other implication to $E_{\mathcal{L}}(A)$? Fortunately the answer is no. Indeed, before removing $A \longrightarrow C$ from $E_{\mathcal{L}}(A)$, anything we could reach from A could be reached from B (because of their equivalence) and thus would have implied C from A and D from B. Thus anything reachable from $C \cup D$ would already be in $E_{\mathcal{L}}(A)$.

Complexity Again, the complexity of this algorithm has been studied and discussed in [17, 24]. It has been challenged (somewhat) in [4, 5]. We provide a complexity analysis anyway to adapt previous material to our notations (and ease comparison with other algorithms) and to have this report to be self-contained as much as possible. Let us recall few elements to analyse complexity:

- $\mathcal{B}(\mathcal{L})$ is the number of distinct bodies in $\mathcal{L}$ (therefore the total number of implications for us), and $|\mathcal{B}(\mathcal{L})|$ its size,
- $|\,\Sigma\,|$ is the number of attributes in the universe Σ,
- $|\,\mathcal{L}\,|$ is the size of $\mathcal{L}$ in memory, that is roughly $|\mathcal{B}(\mathcal{L})| \times |\,\Sigma\,|$.

Let us proceed by steps. First, non-redundancy elimination. Because the inner **foreach** loop goes for all implications of $\mathcal{L}$ requiring $O(|\mathcal{B}(\mathcal{L})|)$ times looping. Then, computing closure of A under $\mathcal{L}$ and $\mathcal{L}^-$ requires $O(|\,\mathcal{L}\,|)$ operations thanks to LinClosure. Hence, redundancy elimination can be done in $O(|\mathcal{B}(\mathcal{L})| \times |\,\mathcal{L}\,|) = O(|\mathcal{B}(\mathcal{L})|^2 \times |\,\Sigma\,|)$ time. Secondly, finding equivalence classes as mentioned previously can be done by using modified version of LinClosure performing same time as the original one. Since we have to go over (at most) all implications to find all implications with bodies implied by a given one, this result in $O(|\mathcal{B}(\mathcal{L})| \times |\,\mathcal{L}\,|)$ complexity. Then, having the $|\mathcal{B}(\mathcal{L})| \times |\mathcal{B}(\mathcal{L})|$ matrix, building equivalence classes need a run over the matrix, that is $O(|\mathcal{B}(\mathcal{L})|^2)$ operations. Finally, removing direct determination can be done at most for all implications, and checking for this property is again made through LinClosure. Thus performing the last step of the algorithm is $O(|\mathcal{B}(\mathcal{L})| \times |\,\mathcal{L}\,|) = O(|\mathcal{B}(\mathcal{L})|^2 \times |\,\Sigma\,|)$ time. As a conclusion, the whole algorithm works in $O(|\mathcal{B}(\mathcal{L})| \times |\,\mathcal{L}\,|)$.

2.1.3 Ausiello Algorithm: minimality through directed graphs

This section is dedicated to a minimization algorithm relying on hypergraphs. It has been set up by Ausiello and al. in [6, 4, 5]. Starting from a directed hypergraph representation, it builds a special kind of DAG, called *FD-Graph* with which it reduces the initial hypergraph. In order, we are going to define what is a FD-graph, provide the general idea for the algorithm as explained in [5] and then go into further details and more precise algorithms for such computations as exposed in [4].

FD-Graphs and minimum covers

In this part, we assume that our basis $\mathcal{L}$ is represented through the framework of hypergraphs (see chapter 1). Again, all the definitions we are going to state here are exposed in [4, 5]. First, let us define the central object we will work on, FD-Graphs.

Definition 14 (FD-Graph)**.** *Given a hypergraph $\mathcal{L} = (\Sigma, C)$ representing an implication basis, the directed graph $G_{\mathcal{L}} = (V,\ E)$ such that:*

- *$V = V_0 \cup V_1$ is the set of nodes where:*
  - *$V_0 = \Sigma$ is the set of simple nodes (a node per attribute in Σ),*
  - *$V_1 = \{X | X \in \mathcal{B}(\mathcal{L})\}$ is the set of compound nodes (a node per distinct body in $\mathcal{L}$),*
- *$E = E_0 \cup E_1$ is the set of arcs where:*
  - *E_0 is the set of full arcs. We have a full arc (X, i) in E_0 if (X, i) is an hyperarc of $\mathcal{L}$,*
  - *E_1 the set of dotted arcs. For each compound node X of V[1], we have a dotted arc $(X,\ i)$ to every attributes i of X,*

is the Functionnal Dependency Graph or FD-Graph associated to $\mathcal{L}$.

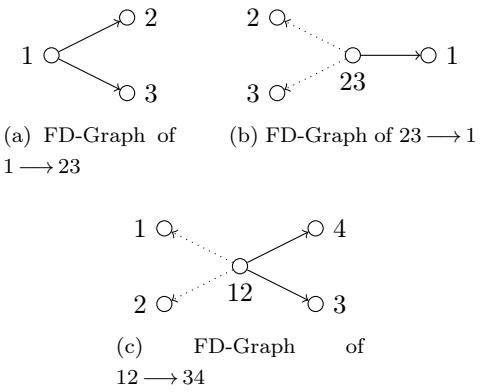
In fact, hypergraph representation is not mandatory to represent a FD-Graph, since as we saw, hypergraphs in our context rely on "usual" implicational basis. To enlighten the definition of FD-Graph, simple examples are shown in 2.1.

We can see in those example that drawing an FD-Graph goes this way:

- For each $A \longrightarrow B$ of $\mathcal{L}$ we draw a *full* arc from the node A to every attribute of B,
- For each compound node A, we draw a *dotted* arc from A to all of its attribute.

which is indeed what we formally defined previously. Furthermore, for this algorithm, we consider a basis $\mathcal{L}$ over an attribute set Σ, such that:

- there is no $A \longrightarrow B$, $A' \longrightarrow B'$ in $\mathcal{L}$ such that $A = A'$ when $B \neq B'$,
- for all $A \longrightarrow B$ of $\mathcal{L}$, $A \cap B = \emptyset$



(a) FD-Graph of $1 \longrightarrow 23$

(b) FD-Graph of $23 \longrightarrow 1$

(c) FD-Graph of $12 \longrightarrow 34$

Figure 2.1: Representation of some FD-graph

An important notion we may need for all the following is *FD-paths*.

Definition 15 (FD-Path). *Given an FD-Graph $G_{\mathcal{L}} = (V, E)$, an FD-Path $\langle i,\ j\rangle$ is a minimal subgraph $\bar{G}_{\mathcal{L}} = (\bar{V}, \bar{E})$ of $G_{\mathcal{L}}$ such that $i, j \in \bar{V}$ and either $(i, j) \in \bar{E}$ or one of the following holds:*

- *j is a simple node and there exists $k \in \bar{V}$ such that $(k, j) \in \bar{E}$ and there exists a FD-Path $\langle i,\ k\rangle$ included in $\bar{G}$,*
- *$j = \bigcup_{k=1}^{n} j_k$ is a compound node and there exists FD-paths $\langle i,\ j_k\rangle$ included in $\bar{G}$, for all $k = 1,\ \ldots,\ n$.*

As is, the definition may not seems clear. Informally, a FD-path from a node i to j describes the implications we use to derive $i \longrightarrow j$ (with Armstrong rules, especially transitivity and union). Intuitively, directed paths are FD-paths. But there is also one case in which we can go "backward" in the graph. For better understanding, see examples of figure 2.3 based on the basis described in figure 2.2.

There are either *dotted* or *full* paths. A path $\langle i, j\rangle$ is dotted if all arcs leaving i are dotted, it is full otherwise.

Having explained FD-Graphs, we will now move to explanations of the algorithm developed by Ausiello and al.

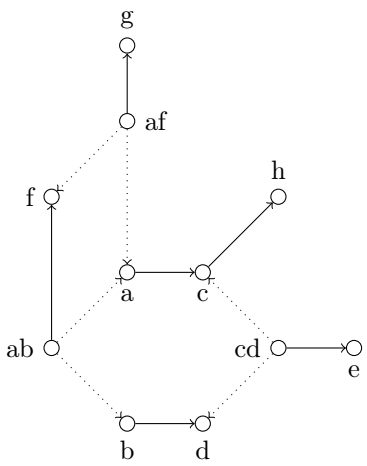


Figure 2.2: FD-Graph of some implicational basis

Ausiello algorithm: general point of view

The following procedure (5) finds from a given basis its *minimal cover*.

Algorithm 5: Ausiello algorithm (1986)

Input: $\mathcal{L}$ an implication basis
Output: $\mathcal{L}_c$ a minimal cover for $\mathcal{L}$

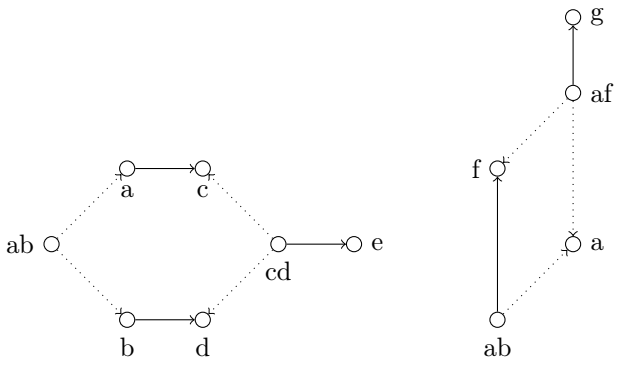
Find the *FD-Graph* of $\mathcal{L}$;
Remove *redundant* nodes ;
Remove *superfluous* nodes ;
Remove *redundant* arc ;
Derive $\mathcal{L}_c$ from the new graph ;

Let us emphasize on the meaning of those removal steps. The algorithm studied in the sources, aims to minimize an implicational basis in terms of bodies. As with the Duquenne-Guigues basis. It uses 3 steps. Two to remove redundancy, the third one aims to lighten bodies and heads of remaining implications. We will try to express each of these steps in terms of sets, closure, and so forth.

Removing redundant nodes

The first step is about removing redundant implications (without right maximization). In terms of FD-graphs, we remove *redundant* nodes. A compound node (only) i is said redundant if for each full arc ij leaving i there exists a dotted path (i, j). We give an example in the figure 2.4.

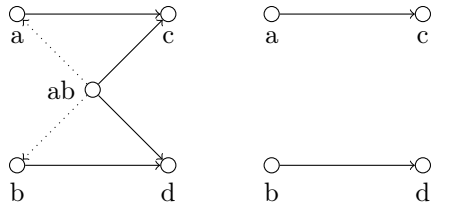
In this example, the basis associated basis is $\mathcal{L} = ab \longrightarrow cd\,;\, a \longrightarrow c\,;\, b \longrightarrow d$. Indeed, in this case, $ab \longrightarrow cd$ is redundant because $\mathcal{L} - ab \longrightarrow cd \models ab \longrightarrow cd$. So removing a redundant node is removing exactly one implication in $\mathcal{L}$ since $\mathcal{L}$ is reduced. In details, let $A \longrightarrow B$ be an implication of $\mathcal{L}$ with $A = a_1 a_2 \dots a_n$ and $B = b_1 b_2 \dots b_m$. $A \longrightarrow B$ will be redundant in terms of FD-Graph if for each b_i there exists $X_i \subset A$



(a) dotted FD-path for $ab \longrightarrow e$

(b) full FD-path for $ab \longrightarrow g$

Figure 2.3: Representation of some FD-paths



(a) FD-Graph with redundant node (ab)

(b) FD-Graph with redundant node removed

Figure 2.4: Elimination of redundant nodes

such that $X_i \longrightarrow b_i$. That is:

$$\bigcup_i X_i \subseteq A \longrightarrow B$$

Which may be rewritten in terms of $\mathcal{L}$ closure as

$$\bigcup_i \mathcal{L}(X_i) = \mathcal{L}(A)$$

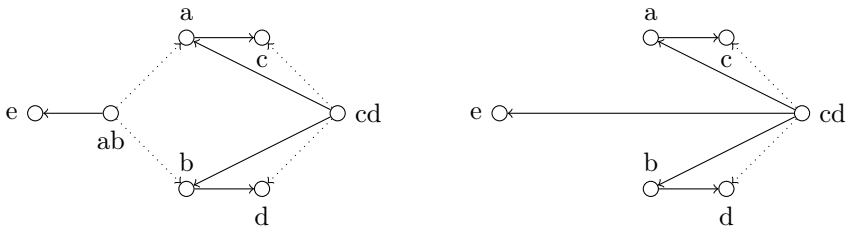
Thinking of opposite direction, $A \longrightarrow B$ will be nonredundant if there exists $b \in B$ such that $((b \in \mathcal{L}(X)) \wedge (X \subseteq A)) \longrightarrow (X = A)$.

To sum up: *the first step is about considering each $A \longrightarrow B$ where $|A| > 1$, and removing it of $\mathcal{L}$ if $\mathcal{L}^-(A) = \mathcal{L}(A)$*. Note: $\mathcal{L}^- = \mathcal{L} - A \longrightarrow B$.

Removing superfluous nodes

Next, we remove from the nonredundant FD-Graph *superfluous* nodes. A node i is *superfluous* if there is an equivalent node j and a dotted path from i to j. Two nodes i, j are *equivalent* if there are paths (i, j) and (j, i).

In terms of sets and closure, two attribute sets $A, B \subseteq \Sigma$ are equivalent in $\mathcal{L}$ if $\mathcal{L} \models A \longrightarrow B, B \longrightarrow A$, that is, if $\mathcal{L}(A) = \mathcal{L}(B)$.



(a) FD-Graph with superfluous node (ab)

(b) FD-Graph with superfluous node removed

Figure 2.5: Elimination of superfluous node

The algorithm suggests the following:

- find a superfluous node i, and an equivalent node j with a dotted path from i to j
- for each full arc ik, we add a full arc jk
- then we remove the node i and all of its outgoing arcs from the graph
- repeat until no more superfluous nodes exist

An example of this procedure is given in the figure 2.5. In this example $\mathcal{L} = ab \longrightarrow e\,;\, a \longrightarrow c\,;\, b \longrightarrow d\,;\, cd \longrightarrow ab$. The node ab is superfluous. Since our basis are reduced, note that removing a superfluous node is removing exactly one implication in $\mathcal{L}$. In this case, the resulting $\mathcal{L}$ will be

$$\mathcal{L} = a \longrightarrow c, b \longrightarrow d, cd \longrightarrow abe$$

Now we may rewrite this operation in our terms. Let $A \longrightarrow B$ and for instance $C \longrightarrow D$ be part of $\mathcal{L}$ to be general. Then A is superfluous body if

$$\mathcal{L} \models A \longrightarrow C, C \longrightarrow A \wedge \exists X \subset A \; s.t \; \mathcal{L} \models X \longrightarrow C$$

In this case, we apply the following operations

- $C \longrightarrow D$ becomes $C \longrightarrow (D \cup B)$
- we remove $A \longrightarrow B$ from $\mathcal{L}$

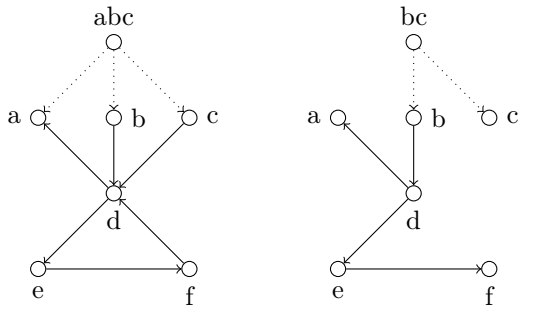
We exhibit some arguments to convince ourselves that this is a valid operation. Let us call temporarily $\mathcal{L}^-$ the basis we obtain after the previous operations. We would like to show that $\mathcal{L}^- \models A \longrightarrow B$, i.e that $\mathcal{L}^- \equiv \mathcal{L}$. We removed $A \longrightarrow B$ but we still have $\mathcal{L}^- \models X \longrightarrow C$ and then $X \longrightarrow B$ because $C \longrightarrow D \cup B$. That is, $B \subset \mathcal{L}^-(X)$. Because $X \subset A$, we have $B \subset \mathcal{L}^-(A)$ which is what we wanted.

Removing redundant arcs

Finally, we remove from a minimum nonredundant FD-Graph, *redundant arcs*:

- dotted case: a dotted arc ij is redundant if there is a dotted path (i, j) not using ij,
- full case: a full arc ij is redundant if there is a dotted/full path (i, j) not using ij.

we can think of eliminating redundant arcs as explicit transitivity elimination. If we remove a full arc in $A \longrightarrow B$ then we are minimizing B. If we remove a dotted arc, we are minimizing A. We have examples in 2.6.



(a) FD-Graph with redundant arcs ($abc \longrightarrow a$, $f \longrightarrow d$)

(b) FD-Graph with redundant arcs removed

Figure 2.6: Elimination of redundant arcs

In terms of sets, consider an implication $A \longrightarrow B$. Removing a dotted arc is saying that given $a \in A$, and substituting $A \longrightarrow B$ by $A - a \longrightarrow B$ in $\mathcal{L}$ preserves $\mathcal{L} \models A \longrightarrow a_i$. On the other side, removing a full arc is, given $b \in B$ and replacing by $A \longrightarrow B - b$, we preserve $\mathcal{L} \models A \longrightarrow b$. It appears that this steps goes beyond our scope, because it aims to minimize implication itself, not the number of implications. So, in our minimization context, we can stick to the first two operations to obtain a minimal representation in our terms. The algorithm we have to follow becomes the procedure 6

Algorithm 6: Ausiello algorithm (1986, reduced)

Input: $\mathcal{L}$ an implication basis
Output: $\mathcal{L}_c$ a minimal cover for $\mathcal{L}$

Find the *FD-Graph* of $\mathcal{L}$;
Remove *redundant* nodes ;
Remove *superfluous* nodes ;
Derive $\mathcal{L}_c$ from the new graph ;

In fact, this algorithm stands for a high-level principle of what has to be done to minimize a given basis. The question we are going to investigate in the next paragraphs is how to do such computations.

Ausiello algorithm: effective procedure

In this part we will focus on how the algorithm proposed by Ausiello in [5] performs redundancy and superfluousness elimination. Actually, those operations are detailed in [4]. But in order to stick to our subject (reviewing the existing algorithms) we study it here. Also, note that we will consider that the mapping step from $\mathcal{L}$ to $G_{\mathcal{L}}$ is already done. Also, as mentioned in the related articles, going from one representation to another can be done in linear time.

The algorithm relies on closure of a FD-Graph. Of course the idea of closure is similar to the one we encountered before. Nevertheless, Ausiello and al. introduce a notion of priority in the way they determine the closure. Because we cannot represent to different arcs with same nodes (two arcs $(i,\ j)$), me must provide priority between dotted and full arcs. The authors decided to stress on dotted arcs since they are more likely to be a criterion for removal during minimization.

Closure of a FD-Graph The closure is based on the following data structures:

- V_0: set of *simple* nodes,
- V_1: set of *compound* nodes,
- D_i $(\forall i \in V)$: nodes from *incoming dotted* arcs $\{j \in V \mid (j,\ i)$ is a dotted arc$\}$,
- L_i^0 $(\forall i \in V)$: nodes from *outgoing full* arcs $\{j \in V \mid (i,\ j)$ is a full arc$\}$,
- L_i^1 $(\forall i \in V)$: nodes from *outgoing dotted* arcs $\{j \in V \mid (i,\ j)$ is a dotted arc$\}$,
- L_i^{0+}, L_i^{1+} $(\forall i \in V)$: the respective closures of L_i^0, L_i^1,
- q_m $(\forall m \in V^1)$: counter of nodes in m belonging to $L_i^{0+} \cup L_i^{1+}$ for some $i \in V$.

To make understanding easier, we first give pseudo-code closer from principle than algorithms. From a general point of view, to determine the closure of a FD-graph, we must compute the closure of all its nodes. The closure of a node is described by its full and dotted outgoing arcs. Because we put a priority on dotted

possibilities, they will be computed before. Principle are given in algorithmic/pseudo-code form so that identification between steps of procedures and ideas of principle are easier to see.

First, we introduce the procedure NodeClosure which computes the closure of a node with respect to a type of arc. In other words, to compute the full closure of a node, we must first apply NodeClosure to its dotted arcs, then to its full arcs. The principle and algorithm for Nodeclosure are 7, 8.

Algorithm 7: NodeClosure (Principle)

Input: L_i: set of nodes for which there exists dotted (resp. full) arcs (i, j)
Output: L_i^+: the dotted (resp. full) closure of i

Initialize a list of nodes to treat S_i to L_i ;
while *there is a node j to treat in S_i* **do**
  remove j from S_i ;
  if *j is simple node* **then**
    forall *compound node m except i, j appears in* **do**
      increase q_m by 1 ;
      if *q_m = number of outgoing dotted arcs from m* **then**
        m is reachable from i by *union* ;
        m must be treated, add it to S_i ;
  add j to the closure L_i^+ ;
  forall *nodes k such that there is an arc $(j,\ k)$* **do**
    if *k is not yet in the closure L_i^+ or in the dotted closure L_i^{1+} of i* **then**
      k is reachable from i by *transitivity* ;
      k must be treated, add it to S_i ;
  return L_i^+ ;

We would like to provide some observations on top of their description. Namely on the *union* step and q_m counters. Say $i \longrightarrow m$ where m is a compound node is a valid implication in a FD-graph. Furthermore say $m = \bigcup_i m_i$ where m_i's are simple nodes. The union step models the fact that if we have $i \longrightarrow m_i$ for all m_i in m, then we must have $i \longrightarrow m$ also. The counter q_m ensures that we indeed reached all m_i's in m. Also, the algorithm has access to all the structures we described above (nodes, sets of arcs, and so forth). Parameters are thus lists we are going to modify somewhat. The NodeClosure algorithm runs in time $O(|\mathcal{L}|)$. The first nested loop runs in at most $O(|\Sigma| \times |\mathcal{B}(\mathcal{L})|) = O(|L|)$ because S_i contains at most $|\Sigma|$ elements, and the block referring to the *union* rule runs over compound nodes, that is bodies of $\mathcal{L}$. For the second loop (transitivity) note that we can at most consider all the edges of the FD-graph. In fact, the cost of transitivity operation for all j is $O(\sum_{j=1}^{n} |L_j^0 \cup L_j^1|)$. But by definition, those sets are disjoints, and therefore we cannot treat more than $|E|$ arcs (the total number of arcs in G), that is $|\mathcal{L}|$.

Algorithm 8: NodeClosure

Input: L_i: set of nodes for which there exists dotted (resp. full) arcs (i, j)
Output: L_i^+: the dotted (resp. full) closure of i

$S_i := L_i$;
while $S_i \neq \emptyset$ **do**
  select j from S_i ;
  if $j \in V^0$ **then**
    forall $m \in D_j - \{i\}$ **do**
      $q_m := q_m + 1$;
      if $q_m = |L_m^1|$ **then**
        $S_i := S_i \cup \{m\}$;
  $S_i^+ := S_i^+ \cup \{j\}$;
  forall $k \in L_j^0 \cup L_j^1$ **do**
    if $k \notin S_i^+ \cup L_i^{1+} \cup \{i\}$ **then**
      $S_i := S_i \cup \{k\}$;
return L_i^+ ;

Next, we present the principle and pseudo-code for the closure of an FD-graph 9, 10. Mostly, the principle is the idea we described previously. There is just one observation to make about setting a counter q_m to 1. This variable helps to see whether we can use union rule as we saw in procedure NodeClosure (7, 8). We initialize it in case i is indeed part of some compound node so that we do not omit to count it when dealing with S_i (because S_i does not contain i). In terms of complexity, we are running NodeClosure on all nodes having outgoing edges, that is $|\mathcal{B}(\mathcal{L})|$ nodes (if a compound node is represented, it must have at least one outgoing full arc). Since NodeClosure operates in $O(|\mathcal{L}|)$, the whole closure algorithm must run in $O(|\mathcal{B}(\mathcal{L})| \times |\mathcal{L}|)$.

Now that algorithms for computing the closure of a FD-graph have been set, we can move to the minimization part.

Minimization algorithm for FD-Graphs We have two steps in this algorithm. First, we need to remove redundant nodes, then superfluous nodes. To delete redundant nodes, the claim of [4, 5] is that removing nodes with dotted arcs only in the closure of an FD-Graph is sufficient. Indeed, if a node i has only dotted paths in the closure of an FD-graph, it means that for every $i \longrightarrow j$ holding, we can find a subset of k of i such that $k \longrightarrow j$. In our terms, it is saying that $\mathcal{L}(i) = \mathcal{L}^-$ where $\mathcal{L}^-$ denotes $\mathcal{L}$ from which we removed the implication having i as a body.

Let us try to investigate the complexity of this algorithm. According to the article and previous study, Closure is achieved in $O(|\mathcal{B}(\mathcal{L})| \times |\mathcal{L}|)$. If the graph is represented as adjacency lists, deleting a node

Algorithm 9: Closure (Principle)

Input: V_0, V_1 and $\forall i \in V$ D_i, L_i^0, L_i^1
Output: $\forall i \in V$ L_i^{0+}, L_i^{1+}

forall *node* i *in* V *with outgoing arcs* **do**
  if i *is an attribute of a compound node* m **then**
    set a counter q_m to 1 ;
  initialize the closure of i to $\emptyset$;
  if i *is a compound node* **then**
    determine *dotted* arcs in the closure of i ;
  determine *full* arcs in the closure of i ;

Algorithm 10: Closure

Input: V_0, V_1 and $\forall i \in V$ D_i, L_i^0, L_i^1
Output: $\forall i \in V$ L_i^{0+}, L_i^{1+}

forall $i \in V$ *with* $L_i^0 \cup L_i^1 \neq \emptyset$ **do**
  forall $m \in V^1$ **do**
    if $m \in D_i$ **then**
      $q_m := 1$;
    else
      $q_m := 0$;
  $L_i^{1+} := \emptyset$;
  $L_i^{0+} := \emptyset$;
  if $i \in V^1$ **then**
    $L_i^{1+} :=$ NodeClosure(L_i^1) ;
  $L_i^{0+} :=$ NodeClosure$(L_i^0 - L_i^{1+})$;

Algorithm 11: Ausiello Minimization Algorithm

Input: $G_{\mathcal{L}}$: the FD-Graph of some basis $\mathcal{L}$

Output: $G_{\mathcal{L}_c}$: the associated minimum FD-Graph

$G^+_{\mathcal{L}} = Closure(G_{\mathcal{L}})$;

// Redundancy Elimination

forall $i \in V^1$ **do**

  if $L_i^{1+} = \emptyset$ **then**

    remove i and its outgoing arcs from $G_{\mathcal{L}}$;

// Superfluousness Elimination

forall $i \in V^1$ **do**

  find an equivalent node j ;

  if j *exists* **then**

    $L_j^{0+} := L_j^{0+} \cup (L_i^{0+} \cap L_j^{1+})$;

    $L_j^{1+} := L_j^{1+} - (L_i^{0+} \cap L_j^{1+})$;

  remove i from the closure ;

  add (i, j) to a list L ;

remove superfluous nodes ;

move arcs to their final destination ;

and its outgoing arc can be $O(1)$ (it consists in freeing the node and its associated two lists). Running over all compound nodes is $O(|\mathcal{B}(\mathcal{L})|)$. In the case we would have to remove arcs one by one, note that we would have at most $O(|\Sigma|)$ arcs to delete. This would yield a $O(|\mathcal{B}(\mathcal{L})| \cdot |\Sigma|) = O(|\mathcal{L}|)$ complexity. *How to remove compound nodes that have disappeared from the closure of other nodes ? It is time consuming*. It is $O(|\mathcal{B}(\mathcal{L})| \cdot |\mathcal{L}^+|)$. We don't count for intern data structure operation. We can consider that adding and removing can be done in $O(1)$. The first loop of superfluousness elimination goes over all compounds nodes, and finding an equivalent node is $O(V)$ Moving elements in lists associated to j is also $O(V)$ thus the first loop is $O(V^1 \cdot V)$. The second loop may need to go over all the full arcs of the FD-graph, which account for a $O(|\mathcal{L}|)$ complexity. As a consequence the whole algorithm may run in $O(|\mathcal{B}(\mathcal{L})| \cdot |\mathcal{L}|)$ (because of the closure computation).

The remaining part of this section will be devoted to propose ways to implement efficiently the principles we just presented. By "implementing efficiently" we mean to give algorithms, using adjacency list structure, to perform the operations with required amount of time. In our terms, time complexity for finding and removing superfluousness must not exceed $O(|B|^2 + |\mathcal{L}|)$. We divide those computations in 2 parts:

1. removing superfluous nodes within the closure (see algorithm 12)
2. removing superfluous nodes in the initial graph (see algorithm 13)

Let us focus first on superfluousness elimination within the closure of an FD-Graph. Let us consider G^+ to be the closure of some FD-graph G. First step is to find superfluous nodes. For the recall, this property can hold only for compound bodies of the implication system G models. At first sight, one could think of finding superfluousness as follows:

foreach $i \in V_c$ **do**
  foreach $j \in L_i^{1+}$ **do**
    foreach $k \in L_j^{0+} \cup L_j^{0+}$ **do**
      if $k = i$ **then**
        i superflous w.r.t j ;

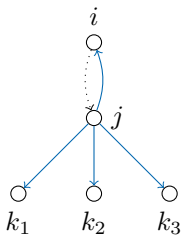


Figure 2.7: Finding superfluous nodes in FD-Graphs

The issue being the required amount of time to perform such computations. Indeed, Such nested loops will account for $O(|V_c||V|^2) = O(|B|(|B| + |\Sigma|)^2) = O(|B|^3 + |B||\mathcal{L}| + |\mathcal{L}||\Sigma|)$ operations.

Algorithm 12: SuperfluousnessClosureElimination

Input: $G^+_{\mathcal{L}}$: the closure of some FD-Graph $G_{\mathcal{L}}$
Output: L: list of pair of nodes to be altered in the initial graph

$L = \emptyset$;
foreach $i \in V_c$ **do**
  foreach $j \in V$ **do**
    $color(j) := 0$;
  foreach $j \in L_i^{1+}$ *s.t* $color(j) = 0$ **do**
    foreach $k \in L_j^{0+} \cup L_j^{1+}$ *s.t* $color(k) = 0$ **do**
      $color(k) := 1$;
      if $k = i$ **then**
        i *superfluous* ;
        keep j ;
  if i *superfluous (w.r.t* j*)* **then**
    foreach $k \in L_i^{0+}$ **do**
      $color(k) := 2$;
    foreach $k \in L_j^{1+}$ **do**
      if $color(k) = 2$ **then**
        move k to L_j^{0+} ;
    $L_i^{0+} = \emptyset$;
    $L_i^{1+} = \emptyset$;
    add (i, j) to L ;
foreach $i \in V$ *s.t* i *not superfluous* **do**
  foreach $j \in L_i^{0+} \cup L_i^{1+}$ **do**
    if j *superfluous* **then**
      remove j from $L_i^{0+} \cup L_i^{1+}$;

2.1.4 Elements of proofs

This section needs to be adjusted and corrected. In fact, statements lacks some precision, but conclusion is unchanged: both algorithm (Maier and Ausiello) performs computations on same objects, but according to the following statement

Proposition 2. *We have equivalence between the following:*

Algorithm 13: SuperfluousnessElimination

Input: L: a list of pair of vertices (i, j), i superfluous w.r.t j

dest array of size $|V|$;
foreach $v \in V$ **do**
    $dest[v] := v$;
    $color(v) := 0$;

foreach $(i, j) \in L$ *in reverse order* **do**
    $dest[i] = dest[j]$;

foreach $i \in V$, i *superfluous* **do**
    foreach $j \in L^0_{dest[i]} \cup L^1_{dest[j]}$ **do**
        $color(j) := 1$;
    foreach $j \in L^0_i$, *s.t* $color(j) = 0$ **do**
        move j to $L^0_{dest[i]}$;
    Reset all colors to 0 ;

foreach $i \in V$, i *is simple* **do**
    foreach $j \in D_i$, j *superfluous* **do**
        remove j from D_i ;

Remove superfluous nodes from V ;

(i) $A \equiv B$ *and* $A \xrightarrow{\mathrm{d}} B$*,*

(ii) A *is superfluous (in particular, "minimal" superfluous with respect to* B*),*

(iii) third part of proposition 7. (to be adapted)

In this section we write various claims which help us prove correctness and equivalence of Maier/Ausiello's algorithms. Knowledge about FD-graphs is assumed. Equivalence of redundancy elimination is not shown (1st step of the algorithm). We place ourselves in a non-redundant context to match definitions of direct determination. Non-redundancy does not affect superfluous definition.

Proposition 3. *The following properties are equivalent (let* $A \longrightarrow C$ *be the implication of* $\mathcal{L}$ *with* A *as body):*

(i) A *node* A *in a FD-graph is superfluous with respect to* B*,*

(ii) $A \equiv B$ *and* $\mathcal{L} - \{A \longrightarrow C\} \models A \longrightarrow B$*.*

Proof. (i) $\longrightarrow$ (ii). If A is superfluous, we have a dotted FD-Path $\langle A,\ B\rangle$. Since it is dotted, let us remove this node A and its outgoing arcs. Actually, none of the nodes pointed by dotted arcs of A have been removed, thus we can still find the nodes a_i (attributes of a) used in the dotted path $\langle A,\ B\rangle$ such that $\bigcup_i a_i \models B$. Because $\bigcup_i a_i \subseteq A$, we end up with $\mathcal{L} - \{A \longrightarrow C\} \models A \longrightarrow B$.

(ii) $\longrightarrow$ (i). In the FD-Graph associated to $\mathcal{L} - \{A \longrightarrow C\}$, the node A is not present. But still, $A \longrightarrow B$ holds. This means that we must be able to find a list of proper subsets A_i of A (possibly single attributes) such that $\bigcup_i A_i \models B$. Adding $A \longrightarrow C$ will add the node A and in particular dotted arcs from A to each attributes of $\bigcup_i A_i \subseteq A$. Thus, we will have a dotted path from A to $\bigcup_i A_i$ and consequently, to B. A is indeed superfluous. Because B is equivalent to A by assumption, this property is preserved when adding a node.

□

Proposition 4. *The following statements are equivalent:*

(i) $A \xrightarrow{\mathrm{d}} B$*,*

(ii) $\mathcal{L} - E_{\mathcal{L}}(A) \models A \longrightarrow B$*.*

Proof. Translation of notions. $A \xrightarrow{\mathrm{d}} B$ means that if we remove all implications with left sides equivalent to A we can still derive $A \longrightarrow B$ using remaining implication, that is $\mathcal{L} - E_{\mathcal{L}}(A) \models A \longrightarrow B$. □

Proposition 5. *the following statements are equivalent, for* A, B *bodies of* $\mathcal{L}$*:*

(i) $A \xrightarrow{\mathrm{d}} B$ *and* $B \equiv A$*,*

(ii) the node A *is superfluous with respect to* B*, and there exists a dotted FD-path from* A *to* B *not using any outgoing full arcs of nodes equivalent to* A*.*

Proof. (i) $\longrightarrow$ (ii). Using propositions 4, 3, showing that there is a direct determination starting from A implies A is a superfluous node in the FD-Graph is straightforward. If $\mathcal{L} - E_{\mathcal{L}}(A) \subseteq \mathcal{L} - \{A \longrightarrow C\} \models A \longrightarrow B$ so does $\mathcal{L} - \{A \longrightarrow C\}$. This holds in particular if $B \subseteq A$. Moreover, notice that using an outgoing full arc from a node D equivalent to A is exactly using an implication with left hand side equivalent to A. Therefore, if there is not dotted FD-path from A to B not using those arcs, we would contradict direct determination.

(ii) $\longrightarrow$ (i). Suppose A is superfluous and there exists a dotted FD-path from A to B not using any outgoing full arcs from nodes equivalent to A. Those full arcs represent exactly the implications contained in $E_{\mathcal{L}}(A)$. Since we don't use them, the path still holds in $\mathcal{L} - E_{\mathcal{L}}(A)$ (we would remove compound nodes without outgoing full arcs of course, but this would only make the path stops to attributes instead of compound node). Having this path in $\mathcal{L} - E_{\mathcal{L}}(A)$ means that $\mathcal{L} - E_{\mathcal{L}}(A) \models A \longrightarrow B$.

□

Proposition 6. *The following two statement are equivalent, given the FD-graph of $\mathcal{L}$:*

(i) A is a superfluous node,

(ii) A is superfluous with respect to B, and there exists a dotted path from A to an equivalent node not using any outgoing full arcs from nodes equivalent to A.

Proof. (ii) $\longrightarrow$ (i) is trivial. let's focus on (i) $\longrightarrow$ (ii). If A is superfluous, then there exists B such that $A \equiv B$ and there has a dotted path from A to B.

If $B \subseteq A$, the dotted path is straight forward. If it is not the case, path from A to B uses outgoing full arcs from nodes equivalent to A, or it does not. If it does not we are done. Now Suppose this path uses an outgoing full arc from a node C equivalent to A. This means, that $A \equiv B \equiv C$ by definition and therefore, there exists a dotted path from A to C (because we need to reach C, so to derive it, to use its outgoing arcs). We can reiterate this reasoning until reaching A. Getting to A or one of its subset would contradict our assumptions meaning that we stopped finding used equivalent arcs earlier.

□

From the previous claims, we can yield the following one

Proposition 7. *The following statements are equivalent:*

(i) there exists $B \equiv A$ such that $A \xrightarrow{\mathrm{d}} B$,

(ii) A is superfluous with respect to B,

(iii) there exists C such that $A \equiv C$ and $\mathcal{L} - \{A \longrightarrow D\} \models A \longrightarrow C$. (possibly, $C = B$).

Proof. (i) $\longleftrightarrow$ (ii) comes from propositions 5 and 6. (ii) $\longleftrightarrow$ (iii) comes from proposition 3.

□

Those claims help to see the relation between operations of the algorithms. Indeed, the last claim states that finding a direct determination is the same as finding a superfluous node (under equivalence constraint), therefore the two algorithms are looking for the same structures in different terminologies. This emphasizes the fact they work on the same computations. Remark: Maier algorithm do it in a special order (some kinds of minimal paths in terms of FD-graph. Can we remove them in any order? It seems to since this is what the Ausiello algorithm does, and the previous proposition states that direct determination is equivalent to the existence of superfluous nodes.

The next claim provides an argument in this way. It also explains in what extent the operation of removal in those algorithm is exact.

Proposition 8. *Let $A \longrightarrow C, B \longrightarrow D$ be implications of $\mathcal{L}$. If $A \equiv B$ and $\mathcal{L} - \{A \longrightarrow C\} \models A \longrightarrow B$, then $\mathcal{L}$ is not minimum.*

Proof. Let $\mathcal{L}$ be an implication basis with implications $A \longrightarrow C, B \longrightarrow D$, $A \equiv B$, and such that $\mathcal{L} - \{A \longrightarrow C\} \models A \longrightarrow B$. Let us build $\mathcal{L}_c$ by removing $A \longrightarrow C$ from $\mathcal{L}$, and replacing $B \longrightarrow D$ by $B \longrightarrow C \cup D$. Obviously, $\mathcal{L}_c \models A \longrightarrow B$. Moreover, $A \longrightarrow C$ still holds since $A \longrightarrow B \longrightarrow C \cup D$. □

This proposition can be stated as its contraposition. That is, if a basis is minimum then we cannot find equivalent bodies with one to be removed. This states that the second step of Ausiello and Maier algorithms end up with a body minimal basis, with help of equivalences proved in proposition 7.

2.2 Minimization by construction

2.2.1 An algorithm relying on a bounding theorem

In this section we are going to expose an algorithm using the so-called Boros theorem (see [14, 13, 12]). Depending on our needs we will state some definitions and some propositions of the sources cited. The algorithm exposed in this part can be found in [14].

Theoretical setting

Again, all the terms we are going to define here can be found in [14, 13, 12]. The articles from Boros use the logical terminology. The context is pure Horn functions, and by Horn function, Horn CNF, we will mean pure.

Definition 16 (Implicates)**.** *Let $\mathcal{L}$ be a Horn CNF. A clause $\mathcal{C}$ is an implicate of $\mathcal{L}$ if $\mathcal{L} \models \mathcal{C}$. $\mathcal{I}(\mathcal{L})$ is the set of implicates of $\mathcal{L}$.*

In fact, $\mathcal{I}(\mathcal{L})$ can be seen as the closure of $\mathcal{L}$, the set of all implications following from $\mathcal{L}$.

Definition 17 (Essential sets)**.** *Let $\mathcal{L}$ be a Horn CNF, and $X \subseteq \Sigma$. The essential set associated to X, $\mathcal{E}_X$ is the set of clauses of $\mathcal{I}(\mathcal{L})$ for which X is a false point:*

$$\mathcal{E}_X = \{\mathcal{C} \in \mathcal{I}(\mathcal{L}) \mid \mathcal{B}(\mathcal{C}) \subseteq X, \mathcal{H}(\mathcal{C}) \notin X\}$$

A few remark on this definition. It could be extended in terms of implicational basis with the following:

$$\mathcal{E}_{\mathcal{X}} = \{\mathcal{A} \longrightarrow \mathcal{B} | \mathcal{A} \subseteq \mathcal{X}, \mathcal{B} \nsubseteq \mathcal{X}\}$$

with a detail to write down anyway. Remind that a clause $\mathcal{C}$ can be rewritten as $A \longrightarrow b$. If $A \longrightarrow b_1$ and $A \longrightarrow b_2$ are elements of $\mathcal{E}_{\mathcal{X}}$ then obviously $A \longrightarrow (b_1 \wedge b_2)$ will be. On the other side, $A \longrightarrow B$ being part of $\mathcal{E}_{\mathcal{X}}$ implies that there is *at least* one $b \in B$ such that $A \longrightarrow b$ is in $\mathcal{E}_{\mathcal{X}}$ also. It does not necessarily concern all the elements of B. Actually, this detail does not cause any problem for the next discussion because existence of such $b \in B$ is sufficient.

state MinMax Theorem.

Algorithm

Pseudo-code and principle The algorithm relies on hypergraph representation. The principle is to construct a minimal basis from a pure Horn CNF, by adding successivly implications with pairwise body-disjoint essential sets. Thanks to the theorem of Boros, this can be done only a minimal number of times, leading to body minimal representation. The pseudo-code is

Algorithm 14: BodyMinimal (Hypergraphs)

Input: $\mathcal{L}$ an implicational basis

Output: $\mathcal{L}_c$ a body-minimal representation of $\mathcal{L}$

Actually, the graph representation is not mandatory. Indeed the authors use hypergraphs because of a forward chaining procedure in linear time, and also due to their proofs relying on graphs. We could rewrite this algorithm as in 15

Algorithm 15: BodyMinimal)

Input: $\mathcal{L}$ an implicational basis

Output: $\mathcal{L}_c$ a body-minimal representation of $\mathcal{L}$

$\mathcal{L}_c := \emptyset$;

while $\exists X \in \mathcal{B}(\mathcal{L}) : \ \mathcal{L}(X) \neq \mathcal{L}_c(X)$ **do**

  $A := \min(\mathcal{L}_c(X) : \ X \in \mathcal{B}(\mathcal{L}) \wedge \mathcal{L}(X) \neq \mathcal{L}_c(X))$;

  $B := \mathcal{L}(X)$;

  $\mathcal{L}_c := \mathcal{L}_c \cup \{A \longrightarrow B - A\}$;

return $\mathcal{L}_c$;

Elements of proof of correctness

Proposition 9. *If two distinct sets P_1, P_2 are pseudo-closed, their essential sets are body disjoint.*

Proof. We are going to consider various cases. First suppose $P_1 \subset P_2$. Denote $\mathcal{E}_{P_1}, \mathcal{E}_{P_2}$ their essential sets. Suppose there exists $A \longrightarrow b_1$ in $\mathcal{E}_{P_1}$, and $A \longrightarrow b_2 \in \mathcal{E}_{P_2}$. If $b_2 \neq b_1$, we would have $b_2 \notin P_2$ and $b_2 \in P_1$ because $A \longrightarrow b_2 \notin \mathcal{E}_{P_1}$ contradicting $P_1 \subset P_2$. Then, let $b = b_1 = b_2$. We have $A \longrightarrow b \in \mathcal{E}_{P_1}$ and $A \longrightarrow b \in \mathcal{E}_{P_2}$. Hence:

- $b \notin P_1$ and $b \notin P_2$
- $A \longrightarrow b \in \mathcal{E}_{P_1}$ implies $\mathcal{L} \models A \longrightarrow b$. Because $A \subseteq P_1$, $\mathcal{L} \models P_1 \longrightarrow b$ holds too. That is $b \in \mathcal{L}(P_1)$

Recall that $P_1 \subset P_2$ and P_1, P_2 are pseudo-closed. Thus, $\mathcal{L}(P_1) \subseteq P_2$. But $b \in \mathcal{L}(P_1)$ and $b \notin P_2$ which is a contradiction. So, if $P_1 \subset P_2$ their essential sets are indeed body disjoint. Note that this case holds also if $P_1 = \emptyset$.

Suppose now $P_1 \cap P_2 = \emptyset$. If one of them is empty, the paragraph shows the result. Otherwise, their essential sets can obviously not contain implication with same bodies, contradicting the empty intersection.

Finally, consider the case $P_1 \cap P_2 \neq \emptyset$. If there is no false point in their intersection, then there is np $A \longrightarrow b$ with $b \notin A$, $A \subset P_1 \cap P_2$ holding. Thus their essential sets cannot share any implications with same body. Hence, let $A \subseteq P_1 \cap P_2$ be the greatest false point of $\mathcal{L}$ within $P_1 \cap P_2$:

(i) if $A \subset P_1 \cap P_2$, then $\mathcal{L}(P_1 \cap P_2) = P_1 \cap P_2$.

(ii) if $A = P_1 \cap P_2$, then by proposition **??**, $P_1 \cap P_2$ must be pseudo-closed.

case (i). Suppose $A \longrightarrow b_1 \in \mathcal{E}_{P_1}$ and $A \longrightarrow b_2 \in \mathcal{E}_{P_2}$. Again either $b = b_1 = b_2$ or $b_1 \neq b_2$. If $b_1 = b_2 = b$ then $b \notin P_1 \cap P_2$ but $b \in \mathcal{L}(A) \subseteq \mathcal{L}(P_1 \cap P_2) = P_1 \cap P_2$ which is a contradiction. If $b_1 \neq b_2$ we have $b_1 \notin P_1$ but $b_1 \in \mathcal{L}(A) \subseteq \mathcal{L}(P_1 \cap P_2) = P_1 \cap P_2 \subseteq P_1$ which is a contradiction. The same goes for b_2. In fact we implicitly used the notion of separation used in [14].

case (ii). P_1, P_2 and $P_1 \cap P_2$ being pseudo-closed leads to:

- $P_1 \cap P_2 \subset \mathcal{L}(P_1 \cap P_2) \subset P_1$,
- $P_1 \cap P_2 \subset \mathcal{L}(P_1 \cap P_2) \subset P_2$

which can be used the same way as in case (i), concluding the proof.

□

Several elements of proofs have been given in [14] for the algorithm 15. Here we shall exhibit another kind of proof, using pseudo-closed sets. The claim is that the algorithm end up with the Duquenne-Guigues basis. Therefore we will show that in the procedure, if we add an implication, its premise is pseudo-closed.

Proposition 10. *In the algorithm BodyMinimal, we add an implication $\mathcal{L}_c(X) \longrightarrow \mathcal{L}(X) - \mathcal{L}_c(X)$ only if $\mathcal{L}_c(X)$ is pseudo-closed.*

Proof. Let us prove this proposition by induction. Our hypothesis that for all $0 \leq n \leq |\mathcal{B}(\mathcal{L})|$, at step n, taking $A = \min(\mathcal{L}_c(X) : X \in \mathcal{B}(\mathcal{L}) \wedge \mathcal{L}(X) \neq \mathcal{L}_c(X))$ implies that A is pseudo-closed.

Initial case At $n = 0$, taking A in $min(I_c(X))$ among bodies of $\mathcal{L}$ is taking A in $min(\mathcal{B}(\mathcal{L}))$ since $I_c = \emptyset$. Then, because minimal bodies of $\mathcal{L}$ are minimal false points of $\mathcal{L}$, $A = min(\mathcal{B}(\mathcal{L}))$ will be pseudo-closed.

Induction Suppose that the hypothesis is valid up to step n. Let us prove it for the step $n+1$. Consider taking A as in the algorithm. We have some cases:

(i) $\mathcal{L}_c(X) = X$

(ii) $X \subset \mathcal{L}_c(X)$

case (i). For all implications $E \longrightarrow F$ of $\mathcal{L}_c$ we have two possibilities, either $E \subseteq A$ or $E \nsubseteq A$. We cannot have $A \subset E$, because we would contradict the minimality constraint over closures of bodies. If for all implications $E \nsubseteq A$ we are in the initial case, because there is no subset of A being a false point of $\mathcal{L}$, then A is pseudo-closed. Now let $E \longrightarrow F$ be an implication such of the basis we are building such that $E \subset A$. Recall that E is a pseudo-closed set of $\mathcal{L}$ by induction hypothesis. Then since $A = \mathcal{L}_c(X) = X$, we conclude that $\mathcal{L}_c(E) = \mathcal{L}(E) \subseteq A$. In other word, E is pseudo-closed and its closure is included in A. Also, note that thanks to minimality constraint we must have taken all pseudo-closed sets included in A before the $n+1$-th iteration of the algorithm. Because this reasoning holds for all such implications, we can conclude that A is indeed pseudo-closed.

case (ii) Suppose $X \subset \mathcal{L}_c(X)$. Then there exists at least one pseudo-closed set E such that $E \longrightarrow \mathcal{L}(E)$ is in $\mathcal{L}_c$. Therefore $\mathcal{L}(E) \subset \mathcal{L}_c(X)$. Hence thanks to minimality constraint and induction hypothesis, all the pseudo-closed included in $\mathcal{L}_c(E)$ are in $\mathcal{L}_c$. $\mathcal{L}_c(X)$ being a false point of $\mathcal{L}$, we conclude that $\mathcal{L}_c(X) = A$ is indeed pseudo-closed.

Since the induction hypothesis is true for the initial step and for a general step n, we conclude that it is true in general, which proves the property.

□

Proposition 10 shows that the algorithm produces indeed a body minimal basis from $\mathcal{L}$ and in particular, the Duquenne-Guigues basis.

Complexity analysis

2.2.2 Angluin Algorithm: Query Learning

In this section we are interested in the Angluin algorithm (1992). [2, 3].

Query Learning

Here, the method for building a minimal base is slightly different. We use so-called *query learning*. The idea is we formulate *queries* to an *oracle* knowing the basis we are trying to learn. The oracle is assumed to provide an answer to our query in constant time. Depending on the query, it might also provide informations on the object we are looking for. For the Angluin algorithm, we need 2 types of queries. Say we want to learn a basis $\mathcal{L}$ over Σ:

1. *membership* query: is $M \subseteq \Sigma$ a model of $\mathcal{L}$? The oracle may answer "yes", or "no".
2. *equivalence* query: is a basis $\mathcal{L}'$ equivalent to $\mathcal{L}$? Again the answers are "yes", or "no". In the second case, the oracle provides a *counterexample* either positive or negative:

(i) *positive*: a model M of $\mathcal{L}$ which is not a model of $\mathcal{L}'$,

(ii) *negative*: a non-model M of $\mathcal{L}$ being a model of $\mathcal{L}'$.

To clarify, the terms negative/positive are related to the base $\mathcal{L}$ we want to learn.

Algorithm

The algorithm has been proved to end up with the Duquenne-Guigue Basis, again we can refer to [2, 3] for further explanations. The first algorithm provided by Angluin et al. relies on clauses with unitary heads. It uses two operations allowing to reduce implications:

- $refine(A \longrightarrow B, M)$: produces $M \longrightarrow \Sigma$ if $B = \Sigma$, $M \longrightarrow B \cup A - M$ otherwise,
- $reduce(A \longrightarrow B, M)$: produces $A \longrightarrow M - A$ if $B = \Sigma$, $A \longrightarrow B \cap M$ otherwise.

Algorithm 16: Angluin Algorithm

Input: $\mathcal{L}$
Output: $\mathcal{L}_c$

$\mathcal{L}_c = \emptyset$;
while *not equivalence*($\mathcal{L}_c$) **do**
  M is the counterexample ;
  if M *is positive* **then**
    foreach $A \longrightarrow B \in \mathcal{L}_c$ *such that* $M \not\models A \longrightarrow B$ **do**
      replace $A \longrightarrow B$ by $reduce(A \longrightarrow B, M)$;
  else
    foreach $A \longrightarrow B \in \mathcal{L}_c$ *such that* $A \cap M \subset A$ **do**
      membership($M \cap A$) ;
    if *Oracle replied "no" for at least one* $A \longrightarrow B$ **then**
      Take the first such $A \longrightarrow B$ in $\mathcal{L}_c$;
      replace $A \longrightarrow B$ by $refine(A \longrightarrow B, A \cap M)$;
    else
      add $M \longrightarrow \Sigma$ to $\mathcal{L}_c$;
return $\mathcal{L}_c$;

Some Observations

- Proved to be polynomial ([2])
- Proved to end up on the DG basis ([3])

- We can see that (apart from previous remark) that we end up on implications of the form $A \longrightarrow \mathcal{L}(A)$ (induction)
- .

Conclusion

Bibliography

[1] Aho, A. V., Garey, M. R., and Ullman, J. D. The Transitive Reduction of a Directed Graph. *SIAM Journal on Computing* (July 2006).

[2] Angluin, D., Frazier, M., and Pitt, L. Learning conjunctions of Horn clauses. *Machine Learning 9*, 2 (July 1992), 147–164.

[3] Arias, M., and Balcázar, J. L. Canonical Horn Representations and Query Learning. In *Algorithmic Learning Theory* (Berlin, Heidelberg, 2009), R. Gavaldà, G. Lugosi, T. Zeugmann, and S. Zilles, Eds., Springer Berlin Heidelberg, pp. 156–170.

[4] Ausiello, G., D'Atri, A., and Saccà, D. Graph Algorithms for Functional Dependency Manipulation. *J. ACM 30*, 4 (Oct. 1983), 752–766.

[5] Ausiello, G., D'Atri, A., and Saccá, D. Minimal Representation of Directed Hypergraphs. *SIAM J. Comput. 15*, 2 (May 1986), 418–431.

[6] Ausiello, G., and Laura, L. Directed hypergraphs: Introduction and fundamental algorithms—A survey. *Theoretical Computer Science 658*, Part B (2017), 293 – 306.

[7] B. Ganter, S. O. *Conceptual Exploration*. Springer, 2016.

[8] Bazhanov, K., and Obiedkov, S. Optimizations in computing the Duquenne–Guigues basis of implications. *Annals of Mathematics and Artificial Intelligence 70*, 1-2 (Feb. 2014), 5–24.

[9] Beeri, C., and Bernstein, P. A. Computational Problems Related to the Design of Normal Form Relational Schemas. *ACM Trans. Database Syst. 4*, 1 (Mar. 1979), 30–59.

[10] Bertet, K., Demko, C., Viaud, J.-F., and Guérin, C. Lattices, closures systems and implication bases: A survey of structural aspects and algorithms. *Theoretical Computer Science* (Nov. 2016).

[11] Boros, E., Čepek, O., and Kogan, A. Horn minimization by iterative decomposition. *Annals of Mathematics and Artificial Intelligence 23*, 3-4 (Nov. 1998), 321–343.

[12] Boros, E., Čepek, O., Kogan, A., and Kučera, P. Exclusive and essential sets of implicates of Boolean functions. *Discrete Applied Mathematics 158*, 2 (2010), 81 – 96.

[13] Boros, E., Čepek, O., and Makino, K. Strong Duality in Horn Minimization. In *Fundamentals of Computation Theory* (Berlin, Heidelberg, 2017), R. Klasing and M. Zeitoun, Eds., Springer Berlin Heidelberg, pp. 123–135.

[14] Bérczi, K., and Bérczi-Kovács, E. R. Directed hypergraphs and Horn minimization. *Information Processing Letters 128* (2017), 32 – 37.

[15] Cori, R., and Lascar, D. *Mathematical Logic: Part 1: Propositional Calculus, Boolean Algebras, Predicate Calculus, Completeness Theorems*. OUP Oxford, Sept. 2000. Google-Books-ID: Cle6_dOLt2IC.

[16] Davey, B. A., and Priestley, H. A. *Introduction to Lattices and Order*. Cambridge University Press, 2002.

[17] David, M. Minimum Covers in Relational Database Model. *J. ACM 27*, 4 (1980), 664 – 674.

[18] Day, A. The Lattice Theory of Functional Dependencies and Normal Decompositions. *International Journal of Algebra and Computation* (1992).

[19] Duquenne, V. Some variations on Alan Day's algorithm for calculating canonical basis of implications. In *Concept Lattices and their Applications (CLA)* (Montpellier, France, 2007), pp. 17–25.

[20] Ganter, B. Two Basic Algorithms in Concept Analysis. In *Formal Concept Analysis* (Mar. 2010), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 312–340.

[21] Ganter, B., and Wille, R. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag, Berlin Heidelberg, 1999.

[22] Guigues J.L, D. V. Familles minimales d'implications informatives résultant d'un tableau de données binaires. *Mathématiques et Sciences Humaines 95* (1986), 5–18.

[23] Hammer, P. L., and Kogan, A. Optimal compression of propositional Horn knowledge bases: complexity and approximation. *Artificial Intelligence 64*, 1 (Nov. 1993), 131–145.

[24] Maier, D. *Theory of Relational Databases*. Computer Science Pr, 1983.

[25] Shock, R. C. Computing the minimum cover of functional dependencies. *Information Processing Letters 22*, 3 (Mar. 1986), 157–159.

[26] Wild, M. Implicational bases for finite closure systems. *Informatik-Bericht 89/3, Institut fuer Informatik* (Jan. 1989).

[27] Wild, M. A Theory of Finite Closure Spaces Based on Implications. *Advances in Mathematics 108*, 1 (Sept. 1994), 118–139.

[28] Wild, M. Computations with finite closure systems and implications. In *Computing and Combinatorics* (Aug. 1995), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 111–120.

[29] Wild, M. The joy of implications, aka pure Horn formulas: Mainly a survey. *Theoretical Computer Science 658* (2017), 264 – 292.