Theory and implementation of efficient canonical systems for sentential calculus, based on Prime Normal Forms

www.bucephalus.org

first version May 1999, revised version September 2002

Abstract

A literal conjunction $\gamma = [\wedge\lambda_1...\lambda_m]$ is a *prime factor* of a *Disjunctive Normal Form (DNF)* $\Delta = [\vee[\wedge\lambda_{11}...\lambda_{1m_1}]...[\wedge\lambda_{n1}...\lambda_{nm_n}]]$ iff $\gamma \Rightarrow \Delta$ and none of the $\lambda_1, ..., \lambda_m$ could be deleted without violating $\gamma \Rightarrow \Delta$. A DNF $\Delta = [\vee\gamma_1...\gamma_n]$ is a *prime* DNF (*PDNF*) iff $\{\gamma_1, ..., \gamma_n\}$ is the set of all its prime factors. Every sentential formula has exactly one equivalent PDNF so that this defines a *canonic* representation which is distinguished from other possible canonizations. PDNF's can be constructed with the well–known *Quine–McCluskey method*, but that implementation is of exponential complexity and thus not practical in general. Therefore an efficient algorithm is designed in this paper. A dual system for *prime conjunctive normal forms (PCNF's)* is presented as well. It is shown how the whole sentential calculus, all junctions and semantic decisions, is effectively handled by these two systems.

Contents

1 Introduction: Normal Forms

Formal expressions, and logical expressions in particular, can be very different but still denote the same.
For instance
- "5/2" and "2.5"
- "$\neg\forall x(f(x) \neq g(x))$" and "$\exists x(f(x) = g(x))$"

These examples are quite simple, but this ambiguity is a complex and important problem in general. Algebra, sometimes even the whole of mathematics is often said to be the reasoning about the equivalence of expressions. What is always desired in particular is a transformation of "difficult", "incomprehensible" forms into "simple", "compact", and "understandable" or "standard" forms. In logic we talk about "normal forms".

Definition

Let F be the set of all formulas (of a particular language) and $N \subseteq F$. If there is for every $\varphi \in F$ a $\nu \in N$ such that $\varphi \Leftrightarrow \nu$ (i.e. φ and ν are equivalent), then N is called a set of **normal forms**.
Every function $nf : F \longrightarrow N$ with $nf(\varphi) \Leftrightarrow \varphi$ for every $\varphi \in F$ is called a (N–)**normalizer** (on F) and $nf(\varphi)$ is a (N–)**normalform** of φ.

Most of the time there are several possibilities to define standard or normal forms. The context then determines which of the definitions is preferred. The decimal notation for numbers has a long tradition in daily life, but in computer science it is more practical to use the dual representation.

There is a series of criteria to describe and compare the qualities of normal forms:
- **Intuitivity**: How comprehensive and readable are the formulas?
- **Canonicity**: How much do syntactic and semantic similarities coincide?
  Definition
  A set N of normal forms is **canonic**, if for all $\nu_1, \nu_2 \in N$: $\nu_1 \Leftrightarrow \nu_2$ iff $\nu_1 = \nu_2$.
  Accordingly a normalizer $nf : F \longrightarrow N$ is said to be **canonic**, if $nf(\varphi_1) = nf(\varphi_2)$ for all $\varphi_1, \varphi_2 \in F$ with $\varphi_1 \Leftrightarrow \varphi_2$.

  If a definition of canonic normal forms is possible, the whole semantics of a language can be defined immediately as well: the normal form then *is* the meaning of an expression or a formula.
- **Complexity of the representation**: How big are the normal forms?
- **Computational complexity**: How high is the effort required for the normalizer and how do the normal forms behave in additional operations?
- **Conservativity** of the normalizer: If φ is a normal form already, is then $nf(\varphi) = \varphi$? This would contribute to the readability.
- **Realizability** as technical devices: Are the occuring junctors implementable? What is the behaviour of the divice in time? etc.

There is quite a number of possibilities to define normal forms in sentential logic. Most often the so–called "disjunctive" and the "conjunctive normal forms" (DNF's and CNF's) are used.
Arguments for their superiority are:
- Their intuitivity:
  - CNF's are like a collection of elementary laws: Axiom systems, deductive systems are essentially CNF's.
  - DNF's are like a collection of elementary possibilities, similar to catalogues or inductive systems.
- As such they can often be modified by just adding or deleting components.
- Their similarity with arithmetics (Boole).

But DNF's and CNF's are not canonic already. Interestingly they are established as standards in sentential logic, but there are hardly any definitions of canonic DNF's and CNF's, not to mention any established standard.[1] I suggest, the reason lies partly in the inner algebraic structure of sentential logic itself, which, at first sight, seems to refuse a canonization that would be superior in all quality criterias: The smallest DNF's (or CNF's) mostly look a little bit different to the, in some respect, most comprehensible[2] DNF's etc. Another reason might be, that potential standards are easily defined, but can be constructed only with considerable efforts.

The goal of this paper is first to show, that there is in fact only just one (reasonable) canonization of the

[1] In fact there are the often so–called "canonic" DNF's, that consist of all their maximal literal conjunctions. For instance, the "canonic DNF" of $[A \rightarrow B]$ would be $[\vee[\wedge\overline{A}\overline{B}][\wedge\overline{A}B][\wedge AB]]$. The advantage of this form is that sentential logic turns into set theory in this way: conjunction, disjunction and negation of formulas become the intersection, union, and complementation of sets of maximal literal conjunctions. But these forms, except for simple cases, cannot be handled in practice because of the exponential demands of their representation. (Besides, strictly spoken these forms are not even canonic. For instance, $[\vee[\wedge]]$ and $[\vee[\wedge A][\wedge\overline{A}]]$ are both "canonic", but they are equivalent and yet not equal.)

[2] see the chapter on subvalence and covering

DNF's and CNF's, a "natural canonic" so to say, that we will call "(ordered) prime disjunctive/conjunctive normal forms". It doesn't need new theorems and proofs to demonstrate that, one must only consequently ask the question for a canonic. The answer then is a proposal for a standardization of sentential logic. Really new, I think, is the method that constructs all the prime factors of any given formula. Different to the common method of Quine–McCluskey our algorithm is not of exponential, but of polynomial complexity[3], so that non–trivial applications can be mastered as well. In this way the proposed standardization of the propositional calculus is not only theoretically, but also practically well founded.

The intention is to equally satisfy two kinds of readers: the specialist in Boolean algebras, who is investigating the possibility of "factorizing" the elements of these structures similar to the factorization in integer domains. Secondly the computer scientist who needs to build fast systems based on the sentential calculus and is looking for the quickest algorithms that can be implemented into sequential machines without further modification or refinement and altogether enable the full range of logical operations (i.e. junctions and semantic decisions). Because these two interests are quite conflicting, we separated the implementations from the algebraic core: the chapters that have " * Implementation: ..." in their title can be seen as appendices and should be skipped.

[3] [Comment September 2002: At the time of the first publication I didn't bother with complexity issues and I only mentioned them in footnotes. But obviously, it was a very naive thing to claim polynomial complexity without proving it. A paper about this neglected aspect is currently in progress.]

2 Basic concepts of sentential logic, DNF's and CNF's

Definition

From now on $\mathcal{A}$ shall be a fixed set of **atoms**. (For example the set of all non–empty strings made of letters and digits.)

We assume that there is a linear order $\leq_{\mathcal{A}}$ defined on $\mathcal{A}$. (For example, the lexical order).

We also write for $\alpha_1, \alpha_2 \in \mathcal{A}$

$$\alpha_1 <_{\mathcal{A}} \alpha_2 \text{ iff } (\alpha_1 \leq_{\mathcal{A}} \alpha_2 \text{ and } \alpha_1 \neq \alpha_2)$$

(Most of the time we will omit the subscript $\mathcal{A}$ and simply write $\leq$ and $<$.)

Definition

The set of all **formulas** (**on** $\mathcal{A}$) is recursively defined as follows:[4]

- Every $\alpha \in \mathcal{A}$ is an (**atomic**) formula.
- 0 and 1 are formulas.
- $\neg\varphi$ the **negation** of φ is a formula for every formula φ.
- If $n \geq 0$ and $\varphi_1, ..., \varphi_n$ are formulas, so are[5]
  $[\wedge\varphi_1...\varphi_n]$ the **conjunction** of $\varphi_1, ..., \varphi_n$
  $[\vee\varphi_1...\varphi_n]$ the **disjunction** of $\varphi_1, ..., \varphi_n$
  $[\overline{\wedge}\varphi_1...\varphi_n]$ the **nand-junction** of $\varphi_1, ..., \varphi_n$
  $[\overline{\vee}\varphi_1...\varphi_n]$ the **nor-junction** of $\varphi_1, ..., \varphi_n$.
- For formulas φ_1 and φ_2 the following expressions are formulas:
  $[\varphi_1 \rightarrow \varphi_2]$ the **subjunction** of φ_1 and φ_2
  $[\varphi_1 \leftrightarrow \varphi_2]$ the **equijunction** of φ_1 and φ_2

Definition

In formulas of the form $[*\varphi_1...\varphi_n]$ we call n the **arity** of **length** and the $\varphi_1, ..., \varphi_n$ the **components** of the junction.

$At(\varphi)$ denotes the set of all atoms occuring in a formula φ.

$=$ denotes the syntactic identity of two formulas.

Examples

If $\varphi = [[\wedge A\neg B] \rightarrow [A \leftrightarrow C]]$ then $At(\varphi) = \{A, B, C\}$.

$[\wedge AB] = [\wedge AB]$ but $[\wedge AB] \neq [\wedge BA]$

Definition

Let $* \in \{\wedge, \vee, \overline{\wedge}, \overline{\vee}\}$, φ' be a formula and $\varphi = [*\varphi_1...\varphi_n]$ and $\psi = [*\psi_1...\psi_m]$ be formulas. We write

- $\varphi' \in \varphi$, φ is an **element** or **component** of φ, iff $\varphi' \in \{\varphi_1, ..., \varphi_n\}$.
- $\varphi' \notin \varphi$, iff φ' is not an element of φ.
- $\varphi \subseteq \psi$, φ is **contained** in ψ, iff $\{\varphi_1, ..., \varphi_n\} \subseteq \{\psi_1, ..., \psi_m\}$.
- $\varphi \not\subseteq \psi$ iff φ is not contained in ψ.
- $\varphi \subset \psi$, φ is **properly contained** in ψ, iff $\varphi \subseteq \psi$ and $\psi \not\subseteq \varphi$.
- $\varphi \not\subset \psi$ iff φ is not properly contained in ψ.

Definition

The order $\Rightarrow$ and the equivalence relation $\Leftrightarrow$ on the set of formulas are defined as usual. We will only submit the notations and terminology. For any two formulas φ_1 and φ_2

- $\varphi_1 \Rightarrow \varphi_2$ means φ_1 is **subvalent** to φ_2 or φ_2 **follows from** φ_1. In that case we say that φ_1 is an **implicand** of φ_2 and that φ_2 is a **consequence** of φ_2.
- $\varphi_1 \not\Rightarrow \varphi_2$ denotes that φ_1 is not subvalent to φ_2.
- $\varphi_1 \Leftrightarrow \varphi_2$ means that φ_1 and φ_2 are **equivalent**.
- $\varphi_1 \not\Leftrightarrow \varphi_2$ denotes that φ_1 and φ_2 are not equivalent.

Furthermore we write

- $\varphi_1 \dot{\Rightarrow} \varphi_2$, φ_1 is **properly subvalent** to φ_2, iff $\varphi_1 \Rightarrow \varphi_2$ and $\varphi_1 \not\Leftrightarrow \varphi_2$.

As usual, we call a formula φ

- **valid** or a **tautology**, iff $\varphi \Leftrightarrow 1$
- **satisfiable** iff $\varphi \not\Leftrightarrow 0$
- a **contradiction** iff $\varphi \Leftrightarrow 0$

Definition

- A **literal** is either an atom (**positive** literal) or a negated atom (**negative** literal).
- For a literal λ, its **complement** is defined as $\overline{\lambda} := \begin{cases} \alpha & \text{if } \lambda = \neg\alpha \text{ is a negative literal} \\ \neg\alpha & \text{if } \lambda = \alpha \text{ is a positive literal} \end{cases}$
- Two literals λ_1 and λ_2 are called **complementary**, iff $\lambda_1 = \overline{\lambda_2}$.
- $|\lambda|$ denotes the atom that occurs in a literal λ.

Definition

- If $\lambda_1, ..., \lambda_n$ are literals, $n \geq 0$, we call
  $[\wedge\lambda_1...\lambda_n]$ a **literal conjunction**, and
  $[\vee\lambda_1...\lambda_n]$ a **literal disjunction**.
- A literal conjunction $[\wedge\lambda_1...\lambda_n]$ is called a **normal literal conjunction** or **NLC** iff $|\lambda_1| < ... < |\lambda_n|$.
- A literal disjunction $[\vee\lambda_1...\lambda_n]$ is called a **normal**

[4] We shall see later in our implementations that our system is very flexible in changing the set of junctors. For every new junctor we only need to add two lines in the algorithm dnf (and cnf).

[5] If there is a danger of confusion, blanks will be inserted between the components of these junctions, and we would for example write $[\wedge \quad \varphi_1 \quad \varphi_2 \quad \varphi_3]$ instead of $[\wedge\varphi_1\varphi_2\varphi_3]$

literal disjunction or **NLD** iff $|\lambda_1| < ... < |\lambda_n|$.[6]

Theorem
- A literal conjunction is a contradiction if and only if it contains a pair of complementary literals.
- A literal disjunction is a tautology if and only if it contains a pair of complementary literals.

And since neither NLC's nor NLC's contain any pair of literals with the same element, we immediately get the following

Theorem
- $0 \dot{\Rightarrow} \gamma$ for each NLC γ.
- $\delta \dot{\Rightarrow} 1$ for each NLD δ.

Theorem
- For any two NLC's γ_1 and γ_2
  - $\gamma_1 \Rightarrow \gamma_2$ iff $\gamma_2 \subseteq \gamma_1$
  - $\gamma_1 \dot{\Rightarrow} \gamma_2$ iff $\gamma_2 \subset \gamma_1$
  - $\gamma_1 \Leftrightarrow \gamma_2$ iff $\gamma_1 = \gamma_2$
- For any two NLC's δ_1 and δ_2
  - $\delta_1 \Rightarrow \delta_2$ iff $\delta_1 \subseteq \delta_2$
  - $\delta_1 \dot{\Rightarrow} \delta_2$ iff $\delta_1 \subset \delta_2$
  - $\delta_1 \Leftrightarrow \delta_2$ iff $\delta_1 = \delta_2$

There are infinitely many literal conjunctions equivalent to a given literal conjunction γ. But only one of them is normal. (The same holds for literal disjunctions.)

There are different possibilities to define a reasonable linear order on the set of NLC's and NLD's. We will need this order(s) only for the reason of completion (see the notes on "canonicity" further on), we don't employ it in the main part of our work. We shall choose the usual lexical order (with the convention, that a positive is lexically smaller than the complementary negative literal). The reader may modify the choice as pleased.

Definition
With $* \in \{\wedge, \vee\}$ let $[*\lambda_1...\lambda_n]$ and $[*\lambda'_1...\lambda'_m]$ be either two NLC's or NLD's. We define

$$[*\lambda_1...\lambda_n] \leq [*\lambda'_1...\lambda'_m]$$
$$\text{iff } n = 0$$
$$\text{or } |\lambda_1| < |\lambda'_1|$$
$$\text{or } \neg\lambda_1 = \lambda'_1$$
$$\text{or } (\lambda_1 = \lambda'_1 \text{ and } [*\lambda_2...\lambda_n] \leq [*\lambda'_2...\lambda'_m]).$$

Otherwise, we also write $[*\lambda_1...\lambda_n] \not\leq [*\lambda'_1...\lambda'_m]$.

Definition
- If $\gamma_1, ..., \gamma_n$ are NLC's with $n \geq 0$, then $[\vee\gamma_1...\gamma_n]$ is called a **disjunctive normal form** or **DNF**.
- If $\delta_1, ..., \delta_n$ are NLD's with $n \geq 0$, then $[\wedge\delta_1...\delta_n]$ is called a **conjunctive normal form** or **CNF**.

Definition
- A DNF $[\vee\gamma_1...\gamma_n]$ is said to be **ordered** iff $\gamma_1 \leq ... \leq \gamma_n$.
- A CNF $[\wedge\delta_1...\delta_n]$ is said to be **ordered** iff $\delta_1 \leq ... \leq \delta_n$.

(If $n = 0$ or $n = 1$, the forms are ordered by definition.)

At first sight our definition of DNF's (and CNF's) might seem inconsequent, because
- either we follow the common convention and don't demand any syntactic order at all: A "DNF" would then be a disjunction of literal conjunctions
- or we consequently approach canonicity and demand a full order inside the forms: A "DNF" would then be our ordered DNF.

The main reason for our definition is the fact that it allows us to implement the fastest algorithms. To demand that every literal conjunction must be normal is certainly no loss for the readability of the form and it immediately speeds up the processes. To demand that the NLC's must be ordered in a sequence would on the other hand just slow down everything. For this reason we will soon even weaken the concept of canonicity for CNF's and DNF's a little bit.

Theorem
- For every DNF $\Delta = [\vee\gamma_1...\gamma_n]$ holds $\Delta \Leftrightarrow 0$ iff $n = 0$.
- For every CNF $\Gamma = [\wedge\delta_1...\delta_n]$ holds $\Gamma \Leftrightarrow 1$ iff $n = 0$.

Proof
Because $0 \dot{\Rightarrow} \gamma$ for every NLC γ and $[\vee\varphi_1...\varphi_n] \Rightarrow [\vee\varphi_1...\varphi_n\gamma]$ for every $n \geq 0$ and all formulas $\varphi_1, ..., \varphi_n$, there is $0 \dot{\Rightarrow} \Delta$ for every DNF $\Delta = [\vee\gamma_1...\gamma_n]$ with $n > 0$.
The dual proof holds for CNF's.

Theorem
Every formula φ has an (ordered) DNF as well as an (ordered) CNF.

[6] Instead of being *one* list of *literals*, NLC's (and NLD's) can be implemented as *two* lists of *atoms*, according to the transformation $[\wedge\overline{A}BC\overline{DE}F] \Leftrightarrow [\wedge[\wedge BCF][\overline{\vee}ADE]]$. Since the atom is the basic unit of sentential logic, one could dispense all junctors with this implementation. The whole system could be managed only by using the data structures *atom* and *list*. DNF's (and CNF's) would then be lists of pairs of ordered atom lists. But in practice it is actually more common to take the literal and not the atom as the basis unit. We won't discuss the arguments for doing so.

This is well known and doesn't need a proof here. Instead we present two sets of algorithms (the "DNF system" and the "CNF system") that include a normalizer each (namely "dnf" and "cnf").

3 * Implementation: The DNF and the CNF system

The following DNF and CNF system are some kind of prototype for the final PDNF and PCNF system (see last chapter).
Their central functions are the normalizers *dnf* and *cnf*. These are recursively implemented by making exhaustive use of de Morgans law and the junctor transformations into the $\wedge$-$\vee$-$\neg$ system, so that only the disjunctions and conjunctions of DNF's and CNF's need to be specified furthermore. These could be done exclusively by using the distributive laws and the concatenation of conjunctions and disjunctions. But as we insist on the "normality" of literal conjunctions and disjunctions, the two functions *nlc-and-nlc* for the conjunction of two NLC's and *nld-or-nlc* for the disjunction of two NLD's must not be implemented as concatenations, but need a more complex treatment. This is done by the auxiliary algorithm *jrec*.
jrec (the *rec*ursive implementation of these *j*unctions) merges its first two parameters stepwise into its third argument, which is returned in the end. We make use of the duality principle and therefore we can use it both for *nlc-and-nlc* and *nld-or-nld*.

The DNF system: Declarations

The Normalizer

dnf (φ)
is a DNF of φ, for every formula φ.

Conjunctions and Disjunctions

nlc-and-nlc (γ_1, γ_2)
is a DNF of $[\wedge\gamma_1\gamma_2]$, for every two NLC's γ_1 and γ_2.

nlc-or-nlc (γ_1, γ_2)
is a DNF for $[\vee\gamma_1\gamma_2]$, for every two NLC's γ_1 and γ_2.

nlc-and-dnf (γ, Δ)
is a DNF for $[\wedge\gamma\Delta]$, for every NLC γ and DNF Δ.

nlc-or-dnf (γ, Δ)
is a DNF for $[\vee\gamma\Delta]$, for every NLC γ and DNF Δ.

dnf-and-dnf (Δ_1, Δ_2)
is a DNF for $[\wedge\Delta_1\Delta_2]$, for every two DNF's Δ_1 and Δ_2.

dnf-or-dnf (Δ_1, Δ_2)
is a DNF for $[\vee\Delta_1\Delta_2]$, for every two DNF's Δ_1 and Δ_2.

dnf-conj $((\Delta_1, \ldots, \Delta_n))$
is a DNF for $[\wedge\Delta_1 \ldots \Delta_n]$, for every list $(\Delta_1, \ldots, \Delta_n)$ of DNF's with $n \geq 0$.

dnf-disj $((\Delta_1, \ldots, \Delta_n))$
is a DNF for $[\vee\Delta_1 \ldots \Delta_n]$, for every list $(\Delta_1, \ldots, \Delta_n)$ of DNF's with $n \geq 0$.

The CNF system: Declarations

The Normalizer

dnf (φ)
is a CNF of φ, for every formula φ.

Conjunctions and Disjunctions

nld-or-nld (δ_1, δ_2)
is a CNF of $[\vee\delta_1\delta_2]$, for every two NLD's δ_1 and δ_2.

nld-and-nld (δ_1, δ_2)
is a CNF for $[\wedge\delta_1\delta_2]$, for every two NLD's δ_1 and δ_2.

nld-or-cnf (δ, Γ)
is a CNF for $[\vee\delta\Gamma]$, for every NLD δ and CNF Γ.

nld-and-cnf (δ, Γ)
is a CNF for $[\wedge\delta\Gamma]$, for every NLD δ and CNF Γ.

cnf-or-cnf (Γ_1, Γ_2)
is a CNF for $[\vee\Gamma_1\Gamma_2]$, for every two CNF's Γ_1 and Γ_2.

cnf-and-cnf (Γ_1, Γ_2)
is a CNF for $[\wedge\Gamma_1\Gamma_2]$, for every two CNF's Γ_1 and Γ_2.

cnf-disj $((\Gamma_1, \ldots, \Gamma_n))$
is a CNF for $[\vee\Gamma_1 \ldots \Gamma_n]$, for every list $(\Gamma_1, \ldots, \Gamma_n)$ of CNF's with $n \geq 0$.

cnf-conj $((\Gamma_1, \ldots, \Gamma_n))$
is a CNF for $[\wedge\Gamma_1 \ldots \Gamma_n]$, for every list $(\Gamma_1, \ldots, \Gamma_n)$ of CNF's with $n \geq 0$.

The DNF system: Implementation

dnf (φ)

$:=$

$[\vee[\wedge\varphi]]$	if φ is an atom
$[\vee]$	if $\varphi = 0$
$[\vee[\wedge]]$	if $\varphi = 1$
$[\vee[\wedge\neg\alpha]]$	if $\varphi = \neg\alpha$ and α is atom
$[\vee[\wedge]]$	if $\varphi = \neg 0$
$[\vee]$	if $\varphi = \neg 1$
dnf (φ')	if $\varphi = \neg\neg\varphi'$
dnf-disj $((dnf\ (\neg\varphi_1), \ldots, dnf\ (\neg\varphi_n)))$	if $\varphi = \neg[\wedge\varphi_1 \ldots \varphi_n]$
dnf-conj $((dnf\ (\neg\varphi_1), \ldots, dnf\ (\neg\varphi_n)))$	if $\varphi = \neg[\vee\varphi_1 \ldots \varphi_n]$
dnf-conj $((dnf\ (\varphi_1), \ldots, dnf\ (\varphi_n)))$	if $\varphi = \neg[\overline{\wedge}\varphi_1 \ldots \varphi_n]$
dnf-disj $((dnf\ (\varphi_1), \ldots, dnf\ (\varphi_n)))$	if $\varphi = \neg[\overline{\vee}\varphi_1 \ldots \varphi_n]$
dnf-and-dnf $(dnf\ (\varphi_1), dnf\ (\neg\varphi_2))$	if $\varphi = \neg[\varphi_1 \to \varphi_2]$
dnf-or-dnf (*dnf-and-dnf* $(dnf\ (\varphi_1), dnf\ (\neg\varphi_2))$, *dnf-and-dnf* $(dnf\ (\neg\varphi_1), dnf\ (\varphi_2))$)	if $\varphi = \neg[\varphi_1 \leftrightarrow \varphi_2]$
dnf-conj $(dnf\ (\varphi_1), \ldots, dnf\ (\varphi_n))$	if $\varphi = [\wedge\varphi_1 \ldots \varphi_n]$
dnf-disj $(dnf\ (\varphi_1), \ldots, dnf\ (\varphi_n))$	if $\varphi = [\vee\varphi_1 \ldots \varphi_n]$
dnf-disj $(dnf\ (\neg\varphi_1), \ldots, dnf\ (\neg\varphi_n))$	if $\varphi = [\overline{\wedge}\varphi_1 \ldots \varphi_n]$
dnf-conj $(dnf\ (\neg\varphi_1), \ldots, dnf\ (\neg\varphi_n))$	if $\varphi = [\overline{\vee}\varphi_1 \ldots \varphi_n]$
dnf-or-dnf $(dnf\ (\neg\varphi_1), dnf\ (\varphi_2))$	if $\varphi = [\varphi_1 \to \varphi_2]$
dnf-or-dnf (*dnf-and-dnf* $(dnf\ (\varphi_1), dnf\ (\varphi_2))$ *dnf-and-dnf* $(dnf\ (\neg\varphi_1), dnf\ (\neg\varphi_2))$)	if $\varphi = [\varphi_1 \leftrightarrow \varphi_2]$

nlc-and-nlc (γ_1, γ_2)
$:=$ *jrec* $(\gamma_1, \gamma_2, [\wedge])$

The CNF system: Implementation

cnf (φ)

$:=$

$[\wedge[\vee\varphi]]$	if φ is an atom
$[\wedge[\vee]]$	if $\varphi = 0$
$[\wedge]$	if $\varphi = 1$
$[\wedge[\vee\neg\alpha]]$	if $\varphi = \neg\alpha$ and α is atom
$[\wedge]$	if $\varphi = \neg 0$
$[\wedge[\vee]]$	if $\varphi = \neg 1$
cnf (φ')	if $\varphi = \neg\neg\varphi'$
cnf-disj $((cnf\ (\neg\varphi_1), \ldots, cnf\ (\neg\varphi_n)))$	if $\varphi = \neg[\wedge\varphi_1 \ldots \varphi_n]$
cnf-conj $((cnf\ (\neg\varphi_1), \ldots, cnf\ (\neg\varphi_n)))$	if $\varphi = \neg[\vee\varphi_1 \ldots \varphi_n]$
cnf-conj $((cnf\ (\varphi_1), \ldots, cnf\ (\varphi_n)))$	if $\varphi = \neg[\overline{\wedge}\varphi_1 \ldots \varphi_n]$
cnf-disj $((cnf\ (\varphi_1), \ldots, cnf\ (\varphi_n)))$	if $\varphi = \neg[\overline{\vee}\varphi_1 \ldots \varphi_n]$
cnf-and-cnf $(cnf\ (\varphi_1), cnf\ (\neg\varphi_2))$	if $\varphi = \neg[\varphi_1 \to \varphi_2]$
cnf-or-cnf (*cnf-and-cnf* $(cnf\ (\varphi_1), cnf\ (\neg\varphi_2))$, *cnf-and-cnf* $(cnf\ (\neg\varphi_1), cnf\ (\varphi_2))$)	if $\varphi = \neg[\varphi_1 \leftrightarrow \varphi_2]$
cnf-conj $(cnf\ (\varphi_1), \ldots, cnf\ (\varphi_n))$	if $\varphi = [\wedge\varphi_1 \ldots \varphi_n]$
cnf-disj $(cnf\ (\varphi_1), \ldots, cnf\ (\varphi_n))$	if $\varphi = [\vee\varphi_1 \ldots \varphi_n]$
cnf-disj $(cnf\ (\neg\varphi_1), \ldots, cnf\ (\neg\varphi_n))$	if $\varphi = [\overline{\wedge}\varphi_1 \ldots \varphi_n]$
cnf-conj $(cnf\ (\neg\varphi_1), \ldots, cnf\ (\neg\varphi_n))$	if $\varphi = [\overline{\vee}\varphi_1 \ldots \varphi_n]$
cnf-or-cnf $(cnf\ (\neg\varphi_1) cnf\ (\varphi_2))$	if $\varphi = [\varphi_1 \to \varphi_2]$
cnf-or-cnf (*cnf-and-cnf* $(cnf\ (\varphi_1), cnf\ (\varphi_2))$ *cnf-and-cnf* $(cnf\ (\neg\varphi_1), cnf\ (\neg\varphi_2)))$	if $\varphi = [\varphi_1 \leftrightarrow \varphi_2]$

nld-or-nld (δ_1, δ_2)
$:=$ *jrec* $(\delta_1, \delta_2, [\vee])$

nlc-or-nlc (γ_1, γ_2)
$:= [\vee \gamma_1 \gamma_2]$

nlc-and-dnf $(\gamma, [\vee \gamma_1 \ldots \gamma_n])$
$:=$ *dnf-disj* $((\textit{nlc-and-nlc}\ (\gamma, \gamma_1), \ldots, \textit{nlc-and-nlc}\ (\gamma, \gamma_n)))$

nlc-or-dnf $(\gamma, [\vee \gamma_1 \ldots \gamma_n])$
$:= [\vee \gamma \gamma_1 \ldots \gamma_n]$

dnf-and-dnf $([\vee \gamma_1 \ldots \gamma_n], \Delta)$
$:=$ *dnf-disj* $((\textit{nlc-and-dnf}\ (\gamma_1, \Delta), \ldots, \textit{nlc-and-dnf}\ (\gamma_n, \Delta)))$

dnf-or-dnf $([\vee \gamma_1 \ldots \gamma_n], [\vee \gamma'_1 \ldots \gamma'_m])$
$:= [\vee \gamma_1 \ldots \gamma_n \gamma'_1 \ldots \gamma'_m]$

dnf-conj $((\Delta_1, \ldots, \Delta_n))$

$$:= \begin{cases} [\vee[\wedge]] & \text{if } n = 0 \\ \Delta_1 & \text{if } n = 1 \\ \textit{dnf-and-dnf}\ (\Delta_1, \textit{dnf-conj}\ ((\Delta_2, \ldots, \Delta_n))) & \text{if } n > 1 \end{cases}$$

dnf-disj $((\Delta_1, \ldots, \Delta_n))$

$$:= \begin{cases} [\vee] & \text{if } n = 0 \\ \Delta_1 & \text{if } n = 1 \\ \textit{dnf-or-dnf}\ (\Delta_1, \textit{dnf-disj}\ ((\Delta_2, \ldots, \Delta_n))) & \text{if } n > 1 \end{cases}$$

nld-and-nld (δ_1, δ_2)
$:= [\wedge \delta_1 \delta_2]$

nld-or-cnf $(\delta, [\wedge \delta_1 \ldots \delta_n])$
$:=$ *cnf-conj* $((\textit{nld-or-nld}\ (\delta, \delta_1), \ldots, \textit{nld-or-nld}\ (\delta, \delta_n)))$

nld-and-cnf $(\delta, [\wedge \delta_1 \ldots \delta_n])$
$:= [\wedge \delta \delta_1 \ldots \delta_n]$

cnf-or-cnf $([\wedge \delta_1 \ldots \delta_n], \Gamma)$
$:=$ *cnf-conj* $((\textit{nld-or-cnf}\ (\delta_1, \Gamma), \ldots, \textit{nld-or-cnf}\ (\delta_n, \Gamma)))$

cnf-and-cnf $([\wedge \delta_1 \ldots \delta_n], [\wedge \delta'_1 \ldots \delta'_m])$
$:= [\wedge \delta_1 \ldots \delta_n \delta'_1 \ldots \delta'_m]$

cnf-disj $((\Gamma_1, \ldots, \Gamma_n))$

$$:= \begin{cases} [\wedge[\vee]] & \text{if } n = 0 \\ \Gamma_1 & \text{if } n = 1 \\ \textit{cnf-or-cnf}\ (\Gamma_1, \textit{cnf-disj}\ ((\Gamma_2, \ldots, \Gamma_n))) & \text{if } n > 1 \end{cases}$$

cnf-conj $((\Gamma_1, \ldots, \Gamma_n))$

$$:= \begin{cases} [\wedge] & \text{if } n = 0 \\ \Gamma_1 & \text{if } n = 1 \\ \textit{cnf-and-cnf}\ (\Gamma_1, \textit{cnf-conj}\ ((\Gamma_2, \ldots \Gamma_n))) & \text{if } n > 1 \end{cases}$$

Auxiliary algorithms

$$\overline{*} := \left\{ \begin{array}{ll} \vee & \text{if } * = \wedge \\ \wedge & \text{if } * = \vee \end{array} \right\} \text{ for } * \in \{\wedge, \vee\}$$

jrec $([*\lambda_1 \ldots \lambda_n], [*\lambda'_1 \ldots \lambda'_m], [*\mu_1 \ldots \mu_k])$

$$:= \begin{cases} [\overline{*}[*\mu_1 \ldots \mu_k \lambda'_1 \ldots \lambda'_m]] & \text{if } n = 0 \\ [\overline{*}[*\mu_1 \ldots \mu_k \lambda_1 \ldots \lambda_n]] & \text{if } m = 0 \\ \textit{jrec}\ ([*\lambda_2 \ldots \lambda_n], [*\lambda'_1 \ldots \lambda'_m], [*\mu_1 \ldots \mu_k \lambda_1]) & \text{if } n > 0 \text{ and } m > 0 \text{ and } |\lambda_1| < |\lambda'_1| \\ \textit{jrec}\ ([*\lambda_1 \ldots \lambda_n], [*\lambda'_2 \ldots \lambda'_m], [*\mu_1 \ldots \mu_k \lambda'_1]) & \text{if } n > 0 \text{ and } m > 0 \text{ and } |\lambda'_1| < |\lambda_1| \\ \textit{jrec}\ ([*\lambda_2 \ldots \lambda_n], [*\lambda'_2 \ldots \lambda'_m], [*\mu_1 \ldots \mu_k \lambda_1]) & \text{if } n > 0 \text{ and } m > 0 \text{ and } \lambda_1 = \lambda'_1 \\ [\overline{*}] & \text{if } n > 0 \text{ and } m > 0 \text{ and } \overline{\lambda_1} = \lambda'_1 \end{cases}$$

4 Special DNF's and CNF's

Introduction

A major disadvantage of the DNF system[7] is the often unnecessary length of the normal forms which impairs the readability and increases the costs for its representations and further operations.
The following two examples are equivalent DNF's:
- $[\vee[\wedge\overline{ABC}][\wedge A\overline{B}][\wedge B][\wedge C][\wedge C]]$
- $[\vee[\wedge]]$

But the second form will certainly be prefered in all practical means.
In our search for a good canonic we will therefore start off with an attempt to find as "small" and "handy" forms as possible. But a remarkable and, on a first view may be even astonishing phenomenon of sentential logic is the fact, that here "small" and "handy" are quite close together, but nevertheless might mean significantly different forms in many cases.
In this chapter we will formally conceptualize the desired criteria and we will find three differend special kinds of DNF's:
- MDNF's or minimal DNF's,which are in fact the absolutely shortest DNF's.
- M2DNF's or pairwise minimal DNF's, where each two components are a 2–ary MDNF $[\vee\gamma_i\gamma_j]$.
- PDNF's or prime DNF's, where the components are exactly the set of all "prime factors".

As we said, these forms are quite often identical. But in fact they are different concepts and for the construction and comparison of DNF systems it is important to work out these differences and to analyse the properties of these forms.

Canonicity

First of all we are going to weaken the concept of canonicity, because we want to neglect the order of the components in the normal forms. It only obstructs the flow of our argumentation and unnecessarily increases the costs of the algorithms. Besides, normalizations that are not canonic in general, but still canonic in certain semantically distinguished cases, deserve an extra attention. For example a normalizer nf that is at least "1–canonic", provides us with a subvalence and equivalence decision, because

$$\varphi_1 \Rightarrow \varphi_2 \text{ iff } nf([\varphi_1 \rightarrow \varphi_2]) = nf(1).$$

These kind of normalizers might not define a whole semantic, but at least they enable the definitions of "tautology / contradiction" and "validity / satisfiability".

Definition
- Two DNF's $\Delta_1 = [\vee\gamma_1...\gamma_n]$ and $\Delta_2 = [\vee\gamma'_1...\gamma'_m]$ are said to be **identical up to the order of their components**, or **similar** in short, written $\Delta_1 \approx \Delta_2$, iff there is a bijection $i : \{1,...,n\} \longrightarrow \{1,...,m\}$ such that $[\vee\gamma_{i(1)}...\gamma_{i(n)}] = [\vee\gamma'_1...\gamma'_m]$.
- Two CNF's $\Gamma_1 = [\wedge\delta_1...\delta_n]$ and $\Gamma_2 = [\wedge\delta'_1...\delta'_m]$ are said to be **identical up to the order of their components**, or **similar** in short, written $\Gamma_1 \approx \Gamma_2$, iff there is a bijection $i : \{1,...,n\} \longrightarrow \{1,...,m\}$ such that $[\vee\delta_{i(1)}...\delta_{i(n)}] = [\vee\delta'_1...\delta'_m]$.

Definition

Let N be a set of DNF's (CNF's) that is itself a set of normal forms (i.e. for every formula there is an equivalent formula in N). We say that N is
- **canonic**, when for all $\nu_1, \nu_2 \in N$ $\nu_1 \Leftrightarrow \nu_2$ iff $\nu_1 = \nu_2$
- **pre-canonic**, when for all $\nu_1, \nu_2 \in N$ $\nu_1 \Leftrightarrow \nu_2$ iff $\nu_1 \approx \nu_2$
- 0–**canonic**, when there is only one contradiction in N
- 1–**canonic**, when there is only one tautology in N.

Accordingly a N–normalizer nf is called
- **canonic**, if $nf(\varphi_1) = nf(\varphi_2)$ for all formulas φ_1 and φ_2 with $\varphi_1 \Leftrightarrow \varphi_2$
- **pre-canonic**, if $nf(\varphi_1) \approx nf(\varphi_2)$ for all formulas φ_1 and φ_2 with $\varphi_1 \Leftrightarrow \varphi_2$
- 0–**canonic**, if $nf(\varphi) = nf(0)$ for all contradictory formulas φ.
- 1–**canonic**, if $nf(\varphi) = nf(1)$ for all contradictory formulas φ.

The following theorem immediately follows from this definition and it guarantees that a pre–canonic normalizer is actually as good as a canonic one.

Theorem

If nf is a pre–canonic normalizer and ord the function, that transforms an arbitrary formula into its ordered normal form, then $ord \circ nf$ is a canonic normalizer.[8]

Theorem

[7] The "principle of duality" guarantees that all true statements about DNF's turn into true statements about CNF's by inverting all occuring junctors. Therefore we will restrict ourselves mainly to DNF's in the following explanations and proofs.

[8] $ord \circ nf(\varphi) := ord(nf(\varphi))$

DNF's are
- neither canonic nor pre–canonic
- not 1–canonic
- 0–canonic

CNF's are
- neither canonic nor pre–canonic
- not 0–canonic
- 1–canonic

Proof
- $[\vee[\wedge A]]$ and $[\vee[\wedge A\overline{B}][\wedge AB]]$ are equivalent DNF's, but they are neither equal, nor similar.
- $[\vee[\wedge\overline{A}][\wedge A]] \Leftrightarrow [\vee[\wedge]] \Leftrightarrow 1$. Thus DNF's are not 1–canonic.
- According to a theorem in chapter 2 it holds for each DNF Δ that $\Delta \Leftrightarrow 0$ iff $\Delta = [\vee]$. Thus DNF's are 0–canonic.

Factors

Definition
- Let γ be a NLC and $\Delta = [\vee\gamma_1...\gamma_n]$ a DNF. γ is called a (**disjunctive**) **factor** or **implicand** of Δ, iff $\gamma \Rightarrow \Delta$. In other words, iff $\Delta \Leftrightarrow [\vee\gamma\gamma_1...\gamma_n]$. In particular each component $\gamma_i \in \Delta$ is a factor of Δ.
- Let δ be a NLD and $\Gamma = [\wedge\delta_1...\delta_n]$ a CNF. δ is called a (**conjunctive**) **factor** or **consequence** of Γ, iff $\Gamma \Rightarrow \delta$. In other words, iff $\Gamma \Leftrightarrow [\wedge\delta\delta_1...\delta_n]$. In particular each component $\delta_i \in \Gamma$ is a factor of Γ.

Definition
- A factor γ of a DNF Δ is **prime**, iff there is no factor γ' of Δ with $\gamma \Rightarrow \gamma'$. Thus "prime" means here not further reducible. No literal can be deleted in γ such that γ is still a factor of Δ.
- A factor δ of a CNF Γ is **prime**, iff there is no factor δ' of Γ with $\delta' \Rightarrow \delta$. Again "prime" means here not further reducible. For each NLD δ' the case $\delta' \subset \delta$ would imply that δ' is not a factor of Γ.

Remark
- $[\vee]$ is the only DNF that has no factors.
- $[\wedge]$ is the only CNF that has no factors.
- Every NLC γ is a factor of $[\vee[\wedge]]$.
- Every NLC δ is a factor of $[\wedge[\vee]]$.

Each DNF different to $[\vee]$ has infinitely many factors. (The same holds for CNF's.) This is because the DNF would then at least have one component $\gamma = [\wedge\lambda_1...\lambda_n]$. Every NLC of the form $\gamma' = [\wedge\lambda_1...\lambda_n\lambda_{n+1}...\lambda_{n+m}]$ is a factor of Δ too, since $\gamma' \Rightarrow \gamma \Rightarrow \Delta$.
But although the number of factors is (almost) always infinite, the number of prime factors is always finite, as follows directly from the following

Theorem
- For each prime factor γ of a DNF Δ holds: $At(\gamma) \subseteq At(\Delta)$.
- For each prime factor δ of a CNF Γ holds: $At(\delta) \subseteq At(\Gamma)$.

Proof
Again we only proof the first half of the theorem.
Let $\Gamma = [\wedge[\vee\lambda_{11}...\lambda_{1m_1}]...[\vee\lambda_{n1}...\lambda_{nm_n}]]$ be a CNF equivalent to Δ and let $\gamma = [\wedge\lambda_1...\lambda_l]$. It holds that γ is a factor of Δ iff $\gamma \Rightarrow \Delta$ iff $[\wedge\gamma\neg\Gamma] \Leftrightarrow 0$ iff $[\wedge\gamma[\vee[\wedge\overline{\lambda_{11}}...\overline{\lambda_{1m_1}}]...[\wedge\overline{\lambda_{n1}}...\overline{\lambda_{nm_n}}]]] \Leftrightarrow 0$ iff $[\vee[\wedge\lambda_1...\lambda_l\overline{\lambda_{11}}...\overline{\lambda_{1m_1}}]...[\wedge\lambda_1...\lambda_l\overline{\lambda_{n1}}...\overline{\lambda_{nm_n}}]] \Leftrightarrow 0$ iff $[\wedge\lambda_1...\lambda_l\overline{\lambda_{i1}}...\overline{\lambda_{im_i}}] \Leftrightarrow 0$ for each $i = 1,...,n$ iff $\{\lambda_1,...,\lambda_l\} \cap \{\lambda_{i1},...,\lambda_{im_i}\} \neq \{\}$ for each $i = 1,...,n$.
Suppose now $At(\gamma) \not\subseteq At(\Delta)$. So there must be a literal in γ — let it be λ_1 — such that $|\lambda_1| \notin At(\Delta)$. But in that case $\{\lambda_1,...,\lambda_l\} \cap \{\lambda_{i1},...,\lambda_{im_i}\} = \{\lambda_2,...,\lambda_l\} \cap \{\lambda_{i1},...,\lambda_{im_i}\}$ for each $i = 1,...,n$. And so $[\wedge\lambda_2...\lambda_l]$ would be a factor of Δ, too. But then γ wouldn't be a prime, and that would contradict the initial assumption.

Corollary
The number of prime factors of a given DNF or CNF is finite.

Prime Normal Forms

Definition
Suppose φ is a formula
- Let Δ be any arbitrary DNF of φ and $\{\gamma_1,...,\gamma_n\}$ the set of all prime factors of Δ. We call $\Delta' := [\vee\gamma_1...\gamma_n]$ a **prime disjunctive normal form** or **PDNF** (of φ or of Δ).
- Let Γ be any arbitrary CNF of φ and $\{\delta_1,...,\delta_n\}$ the set of all prime factors of Γ. We call $\Gamma' := [\wedge\delta_1...\delta_n]$ a **prime conjunctive normal form** or **PCNF** (of φ or of Γ).

Each formula φ has a DNF Δ. The set of all prime factor of Δ is always finite and so the PDNF Δ' is always well defined (up to the order of its components). Δ' is uniquely defined, no matter which DNF Δ of φ was chosen.
Furthermore $\Delta' \Leftrightarrow \Delta$, because:
- Each prime factor γ_i is a factor of Δ, thus $\gamma_i \Rightarrow \Delta$ and $\Delta' \Rightarrow \Delta$.
- Every component $\gamma \in \Delta$ has a prime component

$\gamma_i \in \Delta'$ such that $\gamma \Rightarrow \gamma_i$, and so $\Delta \Rightarrow \Delta'$.

This proofs the following

Theorem

For each formula φ there is

- one (up to the order of its components) uniquely defined PDNF Δ' with $\Delta' \Leftrightarrow \varphi$, called **the** PDNF of φ.
- one (up te the order of its components) uniquely defined PCNF Γ' with $\Gamma' \Leftrightarrow \varphi$, called **the** PCNF of φ.

In other words: PDNF's and PCNF's are pre–canonic and the two according normalizers are well defined.

Corollary

For each PDNF Δ

- $\Delta \Leftrightarrow 0$ iff $\Delta = [\vee]$
- $\Delta \Leftrightarrow 1$ iff $\Delta = [\vee[\wedge]]$

For each PCNF Γ

- $\Gamma \Leftrightarrow 0$ iff $\Gamma = [\wedge[\vee]]$
- $\Gamma \Leftrightarrow 1$ iff $\Gamma = [\wedge]$

The next theorem about 0– and 1–ary DNF's and CNF's, together with one about 2–ary forms stated later on, will be the starting point for our algorithms that can handle all n–ary formulas.

Theorem

- $[\vee]$ is a PDNF.
- $[\wedge]$ is a PCNF.
- Every DNF $[\vee\gamma]$ with one component is a PDNF.
- Every CNF $[\wedge\delta]$ with one component is a PCNF.

This won't need an explicit proof.

Syntactic measures

We will new seek for a precise concept of syntactic "size" and propose two syntactic measures: length and volume.

Definition

For a DNF $\Delta = [\vee[\wedge\lambda_{11}...\lambda_{1n_1}]...[\wedge\lambda_{m1}...\lambda_{mn_m}]]$ we put

- $lg(\Delta) := m$ the **length** of Δ
- $vol(\Delta) := n_1 + n_2 + ... + n_m$ the **volume** of Δ.

For a CNF $\Gamma = [\wedge[\vee\lambda_{11}...\lambda_{1n_1}]...[\vee\lambda_{m1}...\lambda_{mn_m}]]$ we put

- $lg(\Gamma) := m$ the **length** of Γ
- $vol(\Gamma) := n_1 + n_2 + ... + n_m$ the **volume** of Γ.

For example, for $\Delta = [\vee[\wedge ABC][\wedge\overline{ABC}]]$ there is $lg(\Delta) = 2$ and $vol(\Delta) = 6$.

Minimal Normal Forms

Definition

A DNF Δ is said to be

- **of minimal length**, if there is no DNF Δ' such that $\Delta' \Leftrightarrow \Delta$ and $lg(\Delta') < lg(\Delta)$
- **of minimal volume**, if there is no DNF Δ' such that $\Delta' \Leftrightarrow \Delta$ and $vol(\Delta') < vol(\Delta)$
- a **minimal disjunctive normal form** or **MDNF**, if Δ is of minimal length and minimal volume.

A CNF Γ is said to be

- **of minimal length**, if there is no CNF Γ' such that $\Gamma' \Leftrightarrow \Gamma$ and $lg(\Gamma') < lg(\Gamma)$
- **of minimal volume**, if there is no CNF Γ' such that $\Gamma' \Leftrightarrow \Gamma$ and $vol(\Gamma') < vol(\Gamma)$
- a **minimal conjunctive normal form** or **MCNF**, if Γ is of minimal length and minimal volume.

Remark

- In general, minimal length does not imply minimal volume. If we take the following two DNF's as an example
  - $\Delta_1 = [\vee[\wedge A][\wedge\overline{A}B]]$
  - $\Delta_2 = [\vee[\wedge A][\wedge B]]$

  we can easily verify that they are equivalent and both of minimal length, but obviously $2 = vol(\Delta_2) < vol(\Delta_1) = 3$.
- It remains an open question here if a minimal volume implies a minimal length.[9]

If γ is a component of a DNF Δ, but not prime, then there must be a shorter factor γ' of Δ. If Δ' is obtained from Δ by replacing γ with γ', then $\Delta' \Leftrightarrow \Delta$ and $vol(\Delta') < vol(\Delta)$. So, if a DNF is minimal, all its components must be prime factors.

Theorem

- All components of a MDNF are prime factors: Let Δ_m be a MDNF and Δ_p a PDNF with $\Delta_m \Leftrightarrow \Delta_p$, then $\Delta_m \subseteq \Delta_p$.
- All components of a MCNF are prime factors:

[9] I conjecture that this is actually the case: Normal forms of minimal volume are always of minimal length. If this wouldn't be correct, then there wouldn't be a MDNF for every formula and thus MDNF's wouldn't make a proper set of normal forms. To avoid this, we could have identified minimality with minimality of the volume. But eventually, only the MDNF's of 0–,1–, and 2–ary DNF's are of interest in our paper. And restricted to these cases the normalizer is well defined and even pre-canonic, as we will see later on.

Let Γ_m be a MCNF and Γ_p a PCNF with $\Gamma_m \Leftrightarrow \Gamma_p$, then $\Gamma_m \subseteq \Gamma_p$.

For many formulas, the MDNF and PDNF are identic. $[\vee[\wedge A][\wedge B]]$ for instance is both a MDNF and a PDNF.
But very often, they are differnt, as in the following example of two equivalent forms:
- $[\vee[\wedge AB][\wedge\overline{B}C]]$ is a MDNF
- $[\vee[\wedge AB][\wedge\overline{B}C][\wedge AC]]$ is the PDNF

The last theorem indicates a method how MDNF's can be constructed: First, find all prime factors (later on we introduce an algorithm that solves this problem). Then find out which of the subsets of the PDNF still makes an equivalent formula and select the best of them.
But, different to PDNF's MDNF's are not always unique:

Theorem
- MDNF's are not pre–canonic (thus not canonic), but 0– and 1–canonic.
- MCNF's are not pre–canonic (thus not canonic), but 0– and 1–canonic.

Proof
- Both of the following DNF's are minimal
  - $\Delta_1 = [\vee[\wedge\overline{B}C][\wedge B\overline{C}][\wedge AB]]$
  - $\Delta_2 = [\vee[\wedge\overline{B}C][\wedge B\overline{C}][\wedge AC]]$

  and $\Delta_1 \Leftrightarrow \Delta_2$, but obviously not $\Delta_1 \approx \Delta_2$. So, MDNF's are not (pre–)canonic.
- DNF's are 0–canonic according to a theorem in chapter 4 and as such MDNF's are 0–canonic.
- All components of a MDNF Δ are prime factors. The only prime factor of a tautologic DNF is $[\wedge]$. So $\Delta \Leftrightarrow 1$ iff $\Delta = [\vee[\wedge]]$. In other words, MDNF's are 1–canonic.

Theorem
- $[\vee]$ is a MDNF.
- $[\wedge]$ is a MCNF.
- Every DNF $[\vee\gamma]$ with one component is a MDNF.
- Every CNF $[\wedge\delta]$ with one component is a MCNF.

So, altogether we have "DNF = PDNF = MDNF", if the length of the form doesn't exceed 1.

Pairwise Minimal Normal Forms

In a later chapter we are going to proof that every 2–ary DNF $\Delta = [\vee\gamma_L\gamma_R]$ has a (up to the order of the components) unique MDNF Δ', that has at least one and at most two components. We will implement an algorithm to construct Δ' and we write it as $min(\gamma_L, \gamma_R) = \Delta'$. By definition the length (thus either 1 or 2) and the volume of Γ' are at most as big as the length (=2) and volume of Δ.

This algorithm *min* can be used for the following method to minimize pairwise, called the **Minimizing Procedure** or the **M–Procedure**:
(i) Choose any two components γ_L and γ_R of a given DNF Δ and replace them by components of $min(\gamma_L, \gamma_R)$.
(ii) Repeat step (i) until every pair of components of Δ is a MDNF.
M-Procedure (Δ) shall denote the result.

An example: From a given DNF two components are selected (indexed by the two underbraces), their MDNF is computed and its components are inserted into the DNF again (indexed by the one or two overbraces).

$$[\vee[\wedge\overline{ABC}][\wedge A\overline{B}][\wedge B]\underbrace{[\wedge C]}\underbrace{[\wedge C]}]$$

$$\boxed{min([\wedge C],[\wedge C]) = [\vee[\wedge C]]}$$

$$[\vee\underbrace{[\wedge\overline{ABC}]}\underbrace{[\wedge A\overline{B}]}[\wedge B]\overbrace{[\wedge C]}]$$

$$\boxed{min([\wedge\overline{ABC}],[\wedge A\overline{B}]) = [\vee[\wedge\overline{BC}][\wedge A\overline{B}]]}$$

$$[\vee\overbrace{\underbrace{[\wedge\overline{BC}]}}\overbrace{[\wedge A\overline{B}]}\underbrace{[\wedge B]}[\wedge C]]$$

$$\boxed{min([\wedge\overline{BC}],[\wedge B]) = [\vee[\wedge\overline{C}][\wedge B]]}$$

$$[\vee\overbrace{\underbrace{[\wedge\overline{C}]}}[\wedge A\overline{B}]\overbrace{[\wedge B]}\underbrace{[\wedge C]}]$$

$$\boxed{min([\wedge\overline{C}],[\wedge C]) = [\vee[\wedge]]}$$

$$[\vee\overbrace{\underbrace{[\wedge]}}\underbrace{[\wedge A\overline{B}]}]$$

$$\boxed{min([\wedge],[\wedge A\overline{B}]) = [\vee[\wedge]]}$$

$$[\vee\overbrace{\underbrace{[\wedge]}}\underbrace{[\wedge B]}]$$

$$\boxed{min([\wedge],[\wedge B]) = [\vee[\wedge]]}$$

$$[\vee \overbrace{[\wedge]}^{}]$$

Later on we implement an effective version of the M–Procedure, in which the vague instruction "choose any two components" in (i) and the test in (ii) are merged into a single step.

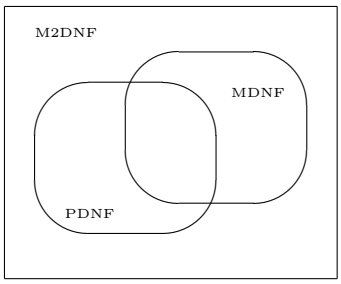
Obviously this whole method is monotone shortening, the DNF is constantly getting better in this respect. One might be tempted to suggest that the finite result is always a MDNF. It therefore might surprise that this is very often the case indeed, but not always. And neither is the result a PDNF in general. Therefore we give the forms that satisfy the terminating condition of the method, an own title:

Definition

- A DNF $\Delta = [\vee\gamma_1...\gamma_n]$ is called **pairwise minimal** or a **M2DNF** iff $[\vee\gamma_i\gamma_j]$ is minimal for all $i,j \in \{1,...,n\}$ with $i \neq j$.
- A CNF $\Gamma = [\wedge\delta_1...\delta_n]$ is called **pairwise minimal** or a **M2CNF** iff $[\wedge\delta_i\delta_j]$ is minimal for all $i,j \in \{1,...,n\}$ with $i \neq j$.

Theorem

- Every MDNF is a M2DNF.
- Not every M2DNF is a MDNF.
- Every PDNF is a M2DNF.
- Not every M2DNF is a PDNF
- Every MCNF is a M2CNF.
- Not every M2CNF is a MCNF.
- Every PCNF is a M2CNF.
- Not every M2CNF is a PCNF.



Proof

- Obviously, every MDNF is a M2DNF.
- $\Delta = [\vee[\wedge A\overline{B}][\wedge B\overline{C}][\wedge AC]]$ is an example for a M2DNF that isn't a MDNF. The MDNF of Δ is $\Delta' = [\vee[\wedge A][\wedge B\overline{C}]]$.
- Suppose there is a PDNF $\Delta = [\vee\gamma_1...\gamma_n]$ that isn't a M2DNF. So there must be $\gamma_i, \gamma_j \in \Delta$ such that there is either a 1–ary DNF $[\vee\gamma'] \Leftrightarrow [\vee\gamma_i\gamma_j]$ or a 2–ary DNF $[\vee\gamma'_i\gamma'_j] \Leftrightarrow [\vee\gamma_i\gamma_j]$ with $\gamma_i \Rightarrow \gamma'_i$ or $\gamma_j \Rightarrow \gamma'_j$. (We will show that in detail later on.). But that would contradict the premise that γ_i and γ_j both have to be prime factors of Δ.
- Again $\Delta = [\vee[\wedge A\overline{B}][\wedge B\overline{C}][\wedge AC]]$ is an example for a M2DNF that isn't a PDNF, which is again Δ'.

Theorem

- M2DNF's are 0–canonic, but not 1–canonic and thus not (pre–)canonic.
- M2CNF's are 1–canonic, but not 0–canonic and thus not (pre–)canonic.

Proof

DNF's are 0–canonic in general, so are M2DNF's.
$\Delta = [\vee[\wedge]]$ and $\Delta' = [\vee\ [\wedge\overline{BCD}]\ [\wedge AC\overline{D}]\ [\wedge\overline{A}B\overline{C}]\ [\wedge AB\overline{C}]\ [\wedge\overline{A}B\overline{D}]\ [\wedge BCD]\ [\wedge\overline{A}CD]\ [\wedge A\overline{B}D]]$ are both tautologies and pairwise minimal, and so the M2DNF's are not 1–canonic.

Quality comparison

Let us now compare the three special types of DNF's according to the given criteria in the introduction.

- Intuitivity
  PDNF's have the big advantage against the other two types that all factors can be read off directly from the form, and thus all subvalent formulas become directly identifiable, if they are given in DNF.[10] In this respect, the PDNF's are the most transparent of all DNF's.
- Canonicity
  Of all three sets of DNF's, only the PDNF's are canonic.
- Complexity of the representation
  Actually all three types of normal forms are relatively condensed — this goal was our initial starting point. But the MDNF's are the absolutely shortest by definition. PDNF's and M2DNF's both allow examples for which one type is shorter than the other.[11]
- Computational complexity
  Later on we are going to introduce an algorithm

[10] see next chapter about "covering"

[11] We already saw that a MDNF is always a subset of the equivalent PDNF. I conjecture that the MDNF is always at least half as long as the PDNF. [Comment September 2002: Actually, this conjecture is not correct in general.]

of polynomial complexity to transform DNF's into M2DNF's. We will extend this idea to build a PDNF–normalizer. PDNF's are very efficient for the test on subvalence (see next chapter). Usually MDNF's are constructed out of PDNF's and thus MDNF's have the highest computational costs.

- Conservativity
  The transformation of an arbitrary formula into a DNF usually changes the whole structure of the form. A further reduction into one of the three special DNF's is, compare to that, very moderate.
- Realizability as technical devices
  I suppose, this is the real domain for the MDNF's.

Conclusion

M2DNF's are not very interesting for practical applications. Nevertheless they are indispensable for the theory and implementation of our PDNF system.

MDNF's are only superior when the representational complexity is the only and most important aspect. There are established methods to construct MDNF's out of given PDNF's and I wouldn't know how to improve them. Therefore we will neglect these forms in the sequel.

PDNF's are the only canonic forms. The rest of this paper is devoted to the construction of an effective PDNF system.

The analysis of the relation between two NLC's stands at the center of our method and therefore this investigation is a chapter on its own. Subsequently we will first build a M2DNF and then a PDNF system. But first of all we demonstate the subvalence test with prime normal forms, that also provides us with a criterion to decide between a PDNF and a PCNF system in concrete applications.

5 Covering and Subvalence

Because the PDNF-normalizer is 1–canonic, it holds for any two formulas φ_1 and φ_2 that

$$\varphi_1 \Rightarrow \varphi_2 \text{ iff } pdnf([\varphi_1 \rightarrow \varphi_2]) = [\vee[\wedge]]$$

In this way it is possible to implement the **subvalence problem**, i.e. the question, whether or not a formular implies another one.
In actual applications it is very often the case that one side is relatively constant (and then it is often called "program", "data bank", "axiom system", etc.), while the other side (called "input", "request", etc.) varies. We could name application systems of the sort $\varphi_{constant} \Rightarrow \varphi_{variable}$ "**consequence systems**" and the others of the sort $\varphi_{variable} \Rightarrow \varphi_{constant}$ "**implicand systems**".

For these two kind of systems there is a much more efficient method for the subvalence problem than the one just mentioned:

- For consequence systems we construct the PCNF $\Gamma_{constant}$ of $\varphi_{constant}$ and a CNF $\Gamma_{variable}$ of $\varphi_{variable}$ and we decide the truth of $\Gamma_{constant} \Rightarrow \Gamma_{variable}$ by using the following covering concept.
- Accordingly the application of DNF's suits very well for imlicand systems.

The "covering method" reduces the subvalence problem to the pairwise comparison of NLC's (NLD's) and that doesn't only accelerate the computer, it also supports the comprehension and mental arithmetic of formulas. The semantic of prime normal forms is exceptionally intuitive.

We noticed earlier that:

- For any two NLC's γ_1 and γ_2: $\gamma_1 \Rightarrow \gamma_2$ iff $\gamma_2 \subseteq \gamma_1$.
- For any two NLD's δ_1 and δ_2: $\delta_1 \Rightarrow \delta_2$ iff $\delta_1 \subseteq \delta_2$.

Definition

For each NLC γ and DNF's $\Delta = [\vee\gamma_1...\gamma_n]$ and Δ' we define:

- $\gamma \sqsubseteq \Delta$, γ is **covered by** Δ, iff there is a $\gamma_i \in \Delta$ such that $\gamma \Rightarrow \gamma_i$.
- $\gamma \not\sqsubseteq \Delta$ iff γ is not covered by Δ.
- $\Delta \sqsubseteq \Delta'$, Δ is **covered by** Δ', iff $\gamma_i \sqsubseteq \Delta'$ for every $\gamma_i \in \Delta$.
- $\Delta \not\sqsubseteq \Delta'$ iff Δ is not covered by Δ'.

For each NLD δ and CNF's $\Gamma = [\wedge\delta_1...\delta_n]$ and Γ' we define:

- $\delta \sqsubseteq \Gamma$, δ is **covered by** Γ, iff there is a $\delta_i \in \Gamma$ such that $\delta_i \Rightarrow \delta$.
- $\delta \not\sqsubseteq \Gamma$ iff δ is not covered by Γ.
- $\Gamma \sqsubseteq \Gamma'$, Γ is **covered by** Γ', iff $\delta_i \sqsubseteq \Gamma'$ for every $\delta_i \in \Gamma$.
- $\Gamma \not\sqsubseteq \Gamma'$ iff Γ is not covered by Γ'.

For DNF's $\Delta \sqsubseteq \Delta'$ always implies $\Delta \Rightarrow \Delta'$. But the reverse is generally only the case if Δ' is prime[12].

Theorem

- Let Δ be a DNF and Δ_p a PDNF. Then $\Delta \Rightarrow \Delta_p$ iff $\Delta \sqsubseteq \Delta_p$.
- Let Γ_p be a PCNF and Γ a CNF. Then $\Gamma_p \Rightarrow \Gamma$ iff $\Gamma \sqsubseteq \Gamma_p$.

Proof

For $\Delta = [\vee\gamma_1...\gamma_n]$ there is $\Delta \Rightarrow \Delta'$ iff $[[\vee\gamma_1...\gamma_n] \rightarrow \Delta'] \Leftrightarrow 1$ iff $[\vee[\wedge\neg\gamma_1...\neg\gamma_n]\Delta'] \Leftrightarrow 1$ iff $[\wedge[\vee\neg\gamma_1\Delta']...[\vee\neg\gamma_n\Delta']] \Leftrightarrow 1$ iff $[\wedge[\gamma_1 \rightarrow \Delta']...[\gamma_n \rightarrow \Delta']] \Leftrightarrow 1$ iff $\gamma_i \Rightarrow \Delta'$ for each $i = 1, ..., n$. Thus every $\gamma_i \in \Delta$ has to be a factor of Δ'. Since Δ' contains all prime factors, there must be a $\gamma'_j \in \Delta'$ for each $\gamma_i \in \Delta$ such that $\gamma_i \Rightarrow \gamma'_j$. Thus $\gamma_i \sqsubseteq \Delta'$ for each $i = 1, ..., n$ and thus $\Delta \sqsubseteq \Delta'$.

Theorem

$\sqsubseteq$ is a quasi-order[13] on the set of all DNF's (CNF's).

[12] More general, if Δ' is "f–complete" (see below).
[13] reflexive and transitive

6 * Implementation of a fast subvalence decision with Prime Normal Forms

The following table of algorithms gives an example of how these results can be implemented. It can later be attached to the fully developed PDNF and PCNF system.

The PDNF system: Declarations	**The PCNF system: Declarations**
For NLC's γ_1 and γ_2 $\text{nlc–sub–nlc}(\gamma_1, \gamma_2) = \begin{cases} 0 & \text{if } \gamma_1 \not\Rightarrow \gamma_2 \\ 1 & \text{if } \gamma_1 \Rightarrow \gamma_2 \end{cases}$	For NLD's δ_1 and δ_2 $\text{nld–sub–nld}(\delta_1, \delta_2) = \begin{cases} 0 & \text{if } \delta_1 \not\Rightarrow \delta_2 \\ 1 & \text{if } \delta_1 \Rightarrow \delta_2 \end{cases}$
For a NLC γ and a DNF Δ $\text{nlc–cov–dnf}(\gamma, \Delta) = \begin{cases} 0 & \text{if } \gamma \not\sqsubseteq \Delta \\ 1 & \text{if } \gamma \sqsubseteq \Delta \end{cases}$	For a NLD δ and a CNF Γ $\text{nld–cov–cnf}(\delta, \Gamma) = \begin{cases} 0 & \text{if } \delta \not\sqsubseteq \Gamma \\ 1 & \text{if } \delta \sqsubseteq \Gamma \end{cases}$
For DNF's Δ_1 and Δ_2 $\text{dnf–cov–dnf}(\Delta_1, \Delta_2) = \begin{cases} 0 & \text{if } \Delta_1 \not\sqsubseteq \Delta_2 \\ 1 & \text{if } \Delta_1 \sqsubseteq \Delta_2 \end{cases}$	For CNF's Γ_1 and Γ_2 $\text{cnf–cov–cnf}(\Gamma_1, \Gamma_2) = \begin{cases} 0 & \text{if } \Gamma_1 \not\sqsubseteq \Gamma_2 \\ 1 & \text{if } \Gamma_1 \sqsubseteq \Gamma_2 \end{cases}$
For a DNF Δ_1 and a PDNF Δ_2 $\text{dnf–sub–pdnf}(\Delta_1, \Delta_2) = \begin{cases} 0 & \text{if } \Delta_1 \not\Rightarrow \Delta_2 \\ 1 & \text{if } \Delta_1 \Rightarrow \Delta_2 \end{cases}$	For a PCNF Γ_1 and a CNF Γ_2 $\text{pcnf–sub–cnf}(\Gamma_1, \Gamma_2) = \begin{cases} 0 & \text{if } \Gamma_1 \not\Rightarrow \Gamma_2 \\ 1 & \text{if } \Gamma_1 \Rightarrow \Gamma_2 \end{cases}$
The PDNF system: Implementations	**The PCNF system: Implementations**
$\text{nlc–sub–nlc}([\wedge\lambda_1 \ldots \lambda_n], [\wedge\lambda'_1 \ldots \lambda'_m])$ $:= \begin{cases} 1 & \text{if } m = 0 \\ 0 & \text{if } m > 0 \text{ and } n = 0 \\ \text{nlc–sub–nlc}([\wedge\lambda_2 \ldots \lambda_n], [\wedge\lambda'_1 \ldots \lambda'_m]) & \text{if } \mid \lambda_1 \mid < \mid \lambda'_1 \mid \\ 0 & \text{if } \mid \lambda'_1 \mid < \mid \lambda_1 \mid \\ \text{nlc–sub–nlc}([\wedge\lambda_2 \ldots \lambda_n], [\wedge\lambda'_2 \ldots \lambda_m]) & \text{if } \lambda_1 = \lambda'_1 \\ 0 & \text{if } \overline{\lambda_1} = \lambda'_1 \end{cases}$	$\text{nld–sub–nld}([\vee\lambda_1 \ldots \lambda_n], [\vee\lambda'_1 \ldots \lambda'_m])$ $:= \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{if } n > 0 \text{ and } m = 0 \\ 0 & \text{if } \mid \lambda_1 \mid < \mid \lambda'_1 \mid \\ \text{nld–sub–nld}([\vee\lambda_1 \ldots \lambda_n], [\vee\lambda'_2 \ldots \lambda'_m]) & \text{if } \mid \lambda'_1 \mid < \mid \lambda_1 \mid \\ \text{nld–sub–nld}([\vee\lambda_2 \ldots \lambda_n], [\vee\lambda'_2 \ldots \lambda_m]) & \text{if } \lambda_1 = \lambda'_1 \\ 0 & \text{if } \overline{\lambda_1} = \lambda'_1 \end{cases}$
$\text{nlc–cov–dnf}(\gamma, [\vee\gamma_1 \ldots \gamma_n])$ $:= \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if nlc–sub–nlc}(\gamma, \gamma_1) = 1 \\ \text{nlc–cov–dnf}(\gamma, [\vee\gamma_2 \ldots \gamma_n]) & \text{if nlc–sub–nlc}(\gamma, \gamma_1) = 0 \end{cases}$	$\text{nld–cov–cnf}(\delta, [\wedge\delta_1 \ldots \delta_n])$ $:= \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if nld–sub–nld}(\delta_1, \delta) = 1 \\ \text{nld–cov–cnf}(\delta, [\wedge\delta_2 \ldots \delta_n]) & \text{if nld–sub–nld}(\delta_1, \delta) = 0 \end{cases}$
$\text{dnf–cov–dnf}([\vee\gamma_1 \ldots \gamma_n], \Delta)$ $:= \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{if nlc–cov–dnf}(\gamma_1, \Delta) = 0 \\ \text{dnf–cov–dnf}([\vee\gamma_2 \ldots \gamma_n], \Delta) & \text{if nlc–cov–dnf}(\gamma_1, \Delta) = 1 \end{cases}$	$\text{cnf–cov–cnf}([\wedge\delta_1 \ldots \delta_n], \Gamma)$ $:= \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{if nld–cov–cnf}(\delta_1, \Gamma) = 0 \\ \text{cnf–cov–cnf}([\wedge\delta_2 \ldots \delta_n], \Gamma) & \text{if nld–cov–cnf}(\delta_1, \Gamma) = 1 \end{cases}$
$\text{dnf–sub–pdnf}(\Delta_1, \Delta_2)$ $:= \text{dnf–cov–dnf}(\Delta_1, \Delta_2)$	$\text{pcnf–sub–cnf}(\Gamma_1, \Gamma_2)$ $:= \text{cnf–cov–cnf}(\Gamma_1, \Gamma_2)$

7 Component pairs

In this chapter we will construct an algorithm that determines the MDNF and the PDNF of a 2-ary DNF $[\vee\gamma_L\gamma_R]$.[14] Both do exist and they are unique (up to the order of their components.)

First we introduce a special notation for a literal conjunction that might not be normal yet, but doesn't contain doubles or complementary literals, and thus can easily be normalized by changing the order of the components:

Notation

Let $[\wedge\lambda_1...\lambda_n]$ be a literal conjunction and $i : \{1,...,n\} \longrightarrow \{1,...,n\}$ a bijection such that $[\wedge\lambda_{i(1)}...\lambda_{i(n)}]$ is a NLC. (Note that this bijection is unique if it exists.) We then write

$$[\dot{\wedge}\lambda_1...\lambda_n] := [\wedge\lambda_{i(1)}...\lambda_{i(n)}]$$

Definition

Let

- $\gamma_L = [\wedge\lambda_1...\lambda_x]$ be a NLC
- $\gamma_R = [\wedge\lambda'_1...\lambda'_y]$ be a NLC

We define the following sets (the order of the elements is of no interest here):

- $\{\pi_1,...,\pi_p\} := \{\lambda_1,...,\lambda_x\} \cap \{\lambda'_1,...,\lambda'_y\}$
- $\{\rho_1,...,\rho_r\} := \{\lambda_1,...,\lambda_x\} \cap \{\overline{\lambda'_1},...,\overline{\lambda'_y}\}$
- $\{\sigma_1,...,\sigma_s\} := \{\lambda_1,...,\lambda_x\} - \{\lambda'_1,...,\lambda'_y,\overline{\lambda'_1},...,\overline{\lambda'_y}\}$
- $\{\tau_1,...,\tau_t\} := \{\lambda'_1,...,\lambda'_y\} - \{\lambda_1,...,\lambda_x,\overline{\lambda_1},...,\overline{\lambda_x}\}$

Thus

- $\gamma_L = [\dot{\wedge}\pi_1...\pi_p\rho_1...\rho_r\sigma_1...\sigma_s]$
- $\gamma_R = [\dot{\wedge}\pi_1...\pi_p\overline{\rho_1}...\overline{\rho_r}\tau_1...\tau_t]$

We define

- $min(\gamma_L,\gamma_R)$
- $prim(\gamma_L,\gamma_R)$

according to the following table, depending on the values of p, r, s, and t. In this table we use the abbreviations[15]

- "N" for any *N*umber, i.e. ≥ 0
- "P" for *P*ositive, i.e. ≥ 1
- "M" for *M*ultiple, i.e. > 1

Without its long and non–informative proof we state the following

Theorem

- $min(\gamma_L,\gamma_R)$ is the (up to the order of its components) uniquely defined MDNF of $[\vee\gamma_L\gamma_R]$.
- $prim(\gamma_L,\gamma_R)$ is the (up to the order of its components) uniquely defined PDNF of $[\vee\gamma_L\gamma_R]$.

Example

$\gamma_L := [\wedge ABD\overline{EGK}]$
$\gamma_R := [\wedge A\overline{B}CE\overline{H}IJ\overline{K}L]$
For these two NLC's we have

$\{\pi_1,...,\pi_p\} = \{A,\overline{K}\}$	thus $p=2$
$\{\rho_1,...,\rho_r\} = \{B,\overline{E}\}$	thus $r=2$
$\{\sigma_1,...,\sigma_s\} = \{D,\overline{G}\}$	thus $s=2$
$\{\tau_1,...,\tau_t\} = \{C,\overline{H},I,J,L\}$	thus $t=5$

This is the case "NMNN"in the table and so $min(\gamma_L,\gamma_R) = prim(\gamma_L,\gamma_R) = [\vee\gamma_L\gamma_R]$.

We can see in the table that the number p of literals that occur both in γ_L and in γ_R, doesn't have any influence on the MDNF and PDNF solution, while the number r of literals that occur in γ_L and complemented in γ_R is very decisive. Therefore we call $[\vee\gamma_L\gamma_R]$ or the relation between γ_L and γ_R

- **incomplementary**, if $r = 0$
- **complementary**, if $r = 1$
- **hyper–complementary** if $r > 1$.

Furthermore we call $[\vee\gamma_L\gamma_R]$ (or γ_L and γ_R) **minimal**, iff $min(\gamma_L,\gamma_R) = [\vee\gamma_L\gamma_R]$.
As the table shows, this is exactly the case when

$$r > 1 \text{ or } (s > 0 \text{ and } t > 0).$$

In case γ_L and γ_R are complementary and minimal — and only in this case — the PDNF and the MDNF of $[\vee\gamma_L,\gamma_R]$ are different: the PDNF contains one additional prime factor. We call it the **complementary prime factor** or shorter the **c–prime factor** of γ_L and γ_R (or $[\vee\gamma_L\gamma_R]$).

[14] The subscripts "L" and "R" stand for "left" and "right".

[15] We introduce these symbols, because in later algorithms we will make use the **case symbols** "N000", "N00P", ..., "NMNN"as identification symbols for the nine different cases.

p	r	s	t	γ_L	γ_R	$min\,(\gamma_L, \gamma_R)$	$prim\,(\gamma_L, \gamma_R)$
N	0	0	0	$[\dot{\wedge}\pi_1 \ldots \pi_p]$	$[\dot{\wedge}\pi_1 \ldots \pi_p]$	$[\vee\gamma_L] = [\vee\gamma_R]$	$[\vee\gamma_L] = [\vee\gamma_R]$
N	0	0	P	$[\dot{\wedge}\pi_1 \ldots \pi_p]$	$[\dot{\wedge}\pi_1 \ldots \pi_p \tau_1 \ldots \tau_t]$	$[\vee\gamma_L]$	$[\vee\gamma_L]$
N	0	P	0	$[\dot{\wedge}\pi_1 \ldots \pi_p \sigma_1 \ldots \sigma_s]$	$[\dot{\wedge}\pi_1 \ldots \pi_p]$	$[\vee\gamma_R]$	$[\vee\gamma_R]$
N	0	P	P	$[\dot{\wedge}\pi_1 \ldots \pi_p \sigma_1 \ldots \sigma_s]$	$[\dot{\wedge}\pi_1 \ldots \pi_p \tau_1 \ldots \tau_t]$	$[\vee\gamma_L\gamma_R]$	$[\vee\gamma_L\gamma_R]$
N	1	0	0	$[\dot{\wedge}\pi_1 \ldots \pi_p \rho_1]$	$[\dot{\wedge}\pi_1 \ldots \pi_p \overline{\rho_1}]$	$[\vee\gamma']$ with $\gamma' := [\dot{\wedge}\pi_1 \ldots \pi_p]$	$[\vee\gamma']$ with $\gamma' := [\dot{\wedge}\pi_1 \ldots \pi_p]$
N	1	0	P	$[\dot{\wedge}\pi_1 \ldots \pi_p \rho_1]$	$[\dot{\wedge}\pi_1 \ldots \pi_p \overline{\rho_1} \tau_1 \ldots \tau_t]$	$[\vee\gamma_L\gamma'_R]$ with $\gamma'_R := [\dot{\wedge}\pi_1 \ldots \pi_p \tau_1 \ldots \tau_t]$	$[\vee\gamma_L\gamma'_R]$ with $\gamma'_R := [\dot{\wedge}\pi_1 \ldots \pi_p \tau_1 \ldots \tau_t]$
N	1	P	0	$[\dot{\wedge}\pi_1 \ldots \pi_p \rho_1 \sigma_1 \ldots \sigma_s]$	$[\dot{\wedge}\pi_1 \ldots \pi_p \overline{\rho_1}]$	$[\vee\gamma'_L\gamma_R]$ with $\gamma'_L := [\dot{\wedge}\pi_1 \ldots \pi_p \sigma_1 \ldots \sigma_s]$	$[\vee\gamma'_L\gamma_R]$ with $\gamma'_L := [\dot{\wedge}\pi_1 \ldots \pi_p \sigma_1 \ldots \sigma_s]$
N	1	P	P	$[\dot{\wedge}\pi_1 \ldots \pi_p \rho_1 \sigma_1 \ldots \sigma_s]$	$[\dot{\wedge}\pi_1 \ldots \pi_p \overline{\rho_1} \tau_1 \ldots \tau_t]$	$[\vee\gamma_L\gamma_R]$	$[\vee\gamma_L\gamma_R\gamma_c]$ with $\gamma_c := [\dot{\wedge}\pi_1 \ldots \pi_p \sigma_1 \ldots \sigma_s \tau_1 \ldots \tau_t]$ **the c–prime factor**
N	M	N	N	$[\dot{\wedge}\pi_1 \ldots \pi_p \rho_1 \ldots \rho_r \sigma_1 \ldots \sigma_s]$	$[\dot{\wedge}\pi_1 \ldots \pi_p \overline{\rho_1} \ldots \overline{\rho_r} \tau_1 \ldots \tau_t]$	$[\vee\gamma_L\gamma_R]$	$[\vee\gamma_L\gamma_R]$

8 * Implementation: Minimal and Prime Normal Forms of component pairs

The implementation of the two functions $min(\gamma_L, \gamma_R)$ and $prim(\gamma_L, \gamma_R)$ could be straightforward: first, determine the four sets $\{\pi_1, ..., \pi_p\}$, $\{\rho_1, ..., \rho_r\}$, $\{\sigma_1, ..., \sigma_s\}$, and $\{\tau_1, ..., \tau_t\}$. Then, look up the case and the according instruction in the table and finally construct the formula. But we are eager to create a fast system. *min* is the core of the M2DNF system, *prim* will be the most often applied function in the PDNF system.

In fact, the algorithms will not use *min* and *prim*, but the "extended" versions *xmin* and *xprim*, where $xmin(\gamma_L, \gamma_R) := (\xi, min(\gamma_L, \gamma_R))$ and $xprim(\gamma_L, \gamma_R) := (\xi, prim(\gamma_L, \gamma_R))$ and ξ is the **case symbol**, i.e. the abbreviations for the numbers of p,r,s, and t in the table.

Example

For $\gamma_L = [\wedge A\overline{B}DF]$ and $\gamma_R = [\wedge BC]$ there are

- $xmin\ (\gamma_L, \gamma_R) =$ ("N1PP", $[\vee[\wedge A\overline{B}DF][\wedge BC]]$)
- $xprim\ (\gamma_L, \gamma_R) =$ ("N1PP", $[\vee\ [\wedge A\overline{B}DF]\ [\wedge BC]\ [\wedge ACDF]]$)

For the implementation of *xprim* we use the auxiliary recursive function *xprec*. *xprec* terminates as soon as the case is clear, and that is mainly what makes it faster than the straightforward version. *xprec* is not the only version that would do the job, we don't need to discuss it and the reader is free to use an own version.
The duality allows us to formulate *xprec* for DNF's and CNF's together and we use $*$ as the junctor symbol and $\overline{*}$ for the dual junctor.

Disjunctive component pairs: Declarations	**Conjunctive component pairs: Declarations**
$prim(\gamma_L, \gamma_R)$ is the PDNF of $[\vee\gamma_L\gamma_R]$, for any two NLC's γ_L and γ_R	$prim(\delta_L, \delta_R)$ is the PCNF of $[\wedge\delta_L\delta_R]$, for any two NLD's δ_L and δ_R
$min(\gamma_L, \gamma_R)$ is the MDNF of $[\vee\gamma_L\gamma_R]$, for any two NLC's γ_L and γ_R	$min(\delta_L, \delta_R)$ is the MCNF of $[\wedge\delta_L\delta_R]$, for any two NLD's δ_L and δ_R
$xprim(\gamma_L, \gamma_R)$ $:= (\xi, prim(\gamma_L, \gamma_R))$ where ξ is the case symbol according to the table in chapter 7	$xprim(\delta_L, \delta_R)$ $:= (\xi, prim(\delta_L, \delta_R))$ where ξ is the case symbol according to the table in chapter 7
$xmin(\gamma_L, \gamma_R)$ $:= (\xi, min(\gamma_L, \gamma_R))$ where ξ is the case symbol according to the table in chapter 7	$xmin(\delta_L, \delta_R)$ $:= (\xi, min(\delta_L, \delta_R))$ where ξ is the case symbol according to the table in chapter 7

Disjunctive component pairs: Implementations	**Conjunctive component pairs: Implementations**
$xprim(\gamma_L, \gamma_R)$	$xprim(\delta_L, \delta_R)$
$:= xprec([\wedge], \gamma_L, [\wedge], \gamma_R, [\wedge], [\wedge], [\wedge], 0, 0, 0)$	$:= xprec([\vee], \delta_L, [\vee], \delta_L, [\vee], [\vee], [\vee], 0, 0, 0)$
$prim(\gamma_L, \gamma_R)$	$prim(\delta_L, \delta_R)$
$:= \Delta$, where Δ is defined by $(\xi, \Delta) := xprim(\gamma_L, \gamma_R)$	$:= \Gamma$, where Γ is defined by $(\xi, \Gamma) := xprim(\delta_L, \delta_R)$
$xmin(\gamma_L, \gamma_R)$	$xmin(\delta_L, \delta_R)$
$:= (\xi, \Delta')$ defined by $(\xi, \Delta) := xprim(\gamma_L, \gamma_R)$ and	$:= (\xi, \Gamma')$ defined by $(\xi, \Gamma) := xprim(\delta_L, \delta_R)$ and
$\Delta' := \begin{cases} \Delta & \text{if } lg(\Delta) \in \{1, 2\} \\ [\vee\gamma_L, \gamma_R] & \text{if } lg(\Delta) = 3 \end{cases}$	$\Gamma' := \begin{cases} \Gamma & \text{if } lg(\Gamma) \in \{1, 2\} \\ [\wedge\delta_L, \delta_R] & \text{if } lg(\Gamma) = 3 \end{cases}$
$min(\gamma_L, \gamma_R)$	$min(\delta_L, \delta_R)$
$:= \Delta$, where Δ is defined by $(\xi, \Delta) := xmin(\gamma_L, \gamma_R)$	$:= \Gamma$, where Γ is defined by $(\xi, \Gamma) := xmin(\delta_L, \delta_R)$

Auxiliary algorithms

$\overline{*} := \left\{ \begin{array}{ll} \vee & \text{if } * = \wedge \\ \wedge & \text{if } * = \vee \end{array} \right\}$ for $* \in \{\wedge, \vee\}$

algorithm *xprec* $([*\lambda_1 ... \lambda_i], [*\lambda_{i+1} ... \lambda_x], [*\lambda'_1 ... \lambda'_j], [*\lambda'_{j+1} ... \lambda'_y], [*\pi_1 ... \pi_a], [*\pi'_1 ... \pi'_b], [*\pi''_1 ... \pi''_c], r, s, t)$

begin

if $r > 1$

then ("NMNN", $[\overline{*}[*\lambda_1 ... \lambda_i \lambda_{i+1} ... \lambda_x][*\lambda'_1 ... \lambda'_j \lambda'_{j+1} ... \lambda'_y]]$)

else *if* $x = i$

then if $r = 0$

then *if* $s = 0$

then if $(j = y \text{ and } t = 0)$

then (“N000”, $[\overline{*}[*\lambda_1 \ldots \lambda_i]]$)

else (“N00P”, $[\overline{*}[*\lambda_1 \ldots \lambda_i]]$)

else if $(j = y \text{ and } t = 0)$

then (“N0P0”, $[\overline{*}[*\lambda'_1 \ldots \lambda'_j]]$)

else (“N0PP”, $[\overline{*}[*\lambda_1 \ldots \lambda_i][*\lambda'_1 \ldots \lambda'_j \lambda'_{j+1} \ldots \lambda'_y]]$)

else if $s = 0$

then if $(j = y \text{ and } t = 0)$

then (“N100”, $[\overline{*}[*\pi_1 \ldots \pi_a]]$)

else (“N10P”, $[\overline{*}[*\lambda_1 \ldots \lambda_i][*\pi'_1 \ldots \pi'_b \lambda'_{j+1} \ldots \lambda'_y]]$)

else if $(j = y \text{ and } t = 0)$

then (“N1P0”, $[\overline{*}[*\pi_1 \ldots \pi_a][*\lambda'_1 \ldots \lambda'_j \lambda'_{j+1} \ldots \lambda'_y]]$)

else (“N1PP”, $[\overline{*}[*\lambda_1 \ldots \lambda_i][*\lambda'_1 \ldots \lambda'_j \lambda'_{j+1} \ldots \lambda'_y][*\pi''_1 \ldots \pi''_c \lambda'_{j+1} \ldots \lambda'_y]]$)

else if $j = y$

then if $r = 0$

then if $t = 0$

then (“N0P0”, $[\overline{*}[*\lambda'_1 \ldots \lambda'_j]]$)

else (“N0PP”, $[\overline{*}[*\lambda_1 \ldots \lambda_i \lambda_{i+1} .. \lambda_x][*\lambda'_1 \ldots \lambda'_j]]$)

else if $t = 0$

then (“N1P0”, $[\overline{*}[*\pi_1 \ldots \pi_a \lambda_{i+1} \ldots \lambda_x][*\lambda'_1 \ldots \lambda'_j]]$)

else (“N1PP”, $[\overline{*}[*\lambda_1 \ldots \lambda_i \lambda_{i+1} \ldots \lambda_x][*\lambda'_1 \ldots \lambda'_j][*\pi''_1 \ldots \pi''_c \lambda_{i+1} \ldots \lambda_x]]$)

else if $|\lambda_{i+1}| < |\lambda'_{j+1}|$

then xprec $([*\lambda_1 \ldots \lambda_i \lambda_{i+1}], [*\lambda_{i+2} \ldots \lambda_x], [*\lambda'_1 \ldots \lambda'_j], [*\lambda'_{j+1} \ldots \lambda'_y], [*\pi_1 \ldots \pi_a \lambda_{i+1}], [*\pi'_1 \ldots \pi'_b], [*\pi''_1 \ldots \pi''_c \lambda_{i+1}], r, s+1, t)$

else if $|\lambda'_{j+1}| < |\lambda_{i+1}|$

then xprec $([*\lambda_1 \ldots \lambda_i], [*\lambda_{i+1} \ldots \lambda_x], [*\lambda'_1 \ldots \lambda'_j \lambda'_{j+1}], [*\lambda'_{j+2} \ldots \lambda'_y], [*\pi_1 \ldots \pi_a], [*\pi'_1 \ldots \pi'_b \lambda'_{j+1}], [*\pi''_1 \ldots \pi''_c \lambda'_{j+1}], r, s, t+1)$

else if $\lambda_{i+1} = \lambda_{j+1}$

then xprec $([*\lambda_1 \ldots \lambda_i \lambda_{i+1}], [*\lambda_{i+2} \ldots \lambda_x], [*\lambda'_1 \ldots \lambda'_j \lambda'_{j+1}], [*\lambda'_{j+2} \ldots \lambda'_y], [*\pi_1 \ldots \pi_a \lambda_{i+1}], [*\pi'_1 \ldots \pi'_b \lambda'_{j+1}], [*\pi''_1 \ldots \pi''_c \lambda'_{j+1}], r, s, t)$

else xprec $([*\lambda_1 \ldots \lambda_i], [*\lambda_{i+2} \ldots \lambda_x], [*\lambda'_1 \ldots \lambda'_j], [*\lambda'_{j+2} \ldots \lambda'_y], [*\pi_1 \ldots \pi_a], [*\pi'_1 \ldots \pi'_b], [*\pi''_1 \ldots \pi''_c], r+1, s, t)$

end.

9 The M–Procedure

We already mentioned the **M-Procedure**:
Given a DNF Δ, then
(i) Choose any two components γ_L and γ_R of Δ and replace them by the components of $min\,(\gamma_L, \gamma_R)$.
(ii) Repeat step (i) until every pair of components of Δ is minimal.
M–Procedure (Δ) shall denote the final result.

The result *M-Procedure* (Δ) is not uniquely defined. Depending on how the instruction "Choose any two components ..." is implemented, the result can be different (i.e. not equal and not similar), as the following example demonstrates. This shows again that the set of M2DNF's is not (pre-)canonic.

Example
First version:

$$\Delta = [\vee \underbrace{[\wedge AB\overline{C}]}\,\underbrace{[\wedge ABC]}[\wedge A\overline{B}C][\wedge \overline{A}BC]]$$

$$\boxed{min([\wedge AB\overline{C}], [\wedge ABC]) = [\vee[\wedge AB]]}$$

$$[\vee \overbrace{[\wedge AB]}\,\underbrace{[\wedge A\overline{B}C]}\,\underbrace{[\wedge \overline{A}BC]}]$$

$$\boxed{min([\wedge A\overline{B}C], [\wedge \overline{A}BC]) = [\vee[\wedge \overline{B}C]]}$$

$$[\vee[\wedge AB]\,\overbrace{[\wedge \overline{B}C]}] =: \Delta_1$$

Second version:

$$\Delta = [\vee \underbrace{[\wedge AB\overline{C}]}\,\underbrace{[\wedge ABC]}[\wedge A\overline{B}C][\wedge \overline{A}BC]]$$

$$\boxed{min([\wedge AB\overline{C}], [\wedge ABC]) = [\vee[\wedge AB]]}$$

$$[\vee \overbrace{[\wedge AB]}\,\underbrace{[\wedge A\overline{B}C]}[\wedge \overline{A}BC]]$$

$$\boxed{min([\wedge AB], [\wedge A\overline{B}C]) = [\vee[\wedge AB][\wedge AC]]}$$

$$[\vee \overbrace{[\wedge AB]}\,\overbrace{[\wedge AC]}\,\underbrace{[\wedge \overline{A}BC]}]$$

$$\boxed{min([\wedge AC], [\wedge \overline{A}BC]) = [\vee[\wedge AC][\wedge \overline{B}C]]}$$

$$[\vee[\wedge AB]\,\overbrace{[\wedge AC]}\,\overbrace{[\wedge \overline{B}C]}] =: \Delta_2$$

Depending on the order in which the components are chosen, the result of *M–Procedure* (Δ) is Δ_1 in the first version and Δ_2 in the second version, with $\Delta_1 \not\approx \Delta_2$. (By the way, Δ_1 is the MDNF and Δ_2 is the PDNF of Δ.)

10 * Implementation of the M–Procedure

Notation

Let $\Delta = [\vee\gamma_1...\gamma_n]$ and $\Delta' = [\vee\gamma'_1...\gamma'_m]$ be two DNF's. We define

$$\Delta \vee \Delta' := [\vee\gamma_1...\gamma_n\gamma'_1...\gamma'_m]$$

Accordingly we write $\Delta_1 \vee ... \vee \Delta_d$ for more than two DNF's.

We are going to create the recursive algorithm *mrec* to implement the M–Procedure.
Let Δ be an arbitrary DNF that is going to be tansformed into a M2DNF.
First of all we split Δ into two DNF's

$$\overbrace{\underbrace{[\vee.....]}_{\Delta_0} \vee \underbrace{[\vee.....]}_{\Psi_2}}^{\Delta}$$

called Δ_0 and Ψ_2, so $\Delta = \Delta_0 \vee \Psi_2$. In particular

- Δ_0 is the part of Δ that isn't pairwise minimal yet, while
- Ψ_2 is the part of Δ, that is already a M2DNF.

At the beginning, of course there is

- $\Delta_0 = \Delta$
- $\Psi_2 = [\vee]$

As the procedure proceeds we attempt to increase Ψ_2, while Δ_0 is getting smaller, until finally $\Delta_0 = [\vee]$ and Ψ_2 is returned as the result of the procedure. Therefore in each single step we take the first component γ_1 of Δ_0 and for every component μ_i of Ψ_2 we minimize by means of $min(\gamma_1, \mu_i)$. We further split Ψ_2 into $\Psi_2 = \Delta_1 \vee \Delta_2$ where

- Δ_1 is the part of Ψ_2 that already is a M2DNF together with γ_1 (we call it Ψ_1) and
- Δ_2 is the remaining part of Ψ_2.

$$\overbrace{\underbrace{[\vee\overbrace{\gamma_1}^{\Psi_1}\gamma_2...\gamma_c]}_{\Delta_0} \vee \underbrace{\underbrace{[\vee\overbrace{\mu_1...\mu_n}^{\Psi_1}]}_{\Delta_1} \vee \underbrace{[\vee\mu_{n+1}...\mu_{n+m}]}_{\Delta_2}}_{\Psi_2}}^{\Delta}$$

At the beginning of this inner loop we initialize

- $\Delta_1 := [\vee]$
- $\Delta_2 := \Psi_2$

When we continue and if we reach $\Delta_2 = [\vee]$, we put $\Psi_2 := [\vee\gamma_1] \vee \Psi_2$ and erase γ_1 from Δ_0.
In the following full definition of *mrec* it is important to note that

- at each step the conditions for Ψ_1 and Ψ_2 are satisfied
- the whole expression $\Delta = \Delta_0 \vee \Delta_1 \vee \Delta_2$ is constantly getting shorter, or it remains the same while Ψ_1 or Ψ_2 is increasing

so that finally Ψ_2 is returned as the M2DNF of Δ.

Definition and theorem

Let

- $\Delta_0 = [\vee\gamma_1...\gamma_c]$ be a DNF with $c \geq 0$
- $\Delta_1 = [\vee\mu_1...\mu_n]$ be a M2DNF with $n \geq 0$
- $\Delta_2 = [\vee\mu_{n+1}...\mu_{n+m}]$ be a M2DNF with $m \geq 0$

such that

- $\Psi_1 := \left\{ \begin{array}{ll} [\vee\mu_1...\mu_n] & \text{if } c = 0 \\ [\vee\gamma_1\mu_1...\mu_n] & \text{if } c > 0 \end{array} \right\}$ is a M2DNF
- $\Psi_2 := \Delta_1 \vee \Delta_2 = [\vee\mu_1...\mu_n\mu_{n+1}...\mu_{n+m}]$ is a M2DNF.

Then we define

$$mrec\,(\Delta_0, \Delta_1, \Delta_2)$$

according to the table at the end of this chapter.
Then it holds that

- $mrec\,(\Delta_0, \Delta_1, \Delta_2) \Leftrightarrow \Delta_0 \vee \Delta_1 \vee \Delta_2$
- $mrec\,(\Delta_0, \Delta_1, \Delta_2)$ is a M2DNF.

The definition of *mrec* immediately allows to derive:

- An implementation of the M–Procedure:
  Definition
  M–Procedure $(\Delta) := mrec\,(\Delta, [\vee], [\vee])$
  for every DNF Δ.
  Theorem
  M–Procedure (Δ) is a M2DNF of Δ, for every DNF Δ.
- A M2DNF normalizer:
  Definition
  m2dnf $(\varphi) :=$ *M–Procedure* $(dnf\,(\varphi))$
  for every formula φ
  Theorem
  m2dnf (φ) is a M2DNF of φ, for every formula φ.
- A special fast disjunctor:
  Definition
  dnf–or–m2dnf $(\Delta_1, \Delta_2) := mrec\,(\Delta_1, [\vee], \Delta_2)$
  for every DNF Δ_1 and M2DNF Δ_2.
  Theorem
  dnf–or–m2dnf (Δ_1, Δ_2) is a M2DNF of $\Delta_1 \vee \Delta_2$, for every DNF Δ_1 and M2DNF Δ_2.

Proof of the correctness of *mrec*

We want to proof that *mrec* is well defined and does exactly what is told. This proof actually comprises four

parts:
(I) *mrec* is well–defined: in any case the recursion calls itself, the conditions of the definition must hold.
(II) *mrec* terminates: the algorithm has to terminate after a finite number of steps for every permitted input.
(III) $mrec\,(\Delta_0, \Delta_1, \Delta_2)$ is a M2DNF.
(IV) $mrec\,(\Delta_0, \Delta_1, \Delta_2) \Leftrightarrow \Delta_0 \vee \Delta_1 \vee \Delta_2$

When we start with $\Delta_0^0 := \Delta_0$, $\Delta_1^0 := \Delta_1$, and $\Delta_2^0 := \Delta_2$, the application of *mrec* initiates a sequence
$mrec\,(\Delta_0^0, \Delta_1^0, \Delta_2^0)$
$= mrec\,(\Delta_0^1, \Delta_1^1, \Delta_2^1)$
$= mrec\,(\Delta_0^2, \Delta_1^2, \Delta_2^2)$
$= ...$
$= mrec\,(\Delta_0^i, \Delta_1^i, \Delta_2^i)$
$= mrec\,(\Delta_0^{i+1}, \Delta_1^{i+1}, \Delta_2^{i+1})$
$= ...$

For the proof of (I) we will need two lemmas, where the first one was mentioned earlier, and the second one is quite obvious:
Lemma 1 Every DNF of length 0 or 1 is a M2DNF.
Lemma 2 If $\Psi \subset \Psi'$ and Ψ' is a M2DNF, so is Ψ.

(I) Proof that *mrec* is well–defined

Each step of the recursion has the general form
$mrec\,(\Delta_0^i, \Delta_1^i, \Delta_2^i)$
$= \left\{ \begin{array}{ll} \Delta_1^i \vee \Delta_2^i & \text{in case (1)} \\ mrec\,(\Delta_0^{i+1}, \Delta_1^{i+1}, \Delta_2^{i+1}) & \text{in case (2) or (3.x)} \end{array} \right.$
We need to show by induction that the recursion step $i+1$ is well defined, in other words that the conditions for Ψ_1^{i+1} and Ψ_2^{i+1} of the definition are met. In particular we have:

(2) $\Psi_1^{i+1} = \left\{ \begin{array}{ll} [\vee] & \text{if } c = 1 \\ [\vee\gamma_2] & \text{if } c > 1 \end{array} \right\}$ is a M2DNF (Lemma 1)
$\Psi_2^{i+1} = [\vee\gamma_1\mu_1...\mu_n] = \Psi_2^i$ is a M2DNF (induction)

(3.1) $\Psi_1^{i+1} = \left\{ \begin{array}{ll} [\vee] & \text{if } c = 1 \\ [\vee\gamma_2] & \text{if } c > 1 \end{array} \right\}$ is a M2DNF (Lemma 1)
$\Psi_2^{i+1} = [\vee\mu_1...\mu_n\mu_{n+1}...\mu_{n+m}] = \Psi_2^i$ is a M2DNF (induction)

(3.2) $\Psi_1^{i+1} = \left\{ \begin{array}{ll} [\vee] & \text{if } c = 1 \text{ and } m = 1 \\ [\vee\mu_{n+2}] & \text{if } c = 1 \text{ and } m > 1 \\ [\vee\gamma_2] & \text{if } c > 1 \end{array} \right\}$ is a M2DNF (Lemma 1)
$\Psi_2^{i+1} = [\vee\gamma_1\mu_1...\mu_n] = \Psi_1^i$ is a M2DNF (induction)

(3.3) $\Psi_1^{i+1} = \left\{ \begin{array}{ll} [\vee] & \text{if } c = 1 \\ [\vee\gamma_2] & \text{if } c > 1 \end{array} \right\}$ is a M2DNF (Lemma 1)
$\Psi_2^{i+1} = [\vee\mu_1...\mu_n\mu_{n+1}...\mu_{n+m}] = \Psi_2^i$ is a M2DNF (induction)

(3.4) $\Psi_1^{i+1} = [\vee\gamma_1\mu_1...\mu_n\mu_{n+1}]$ is a M2DNF because $[\vee\gamma_1\mu_1...\mu_n] = \Psi_1^i$ is a M2DNF (induction), $[\vee\mu_1...\mu_n\mu_{n+1}] \subseteq \Psi_2^i$ is a M2DNF (Lemma 2) and $[\vee\gamma_1\mu_{n+1}]$ is minimal. $\Psi_2^{i+1} = \Psi_2^i$ is a M2DNF (induction).

(3.5) $\Psi_1^{i+1} = \left\{ \begin{array}{ll} [\vee] & \text{if } c = 1 \\ [\vee\gamma_2] & \text{if } c > 1 \end{array} \right\}$ is a M2DNF (Lemma2)
$\Psi_2^{i+1} \subseteq \Psi_2^i$ is a M2DNF (Lemma 2)

(3.6) $\Psi_1^{i+1} = \left\{ \begin{array}{ll} [\vee\gamma_1'] & \text{if } c = 1 \\ [\vee\gamma_2] & \text{if } c > 1 \end{array} \right\}$ is a M2DNF (Lemma 1)
$\Psi_2^{i+1} = [\vee\gamma_1\mu_1...\mu_n] = \Psi_1^i$ is a M2DNF (induction)

(3.7) $\Psi_1^{i+1} = \left\{ \begin{array}{ll} [\vee\gamma_1'] & \text{if } c = 1 \\ [\vee\gamma_2] & \text{if } c > 1 \end{array} \right\}$ is a M2DNF (Lemma 1)
$\Psi_2^{i+1} = \Psi_1^i$ is a M2DNF (induction)

(3.8) same as (3.4)

(3.9) same as (3.4)

Thus, *mrec* is well–defined.

(II) Proof that *mrec* terminates

We define the following strict order[16] $\succ$ on the set of all DNF triples:

$$(\Delta_0, \Delta_1, \Delta_2) \succ (\Delta_0', \Delta_1', \Delta_2')$$
$$\text{iff} \left(\begin{array}{c} vol\,(\Delta_0' \vee \Delta_1' \vee \Delta_2') < vol\,(\Delta_0 \vee \Delta_1 \vee \Delta_2) \\ \text{or} \\ \left(\begin{array}{c} vol\,(\Delta_0' \vee \Delta_1' \vee \Delta_2') = vol\,(\Delta_0 \vee \Delta_1 \vee \Delta_2) \\ \text{and} \\ lg\,(\Delta_0') < lg\,(\Delta_0) \end{array} \right) \\ \text{or} \\ \left(\begin{array}{c} vol\,(\Delta_0' \vee \Delta_1' \vee \Delta_2') = vol\,(\Delta_0 \vee \Delta_1 \vee \Delta_2) \\ \text{and} \\ lg\,(\Delta_0') = lg\,(\Delta_0) \\ \text{and} \\ lg\,(\Delta_2') < lg\,(\Delta_2) \end{array} \right) \end{array} \right)$$

Obviously, every chain
$(\Delta_0^0, \Delta_1^0, \Delta_2^0) \succ (\Delta_0^1, \Delta_1^1, \Delta_2^1) \succ ... \succ (\Delta_0^i, \Delta_1^i, \Delta_2^i) \succ ...$
has to terminate. And because the following table shows that
$$(\Delta_0^i, \Delta_1^i, \Delta_2^i) \succ (\Delta_0^{i+1}, \Delta_1^{i+1}, \Delta_2^{i+1})$$
indeed for every i for which the triple $i+1$ is defined,

[16] asymmetric and transitive

we have proven that $mrec\,(\Delta_0, \Delta_1, \Delta_2)$ terminates for every permitted input.

Case	$vol\,(\Delta_0^{i+1} \vee \Delta_1^{i+1} \vee \Delta_2^{i+1})$	$lg\,(\Delta_0^{i+1})$	$lg\,(\Delta_2^{i+1})$
(2)	$vol\,(\Delta_0^i \vee \Delta_1^i \vee \Delta_2^i)$	$lg\,(\Delta_0^i) - 1$	—
(3.1)	$vol\,(\Delta_0^i \vee \Delta_1^i \vee \Delta_2^i) - lg\,(\gamma_1)$ with $lg\,(\gamma_1) > 0$	—	—
(3.2)	$vol\,(\Delta_0^i \vee \Delta_1^i \vee \Delta_2^i) - lg\,(\mu_{n+1})$ with $lg\,(\mu_{n+1}) > 0$	—	—
(3.3)	$vol\,(\Delta_0^i \vee \Delta_1^i \vee \Delta_2^i) - lg\,(\gamma_1)$ with $lg\,(\gamma_1) > 0$	—	—
(3.4)	$vol\,(\Delta_0^i \vee \Delta_1^i \vee \Delta_2^i)$	$lg\,(\Delta_0^i)$	$lg\,(\Delta_2^i) - 1$
(3.5)	$vol\,(\Delta_0^i \vee \Delta_1^i \vee \Delta_2^i) - lg\,(\gamma_1) - 1$	—	—
(3.6)	$vol\,(\Delta_0^i \vee \Delta_1^i \vee \Delta_2^i) - 1$	—	—
(3.7)	$vol\,(\Delta_0^i \vee \Delta_1^i \vee \Delta_2^i) - 1$	—	—
(3.8)	$vol\,(\Delta_0^i \vee \Delta_1^i \vee \Delta_2^i)$	$lg\,(\Delta_0^i)$	$lg\,(\Delta_2^i) - 1$
(3.9)	$vol\,(\Delta_0^i \vee \Delta_1^i \vee \Delta_2^i)$	$lg\,(\Delta_0^i)$	$lg\,(\Delta_2^i) - 1$

(III) Proof that $mrec\,(\Delta_0, \Delta_1, \Delta_2)$ is a M2DNF

The algorithm terminates if and only if $\Delta_0 = [\vee]$. And then it returnes $\Delta_1 \vee \Delta_2 =: \Psi_2$ which is a M2DNF.

(IV) Proof that $mrec\,(\Delta_0, \Delta_1, \Delta_2) \Leftrightarrow \Delta_0 \vee \Delta_1 \vee \Delta_2$

In each recursion step the components only change their order or they are replaced by equivalent components. All these operations don't change the semantic.

end of proof.

$$
\begin{array}{l}
mrec\,([\vee\gamma_1...\gamma_c],[\vee\mu_1...\mu_n],[\vee\mu_{n+1}...\mu_{n+m}]) \\
:= \begin{cases}
[\vee\mu_1...\mu_n\mu_{n+1}...\mu_{n+m}] & \text{if } c=0 \qquad (1) \\
mrec\,([\vee\gamma_2...\gamma_c],[\vee],[\vee\gamma_1\mu_1...\mu_n]) & \text{if } c>0 \text{ and } m=0 \qquad (2) \\
mrec\,([\vee\gamma_2...\gamma_c],[\vee],[\vee\mu_1...\mu_n\mu_{n+1}...\mu_{n+m}]) & \text{if } c>0 \text{ and } m>0 \text{ and } \\
 & xmin\,(\gamma_1,\mu_{n+1})=(\text{“N000”},[\vee\mu_{n+1}]) \qquad (3.1) \\
mrec\,([\vee\gamma_2...\gamma_c\mu_{n+2}...\mu_{n+m}],[\vee],[\vee\gamma_1\mu_1...\mu_n]) & \text{if } c>0 \text{ and } m>0 \text{ and } \\
 & xmin\,(\gamma_1,\mu_{n+1})=(\text{“N00P”},[\vee\gamma_1]) \qquad (3.2) \\
mrec\,([\vee\gamma_2...\gamma_c],[\vee],[\vee\mu_1...\mu_n\mu_{n+1}...\mu_{n+m}]) & \text{if } c>0 \text{ and } m>0 \text{ and } \\
 & xmin\,(\gamma_1,\mu_{n+1})=(\text{“N0P0”},[\vee\mu_{n+1}]) \qquad (3.3) \\
mrec\,([\vee\gamma_1...\gamma_c],[\vee\mu_1...\mu_n\mu_{n+1}],[\vee\mu_{n+2}...\mu_{n+m}]) & \text{if } c>0 \text{ and } m>0 \text{ and } \\
 & xmin\,(\gamma_1,\mu_{n+1})=(\text{“N0PP”},[\vee\gamma_1\mu_{n+1}]) \qquad (3.4) \\
mrec\,([\vee\gamma_2...\gamma_c\gamma'],[\vee],[\vee\mu_1...\mu_n\mu_{n+2}...\mu_{n+m}]) & \text{if } c>0 \text{ and } m>0 \text{ and } \\
 & xmin\,(\gamma_1,\mu_{n+1})=(\text{“N100”},[\vee\gamma']) \qquad (3.5) \\
mrec\,([\vee\gamma_2...\gamma_c\mu'_{n+1}\mu_{n+2}...\mu_{n+m}],[\vee],[\vee\gamma_1\mu_1...\mu_n]) & \text{if } c>0 \text{ and } m>0 \text{ and } \\
 & xmin\,(\gamma_1,\mu_{n+1})=(\text{“N10P”},[\vee\gamma_1\mu'_{n+1}]) \qquad (3.6) \\
mrec\,([\vee\gamma_2...\gamma_c\gamma'_1],[\vee],[\vee\mu_1...\mu_n\mu_{n+1}...\mu_{n+m}]) & \text{if } c>0 \text{ and } m>0 \text{ and } \\
 & xmin\,(\gamma_1,\mu_{n+1})=(\text{“N1P0”},[\vee\gamma'_1\mu_{n+1}]) \qquad (3.7) \\
mrec\,([\vee\gamma_1...\gamma_c],[\vee\mu_1...\mu_n\mu_{n+1}],[\vee\mu_{n+2}...\mu_{n+m}]) & \text{if } c>0 \text{ and } m>0 \text{ and } \\
 & xmin\,(\gamma_1,\mu_{n+1})=(\text{“N1PP”},[\vee\gamma_1\mu_{n+1}]) \qquad (3.8) \\
mrec\,([\vee\gamma_1...\gamma_c],[\vee\mu_1...\mu_n\mu_{n+1}],[\vee\mu_{n+2}...\mu_{n+m}]) & \text{if } c>0 \text{ and } m>0 \text{ and } \\
 & xmin\,(\gamma_1,\mu_{n+1})=(\text{“NMNN”},[\vee\gamma_1\mu_{n+1}]) \qquad (3.9)
\end{cases}
\end{array}
$$

11 The completeness theorem and the P–Procedure

The completeness theorem

The M–Procedure generates a M2DNF of a given DNF. But actually we are much more interested in a **P–Procedure** to derive the PDNF. In this chapter we will formulate the **completeness theorem** that gives us necessary and sufficient criteria for which a M2DNF is a PDNF by means of the covering concept, and that will be the foundation for the P–Procedure.

Definition

A DNF Δ is called **factor complete** or **f–complete**, iff $\gamma \sqsubseteq \Delta$ for every factor γ of Δ.
In other words, Δ is f–complete iff every prime factor of Δ is a component of Δ (i.e. $\Delta_p \subseteq \Delta$ where Δ_p is the PDNF of Δ).

Examples

- $[\vee]$ is f–complete in a trivial way, because it has no factor.
- $[\vee[\wedge]]$ is f–complete, because every NLC γ is a factor, $\gamma \Rightarrow [\wedge]$ and so $\gamma \sqsubseteq [\vee[\wedge]]$.

Definition

For a DNF Δ we define $CPrimes(\Delta)$ to be the DNF which components are the c–prime factors of all the complementary and minimal pairs of components of Δ. Thus:[17]

Algorithm $CPrimes([\vee\gamma_1 ... \gamma_n])$
begin
  $\Delta' := [\vee]$;
  for $i := 1$ *to* $n-1$ *do*
    for $j := i+1$ *to* n *do*
      if $[\vee\gamma_i\gamma_j]$ is complementary minimal with the c–prime factor γ'
      then $\Delta' := \Delta' \vee [\vee\gamma']$;
  return Δ' ;
end.

Example

$\Delta = [\vee[\wedge A\overline{B}][\wedge B\overline{C}][\wedge\overline{A}C]]$ contains three pairs of components that are complementary minimal, namely

- $[\wedge A\overline{B}]$ and $[\wedge B\overline{C}]$. Their c–prime factor is $[\wedge A\overline{C}]$.
- $[\wedge A\overline{B}]$ and $[\wedge\overline{A}C]$ with c–prime factor $[\wedge\overline{B}C]$.
- $[\wedge B\overline{C}]$ and $[\wedge\overline{A}C]$ with c–prime factor $[\wedge\overline{A}B]$.

So $CPrimes(\Delta) = [\vee[\wedge A\overline{C}][\wedge\overline{B}C][\wedge\overline{A}B]]$

[17] Again $\Delta_1 \vee \Delta_2$ denotes the simple concatenation of two DNF's.

Definition

A DNF Δ is said to be **complementary prime complete** or **c–complete** iff $CPrimes(\Delta) \sqsubseteq \Delta$.

Example

Again let $\Delta = [\vee[\wedge A\overline{B}][\wedge B\overline{C}][\wedge\overline{A}C]]$. Δ is not c–complete because, for example $[\wedge A\overline{C}] \in CPrimes(\Delta)$, but $[\wedge A\overline{C}] \not\sqsubseteq \Delta$ and thus $CPrimes(\Delta) \not\sqsubseteq \Delta$.

Theorem

Every f–complete DNF is c–complete.

Proof

Easy, since every c–prime factor of a pair of complementary minimal components is a factor.

Corollary

- $[\vee]$ and $[\vee[\wedge]]$ are c–complete.
- Every PDNF is c–complete.

Now we are able to state the central completeness theorem:

Theorem (Completeness)

For every DNF Δ the following statements are equivalent
(i) Δ is a PDNF
(ii) Δ is a f–complete M2DNF
(iii) Δ is a c–complete M2DNF

To proof this, we first proof two lemmata:

Lemma 1

For every DNF Δ and NLC γ with $At(\Delta) \subseteq At(\gamma)$ there is $\gamma \Rightarrow \Delta$ iff $\gamma \sqsubseteq \Delta$.

Proof

Let $\gamma' \in \Delta$. With $At(\Delta) \subseteq At(\gamma)$ we have $At(\gamma') \subseteq At(\gamma)$. So there are exactly two different possibilities:

- either $\gamma' \subseteq \gamma$, and then $\gamma \Rightarrow \gamma'$, and so $\gamma \sqsubseteq \Delta$
- or $\gamma' \not\subseteq \gamma$, and then there must be (at least) one literal $\lambda \in \gamma'$ with $\overline{\lambda} \in \gamma$, so that $[\wedge\gamma\gamma'] \Leftrightarrow 0$

We can now proof that $\gamma \Rightarrow \Delta$ iff $\gamma \sqsubseteq \Delta$

- Suppose that $\gamma \Rightarrow \Delta$. Because $\gamma \not\Leftrightarrow 0$ for every NLC γ, there must be at least one $\gamma' \in \Delta$ such that $[\wedge\gamma\gamma'] \not\Leftrightarrow 0$. But this implies that $\gamma' \subseteq \gamma$ and $\gamma \sqsubseteq \Delta$.
- Suppose that $\gamma \sqsubseteq \Delta$. Then there is a $\gamma' \in \Delta$ with $\gamma \Rightarrow \gamma'$ by definition, and so $\gamma \Rightarrow \gamma' \Rightarrow \Delta$.

Thus: $\gamma \Rightarrow \Delta$ iff $\gamma \sqsubseteq \Delta$.

Lemma 2

If Δ is a c–complete M2DNF and γ a factor of Δ, then $\gamma \sqsubseteq \Delta$.

Proof

Suppose $\gamma = [\wedge\lambda_1...\lambda_n]$. We put $\{\alpha_1, ..., \alpha_a\} := At\ (\Delta) - At\ (\gamma)$. (The α_1,...,α_a are uniquely defined if we demand that $\alpha_1 < ... < \alpha_a$. But in fact their order is of no importance here.)

We are going to proof that $\gamma \sqsubseteq \Delta$ by induction on a.

- Suppose $a = 0$.
  Then $At\,(\Delta) \subseteq At\,(\gamma)$. And since γ is a factor and $\gamma \Rightarrow \Delta$, there is $\gamma \sqsubseteq \Delta$ according to lemma 1.
- Suppose $a > 0$.
  We put $\gamma_0 := [\dot\wedge\lambda_1...\lambda_n\overline{\alpha_1}]$ and $\gamma_1 := [\dot\wedge\lambda_1...\lambda_n\alpha_1]$ so that $\gamma = [\wedge\lambda_1...\lambda_n] \Leftrightarrow [\vee\gamma_0\gamma_1]$. By virtue of the induction it holds that $\gamma_0 \sqsubseteq \Delta$ and $\gamma_1 \sqsubseteq \Delta$. So there must be $\gamma_0', \gamma_1' \in \Delta$ such that $\gamma_0 \Rightarrow \gamma_0'$ (i.e. $\gamma_0' \subseteq \gamma_0$) and $\gamma_1 \Rightarrow \gamma_1'$ (i.e. $\gamma_1' \subseteq \gamma_1$). We distinguish the following three cases:
  - First case: $\overline{\alpha_1} \notin \gamma_0'$.
    Then $\gamma_0' \subseteq \gamma_0 \subseteq \gamma$ and so $\gamma \Rightarrow \gamma_0'$, and that means that $\gamma \sqsubseteq \Delta$ because $\gamma_0' \in \Delta$
  - Second case: $\alpha_1 \notin \gamma_1'$.
    Similar to the first case, $\gamma_0' \subseteq \gamma_0 \subseteq \gamma$ and so again $\gamma \sqsubseteq \Delta$.
  - Third case: $\overline{\alpha_1} \in \gamma_0'$ and $\alpha_1 \in \gamma_1'$.
    In this case γ_0' and γ_1' have exactly this one complementary literal, and that makes them complementary NLC's. And since Δ is a M2DNF and γ_0' and γ_1' are components of Δ, they must be a minimal pair. So, $\gamma_0' = [\dot\wedge\pi_1...\pi_p\overline{\alpha_1}\sigma_1...\sigma_s]$ and $\gamma_1' = [\dot\wedge\pi_1...\pi_p\alpha_1\tau_1...\tau_t]$ with $p \geq 0$, $s > 0$, $t > 0$, $\{\sigma_1, ..., \sigma_s\} \cap \{\tau_1, ..., \tau_t\} = \{\}$ and $\{\pi_1, ..., \pi_p, \sigma_1, ..., \sigma_s, \tau_1, ..., \tau_t\} \subseteq \{\lambda_1, ..., \lambda_n\}$. $\gamma_c' := [\dot\wedge\pi_1...\pi_p\sigma_1...\sigma_s\tau_1...\tau_t]$ is the c–prime factor of γ_0' and γ_1' and so $\gamma_c' \subseteq \gamma$, that is $\gamma \Rightarrow \gamma_c'$. Δ is c–complete, so $\gamma_c' \sqsubseteq \Delta$ and that means that $\gamma \sqsubseteq \Delta$.

  Thus, $\gamma \sqsubseteq \Delta$ in each case.

Now we are able to give the whole proof of the completeness theorem:

Proof (Completeness Theorem)

(i) implies (ii)

Every PDNF is both f–complete and pairwise minimal, as we already proved.

(ii) implies (i)

Let Δ be a f–complete M2DNF. Suppose that Δ is not a PDNF. So there must be a component $\gamma \in \Delta$ that is not a prime factor of Δ. Since Δ is f–complete, there must be a $\gamma' \in \Delta$ with $\gamma \dot\Rightarrow \gamma'$. But in that case $min\ (\gamma, \gamma') = [\vee\gamma']$ so that Δ wouldn't be minimal, different to the initial assumption. Thus, every f–complete M2DNF must be a PDNF.

(ii) implies (iii)

Let Δ be a f–complete M2DNF. If there is a complementary and minimal pair γ_L and γ_R of components in Δ and their c–prime factor is γ, then $\gamma \Rightarrow [\vee\gamma_L\gamma_R] \Rightarrow \Delta$, so that γ is a factor of Δ. Because Δ is f–complete it holds that $\gamma \sqsubseteq \Delta$. So Δ is c–complete.

(iii) implies (ii)

Let Δ be a c–complete M2DNF and γ be any factor of Δ. According to lemma 2 there is $\gamma \sqsubseteq \Delta$. So Δ is f–complete.

Summary: (i) iff (ii) iff (iii)

The P–Procedure

Lemma 3

$[\vee\gamma_L\gamma_R] \sqsubseteq min\ (\gamma_L, \gamma_R)$, for any two NLC's γ_L and γ_R

Proof

$min\ (\gamma_L, \gamma_R)$ is

- either of length 1, thus of the form $[\vee\gamma']$, and then $\gamma_L \Rightarrow \gamma'$ and $\gamma_R \Rightarrow \gamma'$
- or of length 2, thus of the form $[\vee\gamma_L'\gamma_R']$, and then $\gamma_L \Rightarrow \gamma_L'$ and $\gamma_R \Rightarrow \gamma_R'$.

So there is always $[\vee\gamma_L\gamma_R] \sqsubseteq min\ (\gamma_L, \gamma_R)$.

Lemma 4

$\Delta \sqsubseteq$ *M–Procedure* (Δ) for every DNF Δ.

Proof

In every step of the M–Procedure two components γ_L and γ_R of Δ are replaced by the components of $min\ (\gamma_L, \gamma_R)$. Let Δ' be the result of this replacement. Because $[\vee\gamma_L\gamma_R] \sqsubseteq min\ (\gamma_L, \gamma_R)$ (Lemma 3) there is $\Delta \sqsubseteq \Delta'$ and thus $\Delta \sqsubseteq ... \sqsubseteq$ *M–Procedure* (Δ). $\sqsubseteq$ is transitive, so $\Delta \sqsubseteq$ *M–Procedure* (Δ).

Lemma 5

Let Δ_1 be a DNF and Δ_2 be a M2DNF. If *M–Procedure* $(\Delta_1 \vee \Delta_2) \approx \Delta_2$, then $\Delta_1 \sqsubseteq \Delta_2$.

Proof

Because *M–Procedure* $(\Delta_1 \vee \Delta_2) \approx \Delta_2$ and Lemma 4, there is $\Delta_1 \vee \Delta_2 \sqsubseteq \Delta_2$ and thus $\Delta_1 \sqsubseteq \Delta_2$.

Now, we have all the ingredients to implement the P–Procedure that transforms any DNF into its PDNF. It

is standing on two feet: the M–Procedure and the completeness theorem, that guarantees us that a M2DNF Δ is a PDNF iff *CPrimes* $(\Delta) \sqsubseteq \Delta$. Because of lemma 5 it follows that for a DNF Δ and $\Pi :=$ *CPrimes* (Δ) holds:

- if *M–Procedure* $(\Pi \vee \Delta) \approx \Delta$, then Δ is a M2DNF with $\Pi \sqsubseteq \Delta$ and thus Δ is a PDNF.

We use this to implement the P–Procedure that generates for a given DNF Δ a sequence $\Delta^1, \Pi^1, \Delta^2, \Pi^2, ..., \Delta^{i-1}, \Pi^{i-1}, \Delta^i, \Pi^i, ...$ where

- $\Delta^1 :=$ *M–Procedure* (Δ)
- $\Pi^i :=$ *CPrimes* (Δ^i)
- $\Delta^{i+1} :=$ *M–Procedure* $(\Pi^i \vee \Delta^i)$

We know that $\Delta \Leftrightarrow \Delta^1 \Leftrightarrow ... \Leftrightarrow \Delta^i \Leftrightarrow ....$ So, if there is a fixpoint z with $\Delta^z \approx \Delta^{z+1}$ (and thus $\Delta^z \approx \Delta^{z+n}$ for all n), Δ^z has to be the PDNF of Δ. Thus, the only thing that is left to be proven is that such a z actually exists for every DNF Δ.[18] But first we formulate this implementation of the P–Procedure properly:

Algorithm P–Procedure (Δ)
begin
  $i := 1$;
  $\Delta^1 :=$ *M–Procedure* (Δ) ;
  repeat
    $\Pi^i :=$ *CPrimes* (Δ^i) ;
    $i := i + 1$;
    $\Delta^i :=$ *M–Procedure* $(\Pi^{i-1} \vee \Delta^{i-1})$;
  until $\Delta^i \approx \Delta^{i-1}$;
  return Δ^i ;
end.

The following theorem will enable the proof that the P–Procedure terminates for every input DNF Δ.

Theorem
$\sqsubseteq$ is a (partial) order on the set of all M2DNF's in the following sense:

- $\sqsubseteq$ is reflexive
- $\sqsubseteq$ is transitive
- $\sqsubseteq$ is anti–symmetric: $\Delta_1 \sqsubseteq \Delta_2$ and $\Delta_2 \sqsubseteq \Delta_1$ implies $\Delta_1 \approx \Delta_2$

Proof
We stated earlier that $\sqsubseteq$ is a quasi–order on the set of DNF's. It remains to show the anti–symmetry: Assume that $\Delta_1 \sqsubseteq \Delta_2$ and $\Delta_2 \sqsubseteq \Delta_1$, but $\Delta_1 \not\approx \Delta_2$. That would mean the existence of a $\gamma \in \Delta_1$ with $\gamma \notin \Delta_2$ (or the other way round). Since $\Delta_1 \sqsubseteq \Delta_2$ there must be a $\gamma' \in \Delta_2$ with $\gamma \Rightarrow \gamma'$. On the other hand $\Delta_2 \sqsubseteq \Delta_1$ implies the existence of a $\gamma'' \in \Delta_1$ with $\gamma' \Rightarrow \gamma''$. But then $\gamma \Rightarrow \gamma''$, and $min\ (\gamma, \gamma'') = [\vee\gamma'']$, and that contradicts the assumption that Δ_1 is pairwise minimal.

Proof (Termination of the P–Procedure)
The sequence $\Delta^1, \Delta^2, ..., \Delta^{i-1}, \Delta^i, ...$ that the P–Procedure generates, is finite because:
(i) The set $\{\Delta^1, \Delta^2, ..., \Delta^{i-1}, \Delta^i, ...\}$ is finite. For every DNF Δ there is *At* (*CPrimes* $(\Delta)) \subseteq$ *At* (Δ) and *At* (*M–Procedure* $(\Delta)) \subseteq$ *At* (Δ) and so *At* $(\Delta^i) \subseteq$ *At* (Δ) for every i. But since there is only a finite number of NLC's on a finite atom set and M2DNF's don't have any two equal components, there is only a finite number of M2DNF's on a finite atom set.
(ii) $\Delta^1 \sqsubseteq \Delta^2 \sqsubseteq \Delta^3 \sqsubseteq ...$ because for every $i = 1, 2, 3, ...$

- $\Delta^i \sqsubseteq \Pi^i \vee \Delta^i$ following immediately from $\Delta^i \subseteq \Pi^i \vee \Delta^i$
- $\Pi^i \vee \Delta^i \sqsubseteq$ *M–Procedure* $(\Pi^i \vee \Delta^i)$ according to lemma 4

and so, since $\sqsubseteq$ is transitive:

- $\Delta^i \sqsubseteq$ *M–Procedure* $(\Pi^i \vee \Delta^i) =: \Delta^{i+1}$

Thus, for every sequence $\Delta^1 \sqsubseteq \Delta^2 \sqsubseteq \Delta^3 \sqsubseteq ...$ there must be a z such that $\Delta^z \approx \Delta^{z+1} \approx \Delta^{z+2} \approx ...$ and this is the condition for the algorithm to terminate.

We summarize:

Theorem
P–Procedure (Δ) is the PDNF of Δ, for every DNF Δ.

[18] There is even an apriori upper bound for z, namely the number of atoms in Δ. But in average applications z is actually very small and quite independent of the number of atoms.

12 * Implementation of the P–Procedure

Our implementation of the P–Procedure comprised three operations: the M–Procedure, the generation of *CPrimes* and the $\approx$–decision. But instead of performing these operations subsequently, we can merge them into one single step and increase the speed of the algorithm considerably. Instead of first constructing the M–Procedure by using *min* and afterwards generating all the c–prime factors, we could immediately combine these by using *prim* instead of *min*, since "*prim* = *min* + c–prime". We define a recursive algorithm *prec* to transform the P–Procedure similar to the algorithm *mrec* by extending its three arguments to four. Δ_3 is the collection of all the c–prime factors of $[\vee\gamma_1\mu_1]$,$[\vee\gamma_1\mu_2]$,...,$[\vee\gamma_1\mu_n]$. If required Δ_3 will be attached to Δ_0 such that the property $\mathit{CPrimes}\ (\Psi_2) \sqsubseteq \Delta_0 \vee \Delta_1 \vee \Delta_2 \vee \Delta_3$ is maintained in every step of the process, so that in the end, when Ψ_2 is returned, it is guaranteed to be c–complete.

Definition and Theorem

Let

- $\Delta_0 = [\vee\gamma_1...\gamma_n]$ be a DNF with $c \geq 0$
- $\Delta_1 = [\vee\mu_1...\mu_n]$ be a M2DNF with $n \geq 0$
- $\Delta_2 = [\vee\mu_{n+1}...\mu_{n+m}]$ be a M2DNF with $m \geq 0$
- $\Delta_3 = [\vee\pi_1...\pi_p]$ a DNF with $p \geq 0$
- $\Delta := \Delta_0 \vee \Delta_1 \vee \Delta_2 \vee \Delta_3$

such that

- $\Psi_1 := \left\{ \begin{array}{ll} [\vee\mu_1...\mu_n] & \text{if } c = 0 \\ [\vee\gamma_1\mu_1...\mu_n] & \text{if } c > 0 \end{array} \right\}$ is a M2DNF
- $\Psi_2 := \Delta_1 \vee \Delta_2 = [\vee\mu_1...\mu_n\mu_{n+1}...\mu_{n+m}]$ is a M2DNF
- $\mathit{CPrimes}\ (\Psi_2) \sqsubseteq \Delta$

Then we define

$$\mathit{prec}\ (\Delta_0, \Delta_1, \Delta_2, \Delta_3)$$

according to the table at the end of this chapter.

Then it holds that

- $\mathit{prec}\ (\Delta_0, \Delta_1, \Delta_2, \Delta_3) \Leftrightarrow \Delta$
- $\mathit{prec}\ (\Delta_0, \Delta_1, \Delta_2, \Delta_3)$ is a PDNF

$$\overbrace{\underbrace{[\vee\overbrace{\gamma_1}^{\Psi_1}\gamma_2...\gamma_c]}_{\Delta_0} \vee \underbrace{[\vee\overbrace{\mu_1...\mu_n}^{\Psi_1}]}_{\Delta_1} \vee \underbrace{[\vee\mu_{n+1}...\mu_{n+m}]}_{\Delta_2}}^{\Psi_2} \vee \underbrace{[\vee\pi_1...\pi_p]}_{\Delta_3}}^{\Delta}$$

The definition of *prec* immediately allows to derive:

- A new and faster implementation of the P–Procedure:
  Definition
  P-Procedure $(\Delta) :=$ *prec* $(\Delta, [\vee], [\vee], [\vee])$ for every DNF Δ.
  Theorem
  P-Procedure (Δ) is the PDNF of Δ, for every DNF Δ.
- A PDNF normalizer:
  Definition
  pdnf $(\varphi) :=$ *P-Procedure* (*dnf* (φ)) for every formula φ.
  Theorem
  pdnf (φ) is the PDNF of φ, for every formula φ.
- A special disjunctor:
  Definition
  dnf-or-pdnf $(\Delta_1, \Delta_2) :=$ *prec* $(\Delta_1, [\vee], \Delta_2, [\vee])$ for every DNF Δ_1 and PDNF Δ_2.
  Theorem
  dnf-or-pdnf (Δ_1, Δ_2) is the PDNF of $\Delta_1 \vee \Delta_2$, for every DNF Δ_1 and PNDF Δ_2.

Proof (Correctness of *prec*)

The correctness of *mrec* and the P–Procedure (in the first version) have been proved thoroughly. The correctness of *prec* can be proven analogue to *mrec*. Only the fact that *prec* always terminates deserves additional attention, and this is guaranteed by the correctness proof of the P–Procedure.

$$
\begin{array}{l}
prec\ ([\vee\gamma_1 ... \gamma_c], [\vee\mu_1 ... \mu_n], [\vee\mu_{n+1} ... \mu_{n+m}], [\vee\pi_1 ... \pi_p]) \\
\\
:= \begin{cases}
[\vee\mu_1 ... \mu_n \mu_{n+1} ... \mu_{n+m}] & \text{if } c = 0 \text{ and } p = 0 \\
\\
prec\ ([\vee\pi_1 ... \pi_p], [\vee], [\vee\mu_1 ... \mu_n \mu_{n+1} ... \mu_{n+m}], [\vee]) & \text{if } c = 0 \text{ and } p > 0 \\
\\
prec\ ([\vee\gamma_2 ... \gamma_c \pi_1 ... \pi_p], [\vee], [\vee\gamma_1 \mu_1 ... \mu_n], [\vee]) & \text{if } c > 0 \text{ and } m = 0 \\
\\
prec\ ([\vee\gamma_2 ... \gamma_c], [\vee], [\vee\mu_1 ... \mu_n \mu_{n+1} ... \mu_{n+m}], [\vee]) & \text{if } c > 0 \text{ and } m > 0 \text{ and} \\
& xprim\ (\gamma_1, \mu_{n+1}) = (\text{“N000”}, [\vee\mu_{n+1}]) \\
prec\ ([\vee\gamma_2 ... \gamma_c \mu_{n+2} ... \mu_{n+m} \pi_1 ... \pi_p], [\vee], [\vee\gamma_1 \mu_1 ... \mu_n], [\vee]) & \text{if } c > 0 \text{ and } m > 0 \text{ and} \\
& xprim\ (\gamma_1, \mu_{n+1}) = (\text{“N00P”}, [\vee\gamma_1]) \\
prec\ ([\vee\gamma_2 ... \gamma_c], [\vee], [\vee\mu_1 ... \mu_n \mu_{n+1} ... \mu_{n+m}], [\vee]) & \text{if } c > 0 \text{ and } m > 0 \text{ and} \\
& xprim\ (\gamma_1, \mu_{n+1}) = (\text{“N0P0”}, [\vee\mu_{n+1}]) \\
prec\ ([\vee\gamma_1 ... \gamma_c], [\vee\mu_1 ... \mu_n \mu_{n+1}], [\vee\mu_{n+2} ... \mu_{n+m}], [\vee\pi_1 ... \pi_p]) & \text{if } c > 0 \text{ and } m > 0 \text{ and} \\
& xprim\ (\gamma_1, \mu_{n+1}) = (\text{“N0PP”}, [\vee\gamma_1 \mu_{n+1}]) \\
prec\ ([\vee\gamma_2 ... \gamma_c \gamma'], [\vee], [\vee\mu_1 ... \mu_n \mu_{n+2} ... \mu_{n+m}], [\vee]) & \text{if } c > 0 \text{ and } m > 0 \text{ and} \\
& xprim\ (\gamma_1, \mu_{n+1}) = (\text{“N100”}, [\vee\gamma']) \\
prec\ ([\vee\gamma_2 ... \gamma_c \mu'_{n+1} \mu_{n+2} ... \mu_{n+m} \pi_1 ... \pi_p], [\vee], [\vee\gamma_1 \mu_1 ... \mu_n], [\vee]) & \text{if } c > 0 \text{ and } m > 0 \text{ and} \\
& xprim\ (\gamma_1, \mu_{n+1}) = (\text{“N10P”}, [\vee\gamma_1 \mu'_{n+1}]) \\
prec\ ([\vee\gamma_2 ... \gamma_c \gamma'_1], [\vee], [\vee\mu_1 ... \mu_n \mu_{n+1} ... \mu_{n+m}], [\vee]) & \text{if } c > 0 \text{ and } m > 0 \text{ and} \\
& xprim\ (\gamma_1, \mu_{n+1}) = (\text{“N1P0”}, [\vee\gamma'_1 \mu_{n+1}]) \\
prec\ ([\vee\gamma_1 ... \gamma_c], [\vee\mu_1 ... \mu_n \mu_{n+1}], [\vee\mu_{n+2} ... \mu_{n+m}], [\vee\pi\pi_1 ... \pi_p]) & \text{if } c > 0 \text{ and } m > 0 \text{ and} \\
& xprim\ (\gamma_1, \mu_{n+1}) = (\text{“N1PP”}, [\vee\gamma_1 \mu_{n+1} \pi]) \\
prec\ ([\vee\gamma_1 ... \gamma_c], [\vee\mu_1 ... \mu_n \mu_{n+1}], [\vee\mu_{n+2} ... \mu_{n+m}], [\vee\pi_1 ... \pi_p]) & \text{if } c > 0 \text{ and } m > 0 \text{ and} \\
& xprim\ (\gamma_1, \mu_{n+1}) = (\text{“NMNN”}, [\vee\gamma_1 \mu_{n+1}])
\end{cases}
\end{array}
$$

13 * Implementation: The PDNF and the PCNF system

Now we are able to gather the pieces to present the final PDNF and PCNF systems:

- Due to the duality principle we can implement the PCNF system dual to the PDNF system. All the auxiliary algorithms (*jrec*, *prec*, *xprim*, *prim*, and *xprec*) are exactly the ones already defined in earlier chapters, except that they make use of the junctor variable $*$, to serve for the PCNF system as well as for the PDNF system.
- In the last chapter we implemented the PDNF normalizer as follows:

  $$\begin{aligned} pdnf\,(\varphi) &:= P\text{–}Procedure\,(dnf\,(\varphi)) \\ &:= prec\,(dnf\,(\varphi), [\vee], [\vee], [\vee]) \end{aligned}$$

  But *dnf* makes repeatedly use of the distributive law and multiple occurings of components are not deleted. The resulting length of the DNF can explode, even for a simple looking φ. And instead of reducing this DNF in a second step by means of *prec*, it is in general more efficient[19] to define *pdnf* as a recursion that operates itself only on PDNF's. Besides, that is more elegant. So we don't define the final PDNF system as an extension of the earlier defined DNF system, but design it as an independent one, only similar to the DNF system.

Finally note that:

- The following systems are only precanonic. We mentioned earlier how this can easily be extended to a canonic system. One can implement
  - *ord* (Φ) a standard sorting algorithm that rearranges the components of the DNF or CNF Φ according to the given (or any other) linear order on the set of NLC's and NLD's, so that the result is an ordered DNF or CNF.
  - $opdnf\,(\varphi) := ord\,(pdnf\,(\varphi))$ for the unique ordered PDNF and $opcnf\ (\varphi) := ord\ (pcnf\ (\varphi))$ for the unique ordered PCNF of each formula φ.
- The part with the algorithms on subvalence and covering can simply be attached.

[19] at least if we consider the worst case

The PDNF and the PCNF system

The PDNF system: Declaration

The Normalizer

pdnf (φ)
is the PDNF of φ, for every formula φ

Conjunctions and Disjunctions

nlc-and-nlc (γ_1, γ_2)
is the PDNF of $[\wedge\gamma_1\gamma_2]$, for every two NLC's γ_1 and γ_2

nlc-or-nlc (γ_1, γ_2)
is the PDNF of $[\vee\gamma_1\gamma_2]$, for every two NLC's γ_1 and γ_2

nlc-and-pdnf (γ, Δ)
is the PDNF of $[\wedge\gamma\Delta]$, for every NLC γ and PDNF Δ

nlc-or-pdnf (γ, Δ)
is the PDNF of $[\vee\gamma\Delta]$, for every NLC γ and PDNF Δ

pdnf-and-pdnf (Δ_1, Δ_2)
is the PDNF of $[\wedge\Delta_1\Delta_2]$, for every two PDNF's Δ_1 and Δ_2

pdnf-or-pdnf (Δ_1, Δ_2)
is the PDNF of $[\vee\Delta_1\Delta_2]$, for every two PDNF's Δ_1 and Δ_2

pdnf-conj $((\Delta_1, \ldots, \Delta_n))$
is the PDNF of $[\wedge\Delta_1 \ldots \Delta_n]$, for every list $(\Delta_1, \ldots, \Delta_n)$ of PDNF's with $n \geq 0$

pdnf-disj $((\Delta_1, \ldots, \Delta_n))$
is the PDNF of $[\vee\Delta_1 \ldots \Delta_n]$, for every list $(\Delta_1, \ldots, \Delta_n)$ of PDNF's with $n \geq 0$

The PCNF system: Declaration

The Normalizer

pcnf (φ)
is the PCNF of φ, for every formula φ

Conjunctions and Disjunctions

nlc-or-nld (δ_1, δ_2)
is the PCNF of $[\vee\delta_1\delta_2]$, for every two NLD's δ_1 and δ_2

nlc-and-nld (δ_1, δ_2)
is the PCNF of $[\wedge\delta_1\delta_2]$, for every two NLD's δ_1 and δ_2

nld-or-pcnf (δ, Γ)
is the PCNF of $[\vee\delta\Gamma]$, for every NLD δ and PCNF Γ

nld-and-pcnf (δ, Γ)
is the PCNF of $[\wedge\delta\Gamma]$, for every NLD δ and PCNF Γ

pcnf-or-pcnf (Γ_1, Γ_2)
is the PCNF of $[\vee\Gamma_1\Gamma_2]$, for every two PCNF's Γ_1 and Γ_2

pcnf-and-pcnf (Γ_1, Γ_2)
is the PCNF of $[\wedge\Gamma_1\Gamma_2]$, for every two PCNF's Γ_1 and Γ_2

pcnf-disj $((\Gamma_1, \ldots, \Gamma_n))$
is the PCNF of $[\vee\Gamma_1 \ldots \Gamma_n]$, for every list $(\Gamma_1, \ldots, \Gamma_n)$ of PCNF's with $n \geq 0$

pcnf-conj $((\Gamma_1, \ldots, \Gamma_n))$
is the PCNF of $[\wedge\Gamma_1 \ldots \Gamma_n]$, for every list $(\Gamma_1, \ldots, \Gamma_n)$ of PCNF's with $n \geq 0$

The PDNF system: Implementation

pdnf (φ)

$:=$

$[\vee[\wedge\varphi]]$	if φ is an atom
$[\vee]$	if $\varphi = 0$
$[\vee[\wedge]]$	if $\varphi = 1$
$[\vee[\wedge\neg\alpha]]$	if $\varphi = \neg\alpha$ and α is atom
$[\vee[\wedge]]$	if $\varphi = \neg 0$
$[\vee]$	if $\varphi = \neg 1$
pdnf (φ')	if $\varphi = \neg\neg\varphi'$
pdnf-disj $((pdnf\ (\neg\varphi_1), \ldots, pdnf\ (\neg\varphi_n)))$	if $\varphi = \neg[\wedge\varphi_1 \ldots \varphi_n]$
pdnf-conj $((pdnf\ (\neg\varphi_1), \ldots, pdnf\ (\neg\varphi_n)))$	if $\varphi = \neg[\vee\varphi_1 \ldots \varphi_n]$
pdnf-conj $((pdnf\ (\varphi_1), \ldots, pdnf\ (\varphi_n)))$	if $\varphi = \neg[\overline{\wedge}\varphi_1 \ldots \varphi_n]$
pdnf-disj $((pdnf\ (\varphi_1), \ldots, pdnf\ (\varphi_n)))$	if $\varphi = \neg[\overline{\vee}\varphi_1 \ldots \varphi_n]$
pdnf-and-pdnf $(pdnf\ (\varphi_1), pdnf\ (\neg\varphi_2))$	if $\varphi = \neg[\varphi_1 \rightarrow \varphi_2]$
pdnf-or-pdnf (*pdnf-and-pdnf* $(pdnf\ (\varphi_1), pdnf\ (\neg\varphi_2))$, *pdnf-and-pdnf* $(pdnf\ (\neg\varphi_1), pdnf\ (\varphi_2))$)	if $\varphi = \neg[\varphi_1 \leftrightarrow \varphi_2]$
pdnf-conj $(pdnf\ (\varphi_1), \ldots, pdnf\ (\varphi_n))$	if $\varphi = [\wedge\varphi_1 \ldots \varphi_n]$
pdnf-disj $(pdnf\ (\varphi_1), \ldots, pdnf\ (\varphi_n))$	if $\varphi = [\vee\varphi_1 \ldots \varphi_n]$
pdnf-disj $(pdnf\ (\neg\varphi_1), \ldots, pdnf\ (\neg\varphi_n))$	if $\varphi = [\overline{\wedge}\varphi_1 \ldots \varphi_n]$
pdnf-conj $(pdnf\ (\neg\varphi_1), \ldots, pdnf\ (\neg\varphi_n))$	if $\varphi = [\overline{\vee}\varphi_1 \ldots \varphi_n]$
pdnf-or-pdnf $(pdnf\ (\neg\varphi_1), pdnf\ (\varphi_2))$	if $\varphi = [\varphi_1 \rightarrow \varphi_2]$
pdnf-or-pdnf (*pdnf-and-pdnf* $(pdnf\ (\varphi_1), pdnf\ (\varphi_2))$, *pdnf-and-pdnf* $(pdnf\ (\neg\varphi_1), pdnf\ (\neg\varphi_2))$)	if $\varphi = [\varphi_1 \leftrightarrow \varphi_2]$

nlc-and-nlc (γ_1, γ_2)

$:=$ *jrec* $(\gamma_1, \gamma_2, [\wedge])$

The PCNF system: Implementation

pcnf (φ)

$:=$

$[\wedge[\vee\varphi]]$	if φ is an atom
$[\wedge[\vee]]$	if $\varphi = 0$
$[\wedge]$	if $\varphi = 1$
$[\wedge[\vee\neg\alpha]]$	if $\varphi = \neg\alpha$ and α is atom
$[\wedge]$	if $\varphi = \neg 0$
$[\wedge[\vee]]$	if $\varphi = \neg 1$
pcnf (φ')	if $\varphi = \neg\neg\varphi'$
pcnf-disj $((pcnf\ (\neg\varphi_1), \ldots, pcnf\ (\neg\varphi_n)))$	if $\varphi = \neg[\wedge\varphi_1 \ldots \varphi_n]$
pcnf-conj $((pcnf\ (\neg\varphi_1), \ldots, pcnf\ (\neg\varphi_n)))$	if $\varphi = \neg[\vee\varphi_1 \ldots \varphi_n]$
pcnf-conj $((pcnf\ (\varphi_1), \ldots, pcnf\ (\varphi_n)))$	if $\varphi = \neg[\overline{\wedge}\varphi_1 \ldots \varphi_n]$
pcnf-disj $((pcnf\ (\varphi_1), \ldots, pcnf\ (\varphi_n)))$	if $\varphi = \neg[\overline{\vee}\varphi_1 \ldots \varphi_n]$
pcnf-and-pcnf $(pcnf\ (\varphi_1), pcnf\ (\neg\varphi_2))$	if $\varphi = \neg[\varphi_1 \rightarrow \varphi_2]$
pcnf-or-pcnf (*pcnf-and-pcnf* $(pcnf\ (\varphi_1), pcnf\ (\neg\varphi_2))$, *pcnf-and-pcnf* $(pcnf\ (\neg\varphi_1), pcnf\ (\varphi_2))$)	if $\varphi = \neg[\varphi_1 \leftrightarrow \varphi_2]$
pcnf-conj $(pcnf\ (\varphi_1), \ldots, pcnf\ (\varphi_n))$	if $\varphi = [\wedge\varphi_1 \ldots \varphi_n]$
pcnf-disj $(pcnf\ (\varphi_1), \ldots, pcnf\ (\varphi_n))$	if $\varphi = [\vee\varphi_1 \ldots \varphi_n]$
pcnf-disj $(pcnf\ (\neg\varphi_1), \ldots, pcnf\ (\neg\varphi_n))$	if $\varphi = [\overline{\wedge}\varphi_1 \ldots \varphi_n]$
pcnf-conj $(pcnf\ (\neg\varphi_1), \ldots, pcnf\ (\neg\varphi_n))$	if $\varphi = [\overline{\vee}\varphi_1 \ldots \varphi_n]$
pcnf-or-pcnf $(pcnf\ (\neg\varphi_1), pcnf\ (\varphi_2))$	if $\varphi = [\varphi_1 \rightarrow \varphi_2]$
pcnf-or-pcnf (*pcnf-and-pcnf* $(pcnf\ (\varphi_1), pcnf\ (\varphi_2))$, *pcnf-and-pcnf* $(pcnf\ (\neg\varphi_1), pcnf\ (\neg\varphi_2))$)	if $\varphi = [\varphi_1 \leftrightarrow \varphi_2]$

nld-or-nld (δ_1, δ_2)

$:=$ *jrec* $(\delta_1, \delta_2, [\vee])$

$nlc\text{-}or\text{-}nlc\ (\gamma_1, \gamma_2)$

$:= prim\ (\gamma_1, \gamma_2)$

$nlc\text{-}and\text{-}pdnf\ (\gamma, [\vee\gamma_1 \ldots \gamma_n])$

$:= pdnf\text{-}disj\ ((nlc\text{-}and\text{-}nlc\ (\gamma, \gamma_1), \ldots, nlc\text{-}and\text{-}nlc\ (\gamma, \gamma_n)))$

$nlc\text{-}or\text{-}pdnf\ (\gamma, \Delta)$

$:= prec\ ([\vee\gamma], [\vee], \Delta, [\vee])$

$pdnf\text{-}and\text{-}pdnf\ ([\vee\gamma_1 \ldots \gamma_n], \Delta)$

$:= pdnf\text{-}disj\ ((nlc\text{-}and\text{-}pdnf\ (\gamma_1, \Delta), \ldots, nlc\text{-}and\text{-}pdnf\ (\gamma_n, \Delta)))$

$pdnf\text{-}or\text{-}pdnf\ (\Delta_1, \Delta_2)$

$:= prec\ (\Delta_1, [\vee], \Delta_2, [\vee])$

$pdnf\text{-}conj\ ((\Delta_1, \ldots, \Delta_n))$

$$:= \begin{cases} [\vee[\wedge]] & \text{if } n = 0 \\ \Delta_1 & \text{if } n = 1 \\ pdnf\text{-}and\text{-}pdnf\ (\Delta_1, pdnf\text{-}conj\ ((\Delta_2, \ldots, \Delta_n))) & \text{if } n > 1 \end{cases}$$

$pdnf\text{-}disj\ ((\Delta_1, \ldots, \Delta_n))$

$$:= \begin{cases} [\vee] & \text{if } n = 0 \\ \Delta_1 & \text{if } n = 1 \\ pdnf\text{-}or\text{-}pdnf\ (\Delta_1, pdnf\text{-}disj\ ((\Delta_2, \ldots, \Delta_n))) & \text{if } n > 1 \end{cases}$$

$nld\text{-}and\text{-}nld\ (\delta_1, \delta_2)$

$:= prim\ (\delta_1, \delta_2)$

$nld\text{-}or\text{-}pcnf\ (\delta, [\wedge\delta_1 \ldots \delta_n])$

$:= pcnf\text{-}conj\ ((nld\text{-}or\text{-}nld\ (\delta, \delta_1), \ldots, nld\text{-}or\text{-}nld\ (\delta, \delta_n)))$

$nld\text{-}and\text{-}pcnf\ (\delta, \Gamma)$

$:= prec\ ([\wedge\delta], [\wedge], \Gamma, [\wedge])$

$pcnf\text{-}or\text{-}pcnf\ ([\wedge\delta_1 \ldots \delta_n], \Gamma)$

$:= pcnf\text{-}conj\ ((nld\text{-}or\text{-}pcnf\ (\delta_1, \Gamma), \ldots, nld\text{-}or\text{-}pcnf\ (\delta_n, \Gamma)))$

$pcnf\text{-}and\text{-}pcnf\ (\Gamma_1, \Gamma_2)$

$:= prec\ (\Gamma_1, [\wedge], \Gamma_2, [\wedge])$

$pcnf\text{-}disj\ ((\Gamma_1, \ldots, \Gamma_n))$

$$:= \begin{cases} [\wedge[\vee]] & \text{if } n = 0 \\ \Gamma_1 & \text{if } n = 1 \\ pcnf\text{-}and\text{-}pcnf\ (\Gamma_1, pcnf\text{-}disj\ ((\Gamma_2, \ldots, \Gamma_n))) & \text{if } n > 1 \end{cases}$$

$pcnf\text{-}conj\ ((\Gamma_1, \ldots, \Gamma_n))$

$$:= \begin{cases} [\wedge] & \text{if } n = 0 \\ \Gamma_1 & \text{if } n = 1 \\ pcnf\text{-}and\text{-}pcnf\ (\Gamma_1, pcnf\text{-}conj\ ((\Gamma_2, \ldots, \Gamma_n))) & \text{if } n > 1 \end{cases}$$

Auxiliary algorithms

$$\overline{*} := \left\{ \begin{array}{ll} \vee & \text{if } * = \wedge \\ \wedge & \text{if } * = \vee \end{array} \right\} \text{ for } * \in \{\wedge, \vee\}$$

$jrec\ ([*\lambda_1 \ldots \lambda_n], [*\lambda'_1 \ldots \lambda'_m], [*\mu_1 \ldots \mu_k])$

$$:= \begin{cases} [\overline{*}[*\mu_1 \ldots \mu_k \lambda'_1 \ldots \lambda'_m]] & \text{if } n = 0 \\ [\overline{*}[*\mu_1 \ldots \mu_k \lambda_1 \ldots \lambda_n]] & \text{if } m = 0 \\ jrec\ ([*\lambda_2 \ldots \lambda_n], [*\lambda'_1 \ldots \lambda'_m], [*\mu_1 \ldots \mu_k \lambda_1]) & \text{if } n > 0 \text{ and } m > 0 \text{ and } |\lambda_1| < |\lambda'_1| \\ jrec\ ([*\lambda_1 \ldots \lambda_n], [*\lambda'_2 \ldots \lambda'_m], [*\mu_1 \ldots \mu_k \lambda'_1]) & \text{if } n > 0 \text{ and } m > 0 \text{ and } |\lambda'_1| < |\lambda_1| \\ jrec\ ([*\lambda_2 \ldots \lambda_n], [*\lambda'_2 \ldots \lambda'_m], [*\mu_1 \ldots \mu_k \lambda_1]) & \text{if } n > 0 \text{ and } m > 0 \text{ and } \lambda_1 = \lambda'_1 \\ [\overline{*}] & \text{if } n > 0 \text{ and } m > 0 \text{ and } \overline{\lambda_1} = \lambda'_1 \end{cases}$$

$prec\ ([*\gamma_1 \ldots \gamma_c], [*\mu_1 \ldots \mu_n], [*\mu_{n+1} \ldots \mu_{n+m}], [*\pi_1 \ldots \pi_p])$

$$:= \begin{cases} [*\mu_1 \ldots \mu_n \mu_{n+1} \ldots \mu_{n+m}] & \text{if } c = 0 \text{ and } p = 0 \\ prec\ ([*\pi_1 \ldots \pi_p], [*], [*\mu_1 \ldots \mu_n \mu_{n+1} \ldots \mu_{n+m}], [*]) & \text{if } c = 0 \text{ and } p > 0 \\ prec\ ([*\gamma_2 \ldots \gamma_c \pi_1 \ldots \pi_p], [*], [*\gamma_1 \mu_1 \ldots \mu_n], [*]) & \text{if } c > 0 \text{ and } m = 0 \\ prec\ ([*\gamma_2 \ldots \gamma_c], [*], [*\mu_1 \ldots \mu_n \mu_{n+1} \ldots \mu_{n+m}], [*]) & \text{if } c > 0 \text{ and } m > 0 \text{ and } xprim\ (\gamma_1, \mu_{n+1}) = (\text{“N000”}, [*\mu_{n+1}]) \\ prec\ ([*\gamma_2 \ldots \gamma_c \mu_{n+2} \ldots \mu_{n+m} \pi_1 \ldots \pi_p], [*], [*\gamma_1 \mu_1 \ldots \mu_n], [*]) & \text{if } c > 0 \text{ and } m > 0 \text{ and } xprim\ (\gamma_1, \mu_{n+1}) = (\text{“N00P”}, [*\gamma_1]) \\ prec\ ([*\gamma_2 \ldots \gamma_c], [*], [*\mu_1 \ldots \mu_n \mu_{n+1} \ldots \mu_{n+m}], [*]) & \text{if } c > 0 \text{ and } m > 0 \text{ and } xprim\ (\gamma_1, \mu_{n+1}) = (\text{“N0P0”}, [*\mu_{n+1}]) \\ prec\ ([*\gamma_1 \ldots \gamma_c], [*\mu_1 \ldots \mu_n \mu_{n+1}], [*\mu_{n+2} \ldots \mu_{n+m}], [*\pi_1 \ldots \pi_p]) & \text{if } c > 0 \text{ and } m > 0 \text{ and } xprim\ (\gamma_1, \mu_{n+1}) = (\text{“N0PP”}, [*\gamma_1 \mu_{n+1}]) \\ prec\ ([*\gamma_2 \ldots \gamma_c \gamma'], [*], [*\mu_1 \ldots \mu_n \mu_{n+2} \ldots \mu_{n+m}], [*]) & \text{if } c > 0 \text{ and } m > 0 \text{ and } xprim\ (\gamma_1, \mu_{n+1}) = (\text{“N100”}, [*\gamma']) \\ prec\ ([*\gamma_2 \ldots \gamma_c \mu'_{n+1} \mu_{n+2} \ldots \mu_{n+m} \pi_1 \ldots \pi_p], [*], [*\gamma_1 \mu_1 \ldots \mu_n], [*]) & \text{if } c > 0 \text{ and } m > 0 \text{ and } xprim\ (\gamma_1, \mu_{n+1}) = (\text{“N10P”}, [*\gamma_1 \mu'_{n+1}]) \\ prec\ ([*\gamma_2 \ldots \gamma_c \gamma'_1], [*], [*\mu_1 \ldots \mu_n \mu_{n+1} \ldots \mu_{n+m}], [*]) & \text{if } c > 0 \text{ and } m > 0 \text{ and } xprim\ (\gamma_1, \mu_{n+1}) = (\text{“N1P0”}, [*\gamma'_1 \mu_{n+1}]) \\ prec\ ([*\gamma_1 \ldots \gamma_c], [*\mu_1 \ldots \mu_n \mu_{n+1}], [*\mu_{n+2} \ldots \mu_{n+m}], [*\pi\pi_1 \ldots \pi_p]) & \text{if } c > 0 \text{ and } m > 0 \text{ and } xprim\ (\gamma_1, \mu_{n+1}) = (\text{“N1PP”}, [*\gamma_1 \mu_{n+1} \pi]) \\ prec\ ([*\gamma_1 \ldots \gamma_c], [*\mu_1 \ldots \mu_n \mu_{n+1}], [*\mu_{n+2} \ldots \mu_{n+m}], [*\pi_1 \ldots \pi_p]) & \text{if } c > 0 \text{ and } m > 0 \text{ and } xprim\ (\gamma_1, \mu_{n+1}) = (\text{“NMNN”}, [*\gamma_1 \mu_{n+1}]) \end{cases}$$

$prim\ ([*\lambda_1 \ldots \lambda_x], [*\lambda'_1 \ldots \lambda'_y])$

$:= \Phi$, where Φ is defined by $(\xi, \Phi) :=$ *xprim* $([*\lambda_1 \ldots \lambda_x], [*\lambda'_1 \ldots \lambda'_y])$

xprim $([*\lambda_1 \ldots \lambda_x], [*\lambda'_1 \ldots \lambda'_y])$

$:=$ *xprec* $([*], [*\lambda_1 \ldots \lambda_x], [*], [*\lambda'_1 \ldots \lambda'_y], [*], [*], [*], 0, 0, 0)$

algorithm xprec $([*\lambda_1 \ldots \lambda_i], [*\lambda_{i+1} \ldots \lambda_x], [*\lambda'_1 \ldots \lambda'_j], [*\lambda'_{j+1} \ldots \lambda'_y], [*\pi_1 \ldots \pi_a], [*\pi'_1 \ldots \pi'_b], [*\pi''_1 \ldots \pi''_c], r, s, t)$

begin

if $r > 1$

then (“NMNN”, $[\overline{*}[*\lambda_1 \ldots \lambda_i \lambda_{i+1} \ldots \lambda_x][*\lambda'_1 \ldots \lambda'_j \lambda'_{j+1} \ldots \lambda'_y]]$)

else if $x = i$

then if $r = 0$

then if $s = 0$

then if ($j = y$ and $t = 0$)

then (“N000”, $[\overline{*}[*\lambda_1 \ldots \lambda_i]]$)

else (“N00P”, $[\overline{*}[*\lambda_1 \ldots \lambda_i]]$)

else if ($j = y$ and $t = 0$)

then (“N0P0”, $[\overline{*}[*\lambda'_1 \ldots \lambda'_j]]$)

else (“N0PP”, $[\overline{*}[*\lambda_1 \ldots \lambda_i][*\lambda'_1 \ldots \lambda'_j \lambda'_{j+1} \ldots \lambda'_y]]$)

else if $s = 0$

then if ($j = y$ and $t = 0$)

then (“N100”, $[\overline{*}[*\pi_1 \ldots \pi_a]]$)

else (“N10P”, $[\overline{*}[*\lambda_1 \ldots \lambda_i][*\pi'_1 \ldots \pi'_b \lambda'_{j+1} \ldots \lambda'_y]]$)

else if ($j = y$ and $t = 0$)

then (“N1P0”, $[\overline{*}[*\pi_1 \ldots \pi_a][*\lambda'_1 \ldots \lambda'_j \lambda'_{j+1} \ldots \lambda'_y]]$)

else (“N1PP”, $[\overline{*}[*\lambda_1 \ldots \lambda_i][*\lambda'_1 \ldots \lambda'_j \lambda'_{j+1} \ldots \lambda'_y][*\pi''_1 \ldots \pi''_c \lambda'_{j+1} \ldots \lambda'_y]]$)

else if $j = y$

then if $r = 0$

then if $t = 0$

then (“N0P0”, $[\overline{*}[*\lambda'_1 \ldots \lambda'_j]]$)

else (“N0PP”, $[\overline{*}[*\lambda_1 \ldots \lambda_i \lambda_{i+1} .. \lambda_x][*\lambda'_1 \ldots \lambda'_j]]$)

else if $t = 0$

then (“N1P0”, $[\overline{*}[*\pi_1 \ldots \pi_a \lambda_{i+1} \ldots \lambda_x][*\lambda'_1 \ldots \lambda'_j]]$)

else (“N1PP”, $[\overline{*}[*\lambda_1 \ldots \lambda_i \lambda_{i+1} \ldots \lambda_x][*\lambda'_1 \ldots \lambda'_j][*\pi''_1 \ldots \pi''_c \lambda_{i+1} \ldots \lambda_x]]$)

else if $|\lambda_{i+1}| < |\lambda'_{j+1}|$

then xprec $([*\lambda_1 \ldots \lambda_i \lambda_{i+1}], [*\lambda_{i+2} \ldots \lambda_x], [*\lambda'_1 \ldots \lambda'_j], [*\lambda'_{j+1} \ldots \lambda'_y], [*\pi_1 \ldots \pi_a \lambda_{i+1}], [*\pi'_1 \ldots \pi'_b], [*\pi''_1 \ldots \pi''_c \lambda_{i+1}], r, s+1, t)$

else if $|\lambda'_{j+1}| < |\lambda_{i+1}|$

then xprec $([*\lambda_1 \ldots \lambda_i], [*\lambda_{i+1} \ldots \lambda_x], [*\lambda'_1 \ldots \lambda'_j \lambda'_{j+1}], [*\lambda'_{j+2} \ldots \lambda'_y], [*\pi_1 \ldots \pi_a], [*\pi'_1 \ldots \pi'_b \lambda'_{j+1}], [*\pi''_1 \ldots \pi''_c \lambda'_{j+1}], r, s, t+1)$

else if $\lambda_{i+1} = \lambda_{j+1}$

then xprec $([*\lambda_1 \ldots \lambda_i \lambda_{i+1}], [*\lambda_{i+2} \ldots \lambda_x], [*\lambda'_1 \ldots \lambda'_j \lambda'_{j+1}], [*\lambda'_{j+2} \ldots \lambda'_y], [*\pi_1 \ldots \pi_a \lambda_{i+1}], [*\pi'_1 \ldots \pi'_b \lambda'_{j+1}], [*\pi''_1 \ldots \pi''_c \lambda'_{j+1}], r, s, t)$

else xprec $([*\lambda_1 \ldots \lambda_i], [*\lambda_{i+2} \ldots \lambda_x], [*\lambda'_1 \ldots \lambda'_j], [*\lambda'_{j+2} \ldots \lambda'_y], [*\pi_1 \ldots \pi_a], [*\pi'_1 \ldots \pi'_b], [*\pi''_1 \ldots \pi''_c], r+1, s, t)$

end.