The Bucanon Manual

www.bucephalus.org

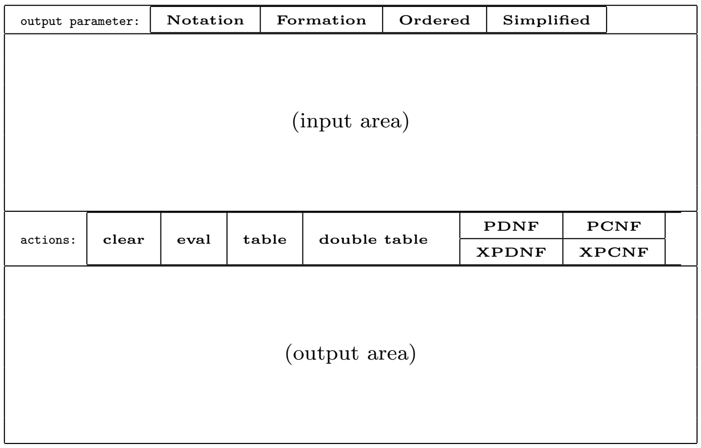
September 6, 2002

Contents

1 The Bucanon panel

1.1 The Bucanon panel

The **Bucanon panel** is the working area for the user of both the Bucanon applet and the Bucanon application. It has the following structure:



From top to bottom the Bucanon panel has four areas:
- A row of **output parameter** buttons, where the settings for the output can be altered.
- An **input area**. Here is the place to insert your input formulas.
- A row of **action** buttons. When you input a formula, press one of the action buttons, and the according results will appear in
- the **output area**.

1.2 Output parameters

The first row of the Bucanon panel contains four **output parameters** and each of these parameters is set to one of two values:
- The **notation** output parameter is either **stroke** (which is the default setting) or **arithmetic**. The syntax of formulas comes with two different notations for negation, conjunction and disjunction:

	stroke notation	arithmetic notation
negation	'x	-x
conjunction	[,] or [,x] or [x_1,...,x_n] with $n \geq 2$	[*] or [*x] or [x_1*...*x_n] with $n \geq 2$
disjunction	[;] or [;x] or [x_1;...;x_n] with $n \geq 2$	[+] or [+x] or [x_1+...+x_n] with $n \geq 2$

  The parameter determines in which notation the formulas will appear in the output area. (For input formulas, both notations can be applied, even in a mixed way, no matter how the notation parameter is set.)
- The **formation** parameter is set to either one of the following three values:
  - **line**: Output formulas are written from left to right, e.g.
```
[ '[a , 'b , c] => ['a ; b ; 'c ] ]
```
  - **tree**: Output formulas are displayed in their tree structure, e.g.
```
[ '[ a
   , 'b
   , c ]
=> ['a
   ; b
   ; 'c ] ]
```
  - **custom (max l)**: The tree structure is only applied when the formula length exceeds the defined maximal line length l. For example, with $l = 20$ we get

```
[ '[a , 'b , c]
=> ['a ; b ; 'c ] ]
```

This usually gives the most compact representation, so the *formation* parameter is set to *custom* with *max* $l := 80$ by default.

- The **ordered** output parameter determines, if the arguments of the output normal forms have to be in an ascending syntactical order or not. For a precise definition see below. The default setting is *ordered: no.*
- In a **simplified** normal form, nullary an unary conjunctions and disjunctions are replaced by bit values or the argument only. For example, `a` is the simplified form of the DNF `[+[*a]]` and the unit bit `!` is the simplified version of the CNF `[*]`. For a precise definition see below. The default setting is *simplified: yes.*

1.3 Actions

clear Clears the input and output area.

eval Generates the *evaluation* of the input formula. Evaluated formulas don't have bit values (i.e. `?` for *false* and `!` for *true*) as subformulas. For example. `[!,a,[?;'!;!],'b]` becomes `[a,'b]` after evaluation. For subvalences (i.e. `[...=>...]`) and equivalences (i.e. `[...<=>...]`) the according truth value (i.e. `?` or `!`) is generated.

table The *truth* or *bit value table* of the input formula is displayed.

double table The table is a vector of bit values on one list of atoms. A *double table* is a matrix of bit values where the one list of atoms is divided into two lists. The double table is constructed after you specified one of these lists (namely the left one).

For example, the table of `[a,c,'b]` for "a and b and not c" is the following one. Next to it there is the double table of the same formula after specifying `[a b]` as the left atom list

a	b	c	
?	?	?	?
!	?	?	?
?	!	?	?
!	!	?	?
?	?	!	?
!	?	!	!
?	!	!	?
!	!	!	?

		?	!	c
?	?	?	?	
!	?	?	!	
?	!	?	?	
!	!	?	?	
a	b			

PDNF The *Prime Disjunctive Normal Form* of the input formula is generated, i.e. the DNF where the arguments of the disjunction are exactly all the irreducible or *prime* literal conjunctions.

PCNF The *Prime Conjunctive Normal Form* of the input formula is generated, which is the dual form of the PDNF.

XPDNF The *eXtended Prime Disjunctive Normal Form* of a given input formula φ has the form `[`Δ `||` $\alpha_1 \ldots \alpha_n$`]`, where Δ is the PDNF of φ and the α_i are the atoms that occur in φ, but not in Δ.

XPCNF Generates the *eXtended Prime Conjunctive Normal Form* of the given input formula.

For a proper definition see the chapter on normal forms below.

2 Syntax

Well–defined strings of the Bucanon program are made of the following **characters**:

- An **identifier character**, which is either a letter (A,. . . ,Z,a,. . . ,z) a digit (0,1,. . . ,9) or the understroke (_).
- An **operator character**, which is either of the following

`?  !  '  ,  ;  -  *  +  <  >  =  |  @`

- A **bracket characters** [or]
- A **white space character**, which is the blank or line feed.

Well–defined **symbols** or **words** are

- **Identifiers**, which are non–empty strings of identifier characters. Six examples of identifiers are given by

`hallo_world  HalloWorld  x0  X0a13  234abc  _1`

- **Operator symbols**, which are exactly the following

`?  !  '  ,  ;  -  *  +  ->  <->`

`=>  <=>  ||  <|  |>  @|  @  -@  +@`

- **Bracket symbols** [or]

Well–defined **formulas** are made of well–defined symbols according to the rules in the right column of figure 1. So basically a formula is either a **theory formula** or a **atom list formula**. Theory formulas are a proper superset of the more common **boolean formulas**.

Note that there are two alternative notations for negation, conjunction and disjunction, called **stroke** and **arithmetic notation**:

	stroke notation	arithmetic notation
negation	'x	-x
conjunction	[,] or [,x] or [x_1,. . . ,x_n] with $n \geq 2$	[*] or [*x] or [x_1*. . . *x_n] with $n \geq 2$
disjunction	[;] or [;x] or [x_1;. . . ;x_n] with $n \geq 2$	[+] or [+x] or [x_1+. . . +x_n] with $n \geq 2$

In the sequel we use the standard LaTeX symbols as described in the middle column of figure 1. In particular, we use the standard ¬,∧,∨ for negation, conjunction and disjunction. For example the (true) formula in LaTeX notation

$$[\,[\,[\,\neg a \vee b\,] \Uparrow c\; a\,] \Leftrightarrow [\,\neg a \vee ? \vee [\,! \wedge d \wedge \neg d\,]\,]\,]$$

can be written in Bucanon notation as

`[[['a + b] <| c a] <=> [-a;?;[!,d,-d]]]`

Between any two symbols in these formulas, any amount of white space is allowed without changing the formula. (Note however, that an atoms list [$\alpha_1\ \alpha_2\ \ldots\ \alpha_n$] requires at least one white space character between each atom.)

The bracket symbols are part of the syntax and cannot be left or added arbitrarily. For example, the input of $[\,a\,]$ for a or $a \wedge b$ for $[\,a \wedge b\,]$ would lead to error massages in both cases.

The use of ? for **zero** or **false** and ! for **unit** or **true** in the Bucanon syntax is less common. As a rule to memorize these symbols, you can think of the shape of "?" as "circle and dot", which is "zero bit". Accordingly "!" is "stroke and dot", which is "unit bit".

Formulas

	in LaTeX notation	in Bucanon notation
formula		
theory formula τ		
atom α	non-empty string of letters ($A,\ldots,Z,a,\ldots,z$), digits ($0,1,\ldots,9$), and the understroke (_)	non-empty string of letters ($\texttt{A},\ldots,\texttt{Z},\texttt{a},\ldots,\texttt{z}$), digits ($\texttt{0},\texttt{1},\ldots,\texttt{9}$), and the understroke (_)
boolean junction		
bit value		
zero bit	?	`?`
unit bit	!	`!`
negation	$\neg\tau$	$\texttt{'}\tau$ $\texttt{-}\tau$
conjunction	$[\wedge]$ or $[\wedge\tau]$ or $[\tau_1 \wedge \cdots \wedge \tau_n]$ with $n \geq 2$	$\texttt{[,]}$ or $\texttt{[,}\tau\texttt{]}$ or $\texttt{[}\tau_1\texttt{,}\ldots\texttt{,}\tau_n\texttt{]}$ with $n \geq 2$ $\texttt{[*]}$ or $\texttt{[*}\tau\texttt{]}$ or $\texttt{[}\tau_1\texttt{*}\ldots\texttt{*}\tau_n\texttt{]}$ with $n \geq 2$
disjunction	$[\vee]$ or $[\vee\tau]$ or $[\tau_1 \vee \cdots \vee \tau_n]$ with $n \geq 2$	$\texttt{[;]}$ or $\texttt{[;}\tau\texttt{]}$ or $\texttt{[}\tau_1\texttt{;}\ldots\texttt{;}\tau_n\texttt{]}$ with $n \geq 2$ $\texttt{[+]}$ or $\texttt{[+}\tau\texttt{]}$ or $\texttt{[}\tau_1\texttt{+}\ldots\texttt{+}\tau_n\texttt{]}$ with $n \geq 2$
subjunction	$[\tau_1 \rightarrow \tau_2]$	$\texttt{[}\tau_1\ \texttt{->}\ \tau_2\texttt{]}$
equijunction	$[\tau_1 \leftrightarrow \tau_2]$	$\texttt{[}\tau_1\ \texttt{<->}\ \tau_2\texttt{]}$
boolean relation		
subvalence	$[\tau_1 \Rightarrow \tau_2]$	$\texttt{[}\tau_1\ \texttt{=>}\ \tau_2\texttt{]}$
equivalence	$[\tau_1 \Leftrightarrow \tau_2]$	$\texttt{[}\tau_1\ \texttt{<=>}\ \tau_2\texttt{]}$
expansion or reduction		
expansion	$[\tau \parallel \lambda]$ or $[\tau \parallel \alpha_1\ \alpha_2\ \ldots\ \alpha_n]$ with $n \geq 0$	$\texttt{[}\tau\ \texttt{\vert\vert}\ \lambda\texttt{]}$ or $\texttt{[}\tau\ \texttt{\vert\vert}\ \alpha_1\ \alpha_2\ \ldots\ \alpha_n\texttt{]}$ with $n \geq 0$
infimum reduction	$[\tau \Uparrow \lambda]$ or $[\tau \Uparrow \alpha_1\ \alpha_2\ \ldots\ \alpha_n]$ with $n \geq 0$	$\texttt{[}\tau\ \texttt{<\vert}\ \lambda\texttt{]}$ or $\texttt{[}\tau\ \texttt{<\vert}\ \alpha_1\ \alpha_2\ \ldots\ \alpha_n\texttt{]}$ with $n \geq 0$
supremum reduction	$[\tau \Downarrow \lambda]$ or $[\tau \Downarrow \alpha_1\ \alpha_2\ \ldots\ \alpha_n]$ with $n \geq 0$	$\texttt{[}\tau\ \texttt{\vert>}\ \lambda\texttt{]}$ or $\texttt{[}\tau\ \texttt{\vert>}\ \alpha_1\ \alpha_2\ \ldots\ \alpha_n\texttt{]}$ with $n \geq 0$
standard reduction	$@\vert\tau$	$\texttt{@\vert}\tau$
atom list formula λ		
atom list	$[\alpha_1\ \alpha_2\ \ldots\ \alpha_n]$ with $n \geq 0$	$\texttt{[}\alpha_1\ \alpha_2\ \ldots\ \alpha_n\texttt{]}$ with $n \geq 0$
atom list function	$@\tau$	$\texttt{@}\tau$
negative atom list function	$-@\tau$	$\texttt{-@}\tau$
positive atom list function	$+@\tau$	$\texttt{+@}\tau$

Boolean formulas

	in LaTeX notation	in Bucanon notation
boolean formula φ		
atom α	non-empty string of letters ($A,\ldots,Z,a,\ldots,z$), digits ($0,1,\ldots,9$), and the understroke (_)	non-empty string of letters ($\texttt{A},\ldots,\texttt{Z},\texttt{a},\ldots,\texttt{z}$), digits ($\texttt{0},\texttt{1},\ldots,\texttt{9}$), and the understroke (_)
boolean junction		
bit value		
zero bit	?	`?`
unit bit	!	`!`
negation	$\neg\varphi$	$\texttt{'}\varphi$ $\texttt{-}\varphi$
conjunction	$[\wedge]$ or $[\wedge\varphi]$ or $[\varphi_1 \wedge \cdots \wedge \varphi_n]$ with $n \geq 2$	$\texttt{[,]}$ or $\texttt{[,}\varphi\texttt{]}$ or $\texttt{[}\varphi_1\texttt{,}\ldots\texttt{,}\varphi_n\texttt{]}$ with $n \geq 2$ $\texttt{[*]}$ or $\texttt{[*}\varphi\texttt{]}$ or $\texttt{[}\varphi_1\texttt{*}\ldots\texttt{*}\varphi_n\texttt{]}$ with $n \geq 2$
disjunction	$[\vee]$ or $[\vee\varphi]$ or $[\varphi_1 \vee \cdots \vee \varphi_n]$ with $n \geq 2$	$\texttt{[;]}$ or $\texttt{[;}\varphi\texttt{]}$ or $\texttt{[}\varphi_1\texttt{;}\ldots\texttt{;}\varphi_n\texttt{]}$ with $n \geq 2$ $\texttt{[+]}$ or $\texttt{[+}\varphi\texttt{]}$ or $\texttt{[}\varphi_1\texttt{+}\ldots\texttt{+}\varphi_n\texttt{]}$ with $n \geq 2$
subjunction	$[\varphi_1 \rightarrow \varphi_2]$	$\texttt{[}\varphi_1\ \texttt{->}\ \varphi_2\texttt{]}$
equijunction	$[\varphi_1 \leftrightarrow \varphi_2]$	$\texttt{[}\varphi_1\ \texttt{<->}\ \varphi_2\texttt{]}$

Figure 1: The syntax of formulas

3 Semantics

We distinguish two or three ways of assigning meanings to formulas:

- **Denotational semantics**
  Each theory formula denotes a unique so–called *world*, which is again uniquely represented by its *table*. The according action in Bucanon is called by pressing the *table* button. There is also a *double table* action, that creates variations of the table. (See below in this chapter)
- **Operational semantics**
  Each input theory formula can be transformed into certain (bi)equivalent, so–called *normal forms*. The according *normalizers* or *canonizers* in Bucanon are the *PDNF*, *PCNF*, *XPDNF*, and *XPCNF* buttons.
- **Evaluation**
  Strictly speaking, this is a normalizer, too. In Bucanon the evaluation of a formula is returned after pressing the *eval* action button.

3.1 Denotational semantics

Formulas are either theory formulas or atom list formulas. The standard interpretation of formulas is the following:

- Each *atom list formula* denotes a *finite atom set*. Because there is a strict linear order $<$ defined on the set of atoms, each finite atom set has a unique representation as an *ordered atom list*. So we can say that the standard meaning of an atom list formula is an ordered atom list.
  In order to determine the ordered atom list of an atom list formula λ with the Bucanon program, input λ and call the *eval* action.
- Each *theory formula* (including each *boolean formula*) τ denotes a *world* or *theory*. In purely mathematical terms a *world* is a function of the type $(A \longrightarrow \mathcal{B}) \longrightarrow \mathcal{B}$, where A is its atom set or ordered atom list and $\mathcal{B} = \{?, !\}$ the set of bit values. Such a world is uniquely represented by its (*bit value* or *truth*) table and vice versa. So we can say that the standard meaning of a theory formula is a table.[1]
  The table of an input theory formula τ is generated in Bucanon by calling the *table* action.

One intuitive, because "meaningful" way to define the method that generates the world or table of a given theory formulas is given in two steps:

- Define the world/table of an atom formula α. This is given by

$$\begin{array}{c|c} \alpha & \\ \hline ? & ? \\ ! & ! \end{array}$$

- Define all the operator symbols $(\neg, ?, \wedge, @, \Uparrow, \Rightarrow, \dots)$ as operations on worlds/tables. This is done in the chapter below, called the *algebra of worlds*, which is also the title of the resulting structure.

The world $world(\tau)$ of an arbitrary theory formula τ is then recursively generated according to these definitions. For example:

$$world(\neg[\, a \wedge \neg a\,]) = \neg[\, world(a) \wedge \neg world(a)\,]$$

$$= \neg \left[\begin{array}{c|c} a & \\ \hline ? & ? \\ ! & ! \end{array} \wedge \neg \begin{array}{c|c} a & \\ \hline ? & ? \\ ! & ! \end{array} \right] = \neg \left[\begin{array}{c|c} a & \\ \hline ? & ? \\ ! & ! \end{array} \wedge \begin{array}{c|c} a & \\ \hline ? & ! \\ ! & ? \end{array} \right]$$

$$= \neg \begin{array}{c|c} a & \\ \hline ? & ? \\ ! & ? \end{array} = \begin{array}{c|c} a & \\ \hline ? & ! \\ ! & ! \end{array}$$

3.2 Operational semantics

Two theory formulas τ_1 and τ_2 are said to be

[1] A full account of the denotational semantics and the generation of tables and double tables is given in *World algebras*, available on *www.bucephalus.org*.

- **equiatomic** or **atomically identical** iff they have the same atom list
  (i.e. iff the evaluations of $@\tau_1$ and $@\tau_2$ return the same results)
- **equivalent** or **boolean identical** iff they are equivalent in the usual sense (see the chapter *the algebra of worlds* for a definition)
  (i.e. iff the evaluation of $eval([\,\tau_1 \Leftrightarrow \tau_2\,])$ returns !)
- **biequivalent** or **theoretically identical** iff they are equiatomic and equivalent
  (i.e. iff their tables are identical)

Each of these three relations is a proper equivalence relation (i.e. reflexive, symmetric and transitive). So each one of them induces a partition of the set of all theory formulas into a set of disjunct equivalence classes.

A subset NF of all theory formulas is a set of **atomic/boolean/theory normal forms** iff for each theory formula τ there is at least one atomically/boolean/theoretically identical form in NF. And NF is **canonic** iff this form is unique for each τ. A function that returns a (the) normal form for each τ is a (**canonic**) **atomic/boolean/theoretical normalizer**. A canonic normalizer is also called a **canonizer**.[2]

The Bucanon program essentially consists of four canonizers:

- Two boolean normalizers *pdnf* and *pcnf* that return the *prime disjunctive/conjunctive normal form* for each given theory formula.
  If the output parameter *ordered* is set to *yes*, they become boolean canonizers, because for every theory formula τ there is exactly one equivalent *ordered prime disjunctive/conjunctive normal form*.
- Two theoretical normalizers *xpdnf* and *xpcnf* that return the *extended prime disjunctive/conjunctive normal form* for each given theory formula.
  If the output parameter *ordered* is set to *yes*, they become theoretical canonizers, because for every theory formula τ there is exactly one biequivalent *ordered extended prime disjunctive/conjunctive normal form*.

While these boolean normalizers always return a boolean formula Φ for each input theory formula φ, the *extended* normal forms have the form $[\,\Phi \parallel [\,\alpha_1 \ldots \alpha_n\,]\,]$. During the transformation of φ into Φ, atoms can get lost because they are redundant for an equivalent representation. For example, in $\varphi = [\,\neg a \wedge [\,b \vee \neg b\,]\,]$, the atom b is redundant, since $\varphi \Leftrightarrow \neg a$. But while $\neg a$ is equivalent to φ, it is not equiatomic and not biequivalent. The list $[\,\alpha_1\ \ldots\ \alpha_n\,]$ in the extended normal form is exactly the list of all these redundant atoms of φ. The **expander** $\parallel$ is an operator that is defined to extend the atom set of a formula without changing the boolean semantics. It can actually be seen as an abbreviation for a boolean form, for example by defining

$$[\,\Phi \parallel \alpha_1\ \ldots\ \alpha_n\,] := [\,\Phi \parallel [\,\alpha_1\ \ldots\ \alpha_n\,]\,] := \begin{cases} [\,\Phi \wedge [\,! \vee \alpha_1 \vee \cdots \vee \alpha_n\,]\,] & \text{or alternatively} \\ [\,\Phi \vee [\,? \wedge \alpha_1 \wedge \cdots \wedge \alpha_n\,]\,] & \end{cases}$$

For example, for $\varphi = [\,\neg a \wedge [\,b \vee \neg b\,]\,]$ the form $[\,\neg a \parallel b\,]$ is a biequivalent form of φ.

Sometimes, these normal forms of the Bucanon program can look odd, especially when they are trivial. For example, the proper form $[\,\vee[\,\wedge\neg a\,]\,]$ is biequivalent to $\neg a$, and usually one would prefer to write the latter. We call this kind of biequivalent transformation a **simplification**. Basically, the nullary junctions $[\,\vee\,]$ and $[\,\wedge\,]$ are replaced by ? and !, respectively, and unary junctions like $[\,\vee\varphi\,]$ and $[\,\wedge\varphi\,]$ are just φ in their simplified form. This simplification of the output normal forms is automatically switched on and off by adjusting the *simplified* output parameter accordingly.

Precise definitions of all the normal forms used in the Bucanon program is given in the chapter *normal forms* below.

[2] Many authors use the term *canonic DNF* (and the dual *canonic CNF*) to refer to DNF's where each literal conjunction contains exactly all the atoms. For example, if $\varphi = [\,a \rightarrow b\,]$, its canonic DNF would be $\Delta = [\,[\,\neg a \wedge \neg b\,] \vee [\,\neg a \wedge b\,] \vee [\,a \wedge b\,]\,]$. These forms are meant to be canonic boolean normal forms, but in fact they are not. For every pair of formulas φ_1 and φ_2 and their canonic DNF's Δ_1 and Δ_2, the canonicity would imply that φ_1 and φ_2 are equivalent iff Δ_1 and Δ_2 are equal. But for example, $\varphi_1 := [\,a \rightarrow a\,]$ and $\varphi_2 := [\,b \rightarrow b\,]$ are equivalent and yet their canonic DNF's $\Delta_1 = [\,[\,\wedge\neg a\,] \vee [\,\wedge a\,]\,]$ and $\Delta_2 = [\,[\,\wedge\neg b\,] \vee [\,\wedge b\,]\,]$ are not equal.
Nevertheless these normal forms are important, at least from a didactic point of view, because there is an obvious bijection between worlds, tables and these formulas. Therefore we would call them *natural* rather than *canonic* normal forms.

3.3 Evaluation

The **evaluation** function, the function of the *eval* action button, is defined for all input formulas. If the input is an atom list formula, the result is the according ordered atom list. For all theory formulas, it is a boolean normalizer, i.e. it always returns an equivalent theory formula. In particular, the output formula is generated according to the following rules:

- If the input is a *boolean formula* φ, the result is a boolean formula, too, but one that is either itself a bit value, or does not contain any bit value at all. This elimination of bit values in boolean formulas is according to the common operational definition of the boolean junctors $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$. In particular

$$
\begin{array}{rcl}
\neg ? & \mapsto & ! \\
\neg ! & \mapsto & ? \\
[\, a \wedge ! \wedge b \,] & \mapsto & [\, a \wedge b \,] \\
[\, a \wedge ? \wedge b \,] & \mapsto & ? \\
[\, a \vee ! \vee b \,] & \mapsto & ! \\
[\, a \vee ? \vee b \,] & \mapsto & [\, a \vee b \,] \\
[\, a \rightarrow ! \,] & \mapsto & ! \\
[\, a \rightarrow ? \,] & \mapsto & \neg a \\
[\, ? \rightarrow a \,] & \mapsto & ! \\
[\, ! \rightarrow a \,] & \mapsto & a \\
[\, a \leftrightarrow ? \,] & \mapsto & \neg a \\
[\, a \leftrightarrow ! \,] & \mapsto & a
\end{array}
$$

  For more complex boolean formulas, this process is recursively applied. For example

$$
\begin{array}{rcccccc}
\neg[\, ! \vee a \,] & \mapsto & \neg ! & \mapsto & ? & & \\
\neg[\, ! \rightarrow [\, ? \wedge a \,]\,] & \mapsto & \neg[\, ! \rightarrow ? \,] & \mapsto & \neg ? & \mapsto & ! \\
\neg[\, a \leftrightarrow \vee[\, ! \vee a \,]\,] & \mapsto & \neg[\, a \leftrightarrow ! \,] & \mapsto & \neg a & &
\end{array}
$$

- If the input is a *boolean relation*, the output is always a bit value. The *subvalence* or *consequence relation* $\Rightarrow$ and the *equivalence* $\Leftrightarrow$ are defined as usual in propositional logic (see the chapter on worlds for a full definition), so

$$
[\, \tau_1 \Rightarrow \tau_2 \,] \quad \mapsto \quad \begin{cases} ! & \text{if } \tau_2 \text{ follows from } \tau_1 \\ ? & \text{if } \tau_2 \text{ does not follow from } \tau_1 \end{cases}
$$

$$
[\, \tau_1 \Leftrightarrow \tau_2 \,] \quad \mapsto \quad \begin{cases} ! & \text{if } \tau_1 \text{ and } \tau_2 \text{ are equivalent} \\ ? & \text{if } \tau_1 \text{ and } \tau_2 \text{ are not equivalent} \end{cases}
$$

- If the input is an expansion or reduction, i.e. if it has one of the forms

$$
[\, \tau \parallel \lambda \,] \qquad [\, \tau \Uparrow \lambda \,] \qquad [\, \tau \Downarrow \lambda \,] \qquad @|\tau
$$

  the returned result is an equivalent form. (For a proper explanation of these expressions, see *the algebra of worlds*.)
- If the input is an atom list formula, the result will be an ordered atom list. In particular

$$
\begin{array}{rcl}
[\, \alpha_1 \, \ldots \, \alpha_n \,] & \mapsto & \text{the same list, but ordered;} \\
 & & \text{for example, } [\, c\, b\, c\, a\, b\, b \,] \mapsto [\, a\, b\, c \,] \\
@\tau & \mapsto & \text{the list of atoms contained in } \tau; \\
 & & \text{for example, } @[\, c \wedge [\, b \vee \neg ba \,]\,] \mapsto [\, a\, b\, c \,] \\
-@\tau & \mapsto & \text{the list of \textit{negative} or \textit{redundant} atoms of } \tau; \\
 & & \text{for example, } -@[\, c \wedge [\, b \vee \neg ba \,]\,] \mapsto [\, a\, b \,] \\
+@\tau & \mapsto & \text{the list of \textit{positive} or \textit{irredundant} atoms of } \tau; \\
 & & \text{for example, } +@[\, c \wedge [\, b \vee \neg ba \,]\,] \mapsto [\, c \,]
\end{array}
$$

3.4 Double tables

Next to the *table* action, there is also the *double table* representation, which is a very useful tool for the intuitive introduction of some of the operations $(\parallel, \Uparrow, \Downarrow, -@, +@)$. In order to let

Bucanon construct the double table of a given theory, you need to specify its left atom list. For example, the table and double table of the theory formula $[\, a \rightarrow [\, c \wedge b\,]\,]$ for "if a, then c and b" are

a	b	c	
?	?	?	!
!	?	?	?
?	!	?	!
!	!	?	?
?	?	!	!
!	?	!	?
?	!	!	!
!	!	!	!

		?	!	c
?	?	!	!	
!	?	?	?	
?	!	!	!	
!	!	?	!	
a	b			

The double table has the left atom list $[\, a\, b\,]$.

4 Normal Forms

In the sequel, we always mean a theory (or boolean) formula, when we talk about *a formula* φ.

4.1 Literals

A **literal** is either an atom α (**positive literal**) or a negated atom $\neg\alpha$ (**negative literal**).

For every literal λ we define

$$|\lambda| := \begin{cases} \alpha & \text{if } \lambda = \alpha \text{ is a positive literal} \\ \alpha & \text{if } \lambda = \neg\alpha \text{ is a negative literal} \end{cases} \qquad \text{the atom of } \lambda$$

$$\mathit{bit}(\lambda) := \begin{cases} ! & \text{if } \lambda = \alpha \text{ is a positive literal} \\ ? & \text{if } \lambda = \neg\alpha \text{ is a negative literal} \end{cases} \qquad \text{the bit value of } \lambda$$

4.2 Syntactical order relations

A strict linear order relation $<$ is defined on each of the following sets:

- Bit values:
  For each pair of bit values β_1 and β_2 we define
  $$\beta_1 < \beta_2 \quad :\text{iff} \quad \beta_1 = ? \text{ and } \beta_2 = !$$
- Identifier characters:
  We define
  $$\mathtt{0} < \mathtt{1} < \cdots < \mathtt{9} < \mathtt{A} < \cdots < \mathtt{Z} < \mathtt{\_} < \mathtt{a} < \cdots < \mathtt{z}$$
- Atoms:
  An atom α_1 is **smaller** than an atom α_2, written $\alpha_1 < \alpha_2$, iff either α_1 is shorter than α_2 or they both have the same length and α_1 is lexically smaller than α_2.
  More precisely, for $\alpha_1 = c_1 c_2 \ldots c_n$ and $\alpha_2 = d_1 d_2 \ldots d_m$ we put
  $$\alpha_1 < \alpha_2 \quad :\text{iff} \quad \left(\begin{array}{c} n < m \text{ or} \\ (n = m \text{ and } c_1 < d_1) \text{ or} \\ (n = m > 1 \text{ and } c_1 = d_1 \text{ and } c_2 \ldots c_n < d_2 \ldots d_m) \end{array} \right)$$
- Literals:
  For each pair λ_1, λ_2 of literals, we define:
  $$\lambda_1 < \lambda_2 \quad :\text{iff} \quad \left(\begin{array}{c} |\lambda_1| < |\lambda_2| \text{ or} \\ (|\lambda_1| = |\lambda_2| \text{ and } \mathit{bit}(\lambda_1) < \mathit{bit}(\lambda_2)) \end{array} \right)$$
- Literal lists:
  The order on literal lists is the usual lexical order based on the literal order.
  More precisely, for every pair $[\,\lambda_1 \ldots \lambda_n\,]$ and $[\,\lambda'_1 \ldots \lambda'_m\,]$ of literal lists, we define
  $$[\,\lambda_1 \ldots \lambda_n\,] < [\,\lambda'_1 \ldots \lambda'_m\,] \quad :\text{iff} \quad \left(\begin{array}{c} (n = 0 \text{ and } m > 0) \text{ or} \\ (n > 0 \text{ and } m > 0 \text{ and } \lambda_1 < \lambda'_1) \text{ or} \\ \left(\begin{array}{c} n > 0 \text{ and } m > 0 \text{ and } \lambda_1 = \lambda'_1 \text{ and} \\ [\,\lambda_2 \ldots \lambda_n\,] < [\,\lambda'_2 \ldots \lambda'_m\,] \end{array} \right) \end{array} \right)$$

We say that

- an atom list $[\,\alpha_1\,\alpha_2 \ldots \alpha_n\,]$ is **ordered** iff $\alpha_1 < \alpha_2 < \cdots < \alpha_n$.
- a literal list $[\,\lambda_1\,\lambda_2 \ldots \lambda_n\,]$ is **ordered** iff $\lambda_1 < \lambda_2 < \cdots < \lambda_n$.
- a literal list $[\,\lambda_1\,\lambda_2 \ldots \lambda_n\,]$ is **normal** iff $|\lambda_1| < |\lambda_2| < \cdots < |\lambda_n|$.

4.3 DNF's and CNF's

Definition

- A **normal literal conjunction** or **NLC** is a formula of the form
$$[\,\lambda_1 \wedge \cdots \wedge \lambda_n\,]$$
where $n \geq 0$ and $[\,\lambda_1\ \dots\ \lambda_n\,]$ is a normal literal lists, i.e. $|\lambda_1| < \cdots < |\lambda_n|$.
- A **normal literal disjunction** or **NLD** is a formula of the form
$$[\,\lambda_1 \vee \cdots \vee \lambda_n\,]$$
where $n \geq 0$ and $[\,\lambda_1\ \dots\ \lambda_n\,]$ is a normal literal lists, i.e. $|\lambda_1| < \cdots < |\lambda_n|$.
- A **disjunctive normal form** or **DNF** is a disjunction of NLC's, i.e. it has the form
$$[\,[\,\lambda_{1,1} \wedge \cdots \wedge \lambda_{1,n_1}\,] \vee \cdots \vee [\,\lambda_{m,1} \wedge \cdots \wedge \lambda_{m,n_m}\,]\,]$$
where $m \geq 0$ and for each $i = 1, \dots, m$ holds: $n_i \geq 0$ and $[\,\lambda_{i,1}\ \dots\ \lambda_{i,n_i}\,]$ is a normal literal list (i.e. $|\lambda_{i,1}| < \cdots < |\lambda_{i,n_i}|$).
- A **conjunctive normal form** or **CNF** is a conjunction of NLD's, i.e. it has the form
$$[\,[\,\lambda_{1,1} \vee \cdots \vee \lambda_{1,n_1}\,] \wedge \cdots \wedge [\,\lambda_{m,1} \vee \cdots \vee \lambda_{m,n_m}\,]\,]$$
where $m \geq 0$ and for each $i = 1, \dots, m$ holds: $n_i \geq 0$ and $[\,\lambda_{i,1}\ \dots\ \lambda_{i,n_i}\,]$ is a normal literal list (i.e. $|\lambda_{i,1}| < \cdots < |\lambda_{i,n_i}|$).

Theorem (DNF's and CNF's are boolean normal forms)
- Every formula φ has an equivalent DNF Δ.
This Δ is unique only if φ is a contradiction. In that case, $\Delta = [\vee]$. Otherwise, there are infinitely many equivalent DNF's.
- Every formula φ has an equivalent CNF Γ.
This Γ is unique only if φ is a tautology. In that case, $\Gamma = [\wedge]$. Otherwise, there are infinitely many equivalent CNF's.

4.4 Ordered DNF's and CNF's

Definition
- A DNF
$$[\,[\,\lambda_{1,1} \wedge \cdots \wedge \lambda_{1,n_1}\,] \vee \cdots \vee [\,\lambda_{m,1} \wedge \cdots \wedge \lambda_{m,n_m}\,]\,]$$
is said to be **ordered** iff
$$[\,\lambda_{1,1}\ \dots\ \lambda_{1,n_1}\,] < \cdots < [\,\lambda_{m,1}\ \dots\ \lambda_{m,n_m}\,]$$
- A CNF
$$[\,[\,\lambda_{1,1} \vee \cdots \vee \lambda_{1,n_1}\,] \wedge \cdots \wedge [\,\lambda_{m,1} \vee \cdots \vee \lambda_{m,n_m}\,]\,]$$
is said to be **ordered** iff
$$[\,\lambda_{1,1}\ \dots\ \lambda_{1,n_1}\,] < \cdots < [\,\lambda_{m,1}\ \dots\ \lambda_{m,n_m}\,]$$

Theorem
- For every DNF $\Delta = [\,\gamma_1 \vee \cdots \vee \gamma_n\,]$ there is a unique ordered DNF $\Delta' = [\,\gamma'_1 \vee \dots \gamma'_m\,]$ with the same components (i.e. $\{\gamma'_1, \dots, \gamma'_m\} = \{\gamma_1, \dots, \gamma_n\}$), called **the ordered** form of Δ. Obviously, Δ and Δ' are (bi)equivalent.
- For every CNF $\Gamma = [\,\delta_1 \wedge \cdots \wedge \delta_n\,]$ there is a unique ordered CNF $\Gamma' = [\,\delta'_1 \wedge \dots \delta'_m\,]$ with the same components (i.e. $\{\delta'_1, \dots, \delta'_m\} = \{\delta_1, \dots, \delta_n\}$), called **the ordered** form of Γ. Obviously, Γ and Γ' are (bi)equivalent.

With the Bucanon program each output DNF and CNF is transformed into its ordered form, when the output parameter *ordered* is set to *yes*.

4.5 Simplified DNF's and CNF's

Definition
- A formula is a **simplified DNF** if it has one of the following forms:
  - ? the zero bit
  - ! the unit bit
  - λ a literal
  - $[\,\lambda_1 \wedge \cdots \wedge \lambda_n\,]$ a NLC with $n \geq 2$

- $[\,\gamma_1 \vee \cdots \vee \gamma_m\,]$ with $m \geq 2$, where each γ_i is either a literal λ or a NLC $[\,\lambda_1 \wedge \cdots \wedge \lambda_n\,]$ with $n \geq 2$.
- A formula is a **simplified CNF** if it has one of the following forms:
  - ? the zero bit
  - ! the unit bit
  - λ a literal
  - $[\,\lambda_1 \vee \cdots \vee \lambda_n\,]$ a NLD with $n \geq 2$
  - $[\,\delta_1 \vee \cdots \vee \delta_m\,]$ with $m \geq 2$, where each δ_i is either a literal λ or a NLD $[\,\lambda_1 \vee \cdots \vee \lambda_n\,]$ with $n \geq 2$.

Definition

- If $[\,\lambda_1 \wedge \cdots \wedge \lambda_n\,]$ is a NLC, then

$$simp([\,\lambda_1 \wedge \cdots \wedge \lambda_n]\,]) := \begin{cases} ! & \text{if } n = 0 \\ \lambda_1 & \text{if } n = 1 \\ [\,\lambda_1 \wedge \cdots \wedge \lambda_n\,] & \text{if } n > 1 \end{cases}$$

- If $[\,\lambda_1 \vee \cdots \vee \lambda_n\,]$ is a NLD, then

$$simp([\,\lambda_1 \vee \cdots \vee \lambda_n]\,]) := \begin{cases} ? & \text{if } n = 0 \\ \lambda_1 & \text{if } n = 1 \\ [\,\lambda_1 \vee \cdots \vee \lambda_n\,] & \text{if } n > 1 \end{cases}$$

- If $[\,\gamma_1 \vee \cdots \vee \gamma_n\,]$ is a DNF, then

$$simp([\,\gamma_1 \vee \cdots \vee \gamma_n]\,]) := \begin{cases} ? & \text{if } n = 0 \\ simp(\gamma_1) & \text{if } n = 1 \\ ! & \text{if } n > 1 \text{ and } ! \in \{simp(\gamma_1), \ldots, simp(\gamma_n)\} \\ [\,simp(\gamma_1) \vee \cdots \vee simp(\gamma_n)\,] & \text{if } n > 1 \text{ and } ! \notin \{simp(\gamma_1), \ldots, simp(\gamma_n)\} \end{cases}$$

- If $[\,\delta_1 \wedge \cdots \wedge \delta_n\,]$ is a CNF, then

$$simp([\,\delta_1 \wedge \cdots \wedge \delta_n]\,]) := \begin{cases} ! & \text{if } n = 0 \\ simp(\delta_1) & \text{if } n = 1 \\ ? & \text{if } n > 1 \text{ and } ? \in \{simp(\delta_1), \ldots, simp(\delta_n)\} \\ [\,simp(\delta_1) \wedge \cdots \wedge simp(\delta_n)\,] & \text{if } n > 1 \text{ and } ? \notin \{simp(\delta_1), \ldots, simp(\delta_n)\} \end{cases}$$

Theorem and definition

- For every DNF Δ, $simp(\Delta)$ is a (bi)equivalent simplified DNF of Δ, called **the simplified** Δ.
- For every CNF Γ, $simp(\Gamma)$ is a (bi)equivalent simplified CNF of Γ, called **the simplified** Γ.

With the Bucanon program each output DNF and CNF is transformed into its simplified form, when the output parameter *simplified* is set to *yes*.

4.6 PDNF's and PCNF's

Definition

- A NLC γ is a **disjunctive factor** or **implicand** or simply a **factor** of a DNF Δ iff $\gamma \Rightarrow \Delta$.
- A NLD δ is a **conjunctive factor** or **consequence** or **cofactor** of a CNF Γ iff $\Gamma \Rightarrow \delta$.
- A factor $\gamma = [\,\lambda_1 \wedge \cdots \wedge \lambda_n\,]$ of a DNF Δ is a **prime** factor, if it is irreducible in the sense that $[\,\lambda_1 \wedge \cdots \wedge \lambda_{i-1} \wedge \lambda_{i+1} \wedge \cdots \wedge \lambda_n\,] \not\Rightarrow \Delta$, for each $i = 1, \ldots, n$.
- A cofactor $\delta = [\,\lambda_1 \vee \cdots \vee \lambda_n\,]$ of a CNF Γ is a **prime** cofactor, if it is irreducible in the sense that $\Gamma \not\Rightarrow [\,\lambda_1 \vee \cdots \vee \lambda_{i-1} \vee \lambda_{i+1} \vee \cdots \vee \lambda_n\,]$, for each $i = 1, \ldots, n$.
- A DNF $\Delta = [\,\gamma_1 \vee \cdots \vee \gamma_n\,]$ is a **prime disjunctive normal form** or **PDNF** iff $\{\gamma_1, \ldots, \gamma_n\}$ is the set of its n prime factors.
- A CNF $\Gamma = [\,\delta_1 \wedge \cdots \wedge \delta_n\,]$ is a **prime conjunctive normal form** or **PCNF** iff $\{\delta_1, \ldots, \delta_n\}$ is the set of its n prime cofactors.

4.7 Boolean canonizations

Theorem (Ordered PDNF's and PCNF's are canonic boolean normal forms)

- Every formula φ has one equivalent PDNF Δ. This PDNF is unique up to the order of its components. In other words, every formula φ has exactly one equivalent ordered PDNF Δ, called **the (canonic or ordered) PDNF** of φ.
  Furthermore, each such Δ can be transformed into its (bi)equivalent simplified form $simp(\Delta)$.
- Every formula φ has one equivalent PCNF Γ. This PCNF is unique up to the order of its components. In ohter words, every formula φ has exactly one equivalent ordered PDNF Γ, called **the (canonic or ordered) PCNF** of φ.
  Furthermore, each such Γ can be transformed into its (bi)equivalent simplified form $simp(\Gamma)$.

With the Bucanon program these PDNF's and PCNF's are generated after the input of φ by pressing the according action button. The resulting normal form is ordered and/or simplified according to the settings of the *ordered* and *simplified* output parameter settings.

4.8 XPDNF's and XPCNF's

Definition

- An **extended prime disjunctive normal form** or **XPDNF** is a formula

$$[\,\Delta \parallel [\,\alpha_1 \,\ldots\, \alpha_n\,]\,]$$

  where Δ is a PDNF and $[\,\alpha_1 \,\ldots\, \alpha_n\,]$ is an ordered list of atoms (i.e. $\alpha_1 < \cdots < \alpha_n$) with $n \geq 0$, such that none of these atoms occurs in Δ.
- An **extended prime conjunctive normal form** or **XCNF** is a formula

$$[\,\Gamma \parallel [\,\alpha_1 \,\ldots\, \alpha_n\,]\,]$$

  where Γ is a PCNF and $[\,\alpha_1 \,\ldots\, \alpha_n\,]$ is an ordered list of atoms (i.e. $\alpha_1 < \cdots < \alpha_n$) with $n \geq 0$, such that none of these atoms occurs in Γ.

4.9 Ordered XPDNF's and XPCNF's

Definition

- A XPDNF $[\,\Delta \parallel [\,\alpha_1 \,\ldots\, \alpha_n\,]\,]$ is said to be **ordered** iff Δ is ordered.
- A XCNF $[\,\Gamma \parallel [\,\alpha_1 \,\ldots\, \alpha_n\,]\,]$ is said to be **ordered** iff Γ is ordered.

4.10 Simplified XPDNF's and XPCNF's

Definition

- A formula is a **simplified XPDNF** if it has one of the following forms
  - Δ a simplified PDNF
  - $[\,\Delta \parallel [\,\alpha_1 \,\ldots\, \alpha_n\,]\,]$, where Δ is a simplified PDNF and $[\,\alpha_1 \,\ldots\, \alpha_n\,]$ is an ordered list of atoms, none of them occuring in Δ, with $n \geq 2$
- A formula is a **simplified XCDNF** if it has one of the following forms
  - Γ a simplified PCNF
  - $[\,\Gamma \parallel [\,\alpha_1 \,\ldots\, \alpha_n\,]\,]$, where Γ is a simplified PCNF and $[\,\alpha_1 \,\ldots\, \alpha_n\,]$ is an ordered list of atoms, none of them occuring in Γ, with $n \geq 2$

Definition

- If $[\,\Delta \parallel [\,\alpha_1 \,\ldots\, \alpha_n\,]\,]$ is a XDNF, then

$$simp([\,\Delta \parallel [\,\alpha_1 \,\ldots\, \alpha_n\,]\,]) := \begin{cases} simp(\Delta) & \text{if } n = 0 \\ [\,simp(\Delta) \parallel [\,\alpha_1 \,\ldots\, \alpha_n\,]\,] & \text{if } n > 0 \end{cases}$$

- If $[\,\Gamma \parallel [\,\alpha_1 \,\ldots\, \alpha_n\,]\,]$ is a XCNF, then

$$simp([\,\Gamma \parallel [\,\alpha_1 \,\ldots\, \alpha_n\,]\,]) := \begin{cases} simp(\Gamma) & \text{if } n = 0 \\ [\,simp(\Gamma) \parallel [\,\alpha_1 \,\ldots\, \alpha_n\,]\,] & \text{if } n > 0 \end{cases}$$

4.11 Theory canonizations

Theorem (Ordered XPDNF's and XPCNF's are canonic theory normal forms)

- Every formula φ has one biequivalent XPDNF $[\,\Delta \parallel [\,\alpha_1\ \dots\ \alpha_n\,]\,]$, which is unique in case it is ordered. This ordered form is called **the (canonic or ordered) XPDNF** of φ.
  Furthermore, each such $[\,\Delta \parallel [\,\alpha_1 \dots \alpha_n\,]\,]$ can be transformed into its biequivalent simplified form $simp([\,\Delta \parallel [\,\alpha_1\ \dots\ \alpha_n\,]\,])$.
- Every formula φ has one biequivalent ordered XPCNF $[\,\Gamma \parallel [\,\alpha_1\ \dots\ \alpha_n\,]\,]$, which is unique in case it is ordered. This ordered form is called **the (canonic or ordered) XPCNF** of φ.
  Furthermore, each such $[\,\Gamma \parallel [\,\alpha_1 \dots \alpha_n\,]\,]$ can be transformed into its biequivalent simplified form $simp([\,\Gamma \parallel [\,\alpha_1\ \dots\ \alpha_n\,]\,])$.

With the Bucanon program these XPDNF's and XPCNF's are generated after the input of φ by pressing the according action button. The resulting normal form is ordered and/or simplified according to the settings of the *ordered* and *simplified* output parameter settings.