bmlut

keys:	**[to]**	*<number>*	destination picture to contain mapping tables
	if	*<expression>*	logical expression defining which neighbourhood configurations give a positive (non-zero) result
	unless	*<expression>*	logical expression defining which neighbourhood configurations give a negative (zero) result
	neighbours/with	*<number>*	use only with **erode** or **dilate** option:
	neighbours		minimum number of clear/set neighbours in order to erode/dilate centre pixel
	with		picture containing user supplied mapping table for use with **erode** or **dilate** command
options:	**erode/dilate/median**		generate mapping table for 3 x 3 neighbouhood transformations supported by **erode, dilate** or **median** commands
	skeletonise/ends/nodes/outline/ol4		use only with **erode** option:
	skeletonise		erode to 8-connected skeleton
	ends		erode ends of protruding branches/hairs
	nodes		erode skeleton node points, separating branches
	outline		generate 8-connected outline
	ol4		generate 4-connected outline
	separately		use only with **dilate** option: dilate, preserving separation between forground regions (reduces background to 4-connected skeleton)

You use **bmlut** to generate neighbourhood mapping tables. These can define any kind of 3 by 3 binary neighbourhood transformation. A mapping table is applied with the separate command **bmmap**. Once a mapping table has been created it can be used any number of times. With the option **erode, dilate** or **median**, you can generate mapping tables that correspond exactly to any of the morphological transformations supported by the commands **erode, dilate** and **median**. More general transformations can be defined by specifying logical expressions with the keys **if** and **unless** to define the outcome for the 512 possible configurations in a 3 by 3 neighbouring. These logical expressions should include at least one of the variables *p0* to *p8, n4, n8, c, c4* and *c8* to characterise the neighbourhood configurations as required for a particular morphological transformation.

bmlut

Examples

```
bmlut 99 erode outline; bmmap 1 2 with 99
```

This example outputs outlines of foreground regions in picture 1 to picture 2 (same result as **erode 1 2 outline**).

```
bmlut 99 if (p4+n8) > 4
```

This command outputs the mapping table for a binary median filter (same result as **bmlut 99 median**).

```
bmlut 99 if (p4 & n8~=0) | (~p4 & n8=8); bmmap 1 with 99
```

This example removes salt-and-pepper noise from picture 1.

```
bmlut 99 if p4 & n8=1
```

This command outputs a mapping table to detect line ends.

Description

Bmlut lets you set up any kind of transformation of a 3 by 3 binary neighbourhood. What you have to do is specify the outcome (a 0 or a 1) for each of the 512 possible neighbourhood configurations. **Bmlut** stores this neighbourhood mapping table in a Semper picture for later use with the **bmmap** command. Each configuration is represented by a 9-bit index value in the range 0 to 511. Each bit in the index value represents the state of one of the pixel locations in the 3 by 3 neighbourhood as follows.

```
0 3 6
1 4 7
2 5 8
```

where bit 0 is the least significant bit and bit 4 corresponds to the central pixel. The index value points to the entry in the mapping table where the result for that neighbourhood configuration is stored.

You may decide to create mapping tables directly by creating a new picture of the appropriate size and setting each mapping table entry with the **pixel** command. However, if the outcome for each neighbourhood configuration can be specified in the form of a common logical expression, **bmlut** can be directed to generate the mapping table for you.

bmlut

You specify the necessary logical conditions with the keys **if** and **unless**. Either key accepts a general Semper logical expression which **bmlut** evaluates for each of the 512 neighbourhood configurations. Each logical expression should normally refer to at least one of the variables *p0* to *p8, n4, n8, c, c4* and *c8* which **bmlut** sets itself before it evaluates the expression. The result obtained determines the value stored in the mapping table. If the **if** expression is true and the **unless** expression is false, a 1 is output, otherwise, a zero is output.

Two keys are provided for convenience in specifying some logical conditions but exactly the same result could be achieved with just one key, that is

```
bmlut if <if-expression> unless <unless-expression>
bmlut if <if-expression> & ~<unless-expression>
bmlut unless ~<if-expression> | <unless-expression>
```

are all equivalent. You may also omit either one of the two expressions

```
bmlut if <if-expression>
bmlut unless <unless-expression>
```

Bmlut sets the variables *po* to *p8* according to the value of the pixels in a particular neighbourhood configuration. For example a neighbourhood index value of 463

```
                    1 1 1
463 = 111001111  =>  1 0 1
                    1 0 1
```

causes variables *p4* and *p5* to be set to zero (false) and the variables *p0, p1, p2, p3, p6, p7* and *p8* to be set to 1 (true).

To create a mapping table which detects just this neighbourhood configuration (a **hit** or **miss** transform) you could specify the following logical expression with the **if** key

```
p0 & p1 & p2 & p3 & p4 & ~p4 & ~p5 & p6 & p7 & p8
```

This would cause a mapping table to be output with all entries set to zero except entry 463.

Of course, there are more useful neighbourhood transformations that can be defined besides these simple Hit or Miss transforms (which, anyway, are more efficiently carried out with the command **bhmt**). In particular, you should note that you are not restricted to pure logical expressions when using **bmlut**. For example, to create the mapping table to detect all neighbourhood configurations where the central pixel is surrounded by exactly 4 non-zero neighbours, you could specify the following logical expression

```
( p0 + p1 + p2 + p3 + p5 + p6 + p7 + p8 ) = 4
```

bmlut

Logical expressions which count pixels in a 3 by 3 neighbourhood are very common in defining the necessary conditions for many morphological transforms. Consequently, **bmlut** provides further variables *n4, n8, c, c4* and *c8* for added convenience. They are defined in the following way

```
n4 = p1 + p3 + p5 + p7

n8 = p0 + p1 + p2 + p3 + p5 + p6 + p7 + p8

c = (~p0 & p1) + (~p1 & p2) + (~p2 & p5) + (~p5 & p8) + (~p8 & p7)
                            + (~p7 & p6) + (~p6 & p3) + (~p3 & p0)

c4 = p1 + p3 + p5 + p7  - (p0 & p1 & p3) - (p2 & p1 & p5)
                        - (p6 & p3 & p7) - (p8 & p5 & p7)

c8 = ~p1 + ~p3 + ~p5 + ~p7 - (~p0 & ~p1 & ~p3) - (~p2 & ~p1 & ~p5)
                           - (~p6 & ~p3 & ~p7) - (~p8 & ~p5 & ~p7)
```

n4 and *n8* represent the number of 4-connected and 8-connected neighbours around the central pixel.

c represents the neighbourhood "crossing number" which is the number of 0–1 transitions in the central pixel's surround. This takes no account of connectivity between neighbouring pixels in the central pixel's surround, so the variables *c4* and *c8* are provided for this. They represent the number of separate 4-connected and 8-connected foreground regions which surround the central pixel. These two parameters are somethimes referred to as "connectivity numbers" in the morphological literature.

Further parameters mentioned in the literature include the bond number and the coefficients of curvature for 4-connected and 8-connected surrounds. These are not provided in variable form because they are less commonly used and are easily derived from the variables described above

bond number	`n4 + n8`
4-connected coefficient of curvature	`( 4 - n4 ) - c4`
8-connected coefficient of curvature	`( 8 - n8 ) - (c8 + c + c8)`

The 14 variables supported by **bmlut** in logical expressions specified with the keys **if** and **unless** provide a simple and elegant way to generate the mapping tables for many useful morphological transformations, as the following list of examples illustrate

bmlut

4-neighbour erode	`bmlut if p4 & n4=4`
8-neighbour erode	`bmlut if p4 & n8=8`
4-neighbour dilate	`bmlut if p4 | n4~=0`
8-neighbour dilate	`bmlut if p4 | n8~=0`
Hole detect	`bmlut if ~p4 & n4=4`
Interior pixel fill	`bmlut if p4 | n4=4`
End detect	`bmlut if p4 & n8=1`
End remove	`bmlut if p4 unless n8=1`
Isolated pixel detect	`bmlut if p4 & n8=0`
Isolated pixel remove	`bmlut if p4 unless n8=0`
Remove salt–and–pepper noise	`bmlut if (p4 & n8~=0) | (~p4 & n8=8)`
Spur detect	`bmlut if p4 & (n4 + n8) = 1`
Spur remove	`bmlut if p4 unless (n4 + n8) = 1`
4-connected outline	`bmlut if p4 unless n8=8`
8-connected outline	`bmlut if p4 unless n4=4`
H detect	`bmlut if p4 & n8=6 & c4=2 & c8=2`
H break	`bmlut if p4 unless n8=6 & c4=2 & c8=2`
Triple point detect	`bmlut if p4 & c>2`
Median filter	`bmlut if (p4 + n8) >4`

All of these examples have quite simple logical conditions. **Bmlut** supports general and quite lengthy logical expresssions as in the following example which generates the mapping table for obtaining the octagonal convex hull

```
bmlut if p4 | (p1 & p3 & p5)                              +
            | (p3 & p5 & p7)                              +
            | (p5 & p7 & p1)                              +
            | (p7 & p1 & p3)                              +
            | (p1 & p3 & (p2 | p6) & ~p5 & ~p7 & ~p8)     +
            | (p5 & p7 & (p2 | p6) & ~p1 & ~p3 & ~p0)     +
            | (p1 & p5 & (p0 | p8) & ~p3 & ~p7 & ~p6)     +
            | (p3 & p7 & (p0 | p8) & ~p1 & ~p5 & ~p2)
```

where "+" is Semper's line continuation character.

The first four lines of the logical expression can in fact be replaced with just "p4 | n4 >= 3".

Some morphological operations require a series of morphological transformations to be applied in turn (opening, for example, consists of erosion followed by dilation). **Bmlut** allows you to specify more than one logical expression with the **If** or **unless** keys. Each expression is used to generate a separate mapping table which is stored as a separate row in the output picture. You use commas "," to separate the expressions.

bmlut

For example, a more rounded result for dilation can be achieved by making alternate use of 4-connected and 8-connected dilation. The mapping table for this can be generated in the following way

```
bmlut if p4 | n4~=0, p4 | n8~=0
```

If the **If** and **unless** keys are used together and they specify a different number of expressions, **bmlut** will repeat the shorter set of expressions to make up the same number as the longer set. For example, quite a good thinning transformation can be obtained with four mapping tables as follows

```
bmlut if p4, p4, p4, p4 unless  c8=1 & n8~=1 & (~p1 & p7), +
                                c8=1 & n8~=1 & (~p7 & p1), +
                                c8=1 & n8~=1 & (~p3 & p5), +
                                c8=1 & n8~=1 & (~p5 & p3)
```

The string of expressions given by the **If** key can be replaced with just **p4** to give the same result.

```
bmlut if p4 unless  c8=1 & n8~=1 & (~p1 & p7),+
                    c8=1 & n8~=1 & (~p7 & p1),+
                    c8=1 & n8~=1 & (~p3 & p5),+
                    c8=1 & n8~=1 & (~p5 & 3)
```

The logical condition **unless c8=1 & n8~=1** is the basic condition for eroding pixels in order to thin objects down to an 8-connected skeleton (**c8=1** selects edge pixels that can be eroded without causing a break in the central pixel's surround and **n8~=1** protects line ends). Because neighbourhood transformations are applied in parallel to all the neighbourhoods in the source image, without reference to any changes that might occur in neighbouring pixel locations, this is not enough to maintain the connectivity of the result. This accounts for the extra condition on the end of each expression which restricts the erosion during each pass to one of the four cardinal directions and ensures that two-pixel-wide features are not completely eroded.

Substituting *variable c4* and *n4* for *c8* and *n8* will give a similar, but 4-connected result.

```
bmlut if p4 unless  c4=1 & n4~=1 & (~p1 & p7),+
                    c4=1 & n4~=1 & (~p7 & p1),+
                    c4=1 & n4~=1 & (~p3 & p5),+
                    c4=1 & n4~=1 & (~p5 & p3)
```

These thinning transformations can be thought of as erosions which preserve the connectivity of foreground regions and which do not erode line ends. One can easily modify these examples to model thickening transformations to produce background skeletons.

bmlut

Inverting the basic conditions for the 8-connected skeleton, you obtain the following command

```
bmlut if p4 | (c8=1 & n8~=7 & (~p1 & p7)), +
          p4 | (c8=1 & n8~=7 & (~p7 & p1)), +
          p4 | (c8=1 & n8~=7 & (~p3 & p5)), +
          p4 | (c8=1 & n8~=7 & (~p5 & p3))
```

This transformation preserves 8-connectivity of foreground regions – so the background skeleton will be 4-connected. For an 8-connected result, substitute *c4* and *n4* for *c8* and *n8*, as before, but notice that the inequality involving n4 must also be modified, that is

```
bmlut if p4 | (c4=1 & n4~=3 & (~p1 & p7)), +
          p4 | (c4=1 & n4~=3 & (~p7 & p1)), +
          p4 | (c4=1 & n4~=3 & (~p3 & p5)), +
          p4 | (c4=1 & n4~=3 & (~p5 & p3))
```

If you invert the result obtained by applying the mapping tables generated by this last example, you will obtain the 8-connected background skeleton, or exo-skeleton.

If you omit from the last example the logical condition **n4~=3** for protecting line ends , you will obtain, after inverting the processed result, the 8-connected skiz (skeleton by zones of influence) of the source image.

```
bmlut if p4 | (c4=1 & (~p1 & p7)), +
          p4 | (c4=1 & (~p7 & p1)), +
          p4 | (c4=1 & (~p3 & p5)), +
          p4 | (c4=1 & (~p5 & p3))
```

Bmlut also has keys and options (as an alternative to using the keys **if** or **uniess** which allow you to generate mapping tables for any of the neighbourhood transformations supported by the commands **erode, dilate** and **median (bmmap** is usually much faster to use than **erode, dilate** or **median**). You select which command by setting the corresponding option and you append exactly the same string of keys or options as you would use with the command itself. For example,

```
bmlut dilate separately
```

would generate the mapping tables that **bmmap** would require to carry out exactly the same morphological transformation as the command

```
dilate separately
```

bmlut

Note that the user-defined mapping tables supplied to **erode** and **dilate**, by specifying the key **with**, are 8-bit mapping tables which refer only to the central pixel's surround and have a different pattern of bits from that used by **bmlut**

```
7 6 5
0 * 4
1 2 3
```

Note also that when using the commands **erode skeletonise**, **erode ends** and **dilate separately**, the **times** key automatically defaults to 0 (infinite number of iterations). With **bmmap**, the default for the **times** key is always 1, so you must explicitly set it to zero in order to get the same result.

For more detailed information about these morpholgical transformations, consult the documentation for the commands **erode, dilate** and **median.**

Notes

see also: **bmmap, bhmt, erode, dilate, median,**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
if	true	valid expression
unless	false	valid expression
neighbours	none	1 to 8
with	none	valid picture number