MyCreativity and MyReports Manual

for Southbeach Modeller 3.0

www.southbeachinc.com

Beyond static diagrams. MyCreativity will transform your drawing and modeling experience. By adding simple rules and phrases, models become 'active'. As you click on, navigate or edit a model, the output generated guides problem solving, innovation and many other creative tasks.

Anyone can use MyCreativity - from school child to PhD scientist. Write rules that capture the way you work and have Southbeach Modeller emulate hundreds of creative and problem solving tools.

useful+potential “What will you do with {this}?”

Contents

Introduction

This guide explains the ‘creativity’ rules engine and reporting language within Southbeach Modeller. It is intended for those who want to exploit the full power of the software. It assumes you know how to use the Southbeach Modeller user interface.

The guide is divided into two major sections:

1. MyCreativity
2. MyReports

MyReports builds on MyCreativity. Learn one, and you have (virtually) learnt the other.

MyReports adds a small number of syntax extensions that apply to reporting, e.g. numbering of output, as well as a few extra macros that only make sense in the context of report generation.

Developing MyCreativity rules and MyReports templates may initially look like 'programming'. That is not the case. Don’t be put off. The syntax is easy to use once you have understood the basics.

If you need help writing your first creativity rules, write to:

support@southbeachinc.com

If you are a licensed user, we promise to help you.

Over time, we will release further extensions to the creativity engine. Please register your software to be informed of these as and when they become available. To register, use the menu:

Help - Register My Software

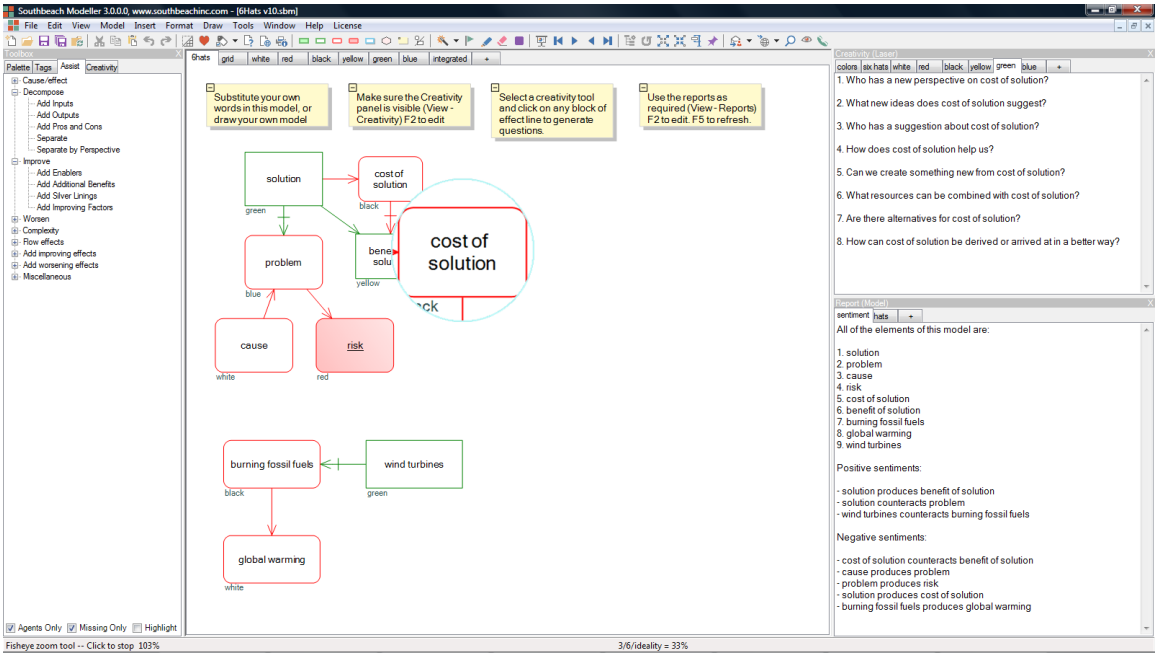
MyCreativity

The simple idea behind MyCreativity is this: as you click around a model, on either agents or effects, if the pattern in the model (or within an 'extent') matches a rule in your system, it will output a sentence.

The sentences can be anything you like. As they are output they can pull data from the model and insert that into the sentence. Here is a typical rule:

harmful "Find a way to reduce or eliminate {this}"

The 'harmful' part is called a pattern. It says: match any harmful agent. The text in quotes is the output. The {this} refers to the agent being processed and will be replaced at run time by the name of the agent. Thus, the sentence generated will depend on which block you click on.



'Extent' is how far Modeller searches for matching patterns around the place where you click. They are:

- 'Laser' only fires rules that match the ‘clicked on’ object
- 'Narrow' extends 'laser' by one connected object or effect
- 'Extended' extends 'narrow' by another
- Ditto 'Wide' extends 'extended'
- Ditto 'Wider' extends 'wide'
- 'Widest' matches all agents and effects it can find, anywhere in the model, by following connected links from the ‘clicked on’ object.

MyCreativity Reference Guide

This section specifies the full range of patterns and macros provided by MyCreativity.

All rule-sets (in the library) are written as follows:

```
#name
pattern1 "output containing macros in {} brackets"
pattern2 "output containing macros in {} brackets"
etc.
```

You can also write multi-line rules that include embedded returns.

```
pattern "output text line 1\n
output line 2\n
output line 3"
```

Here is a simple rule:

```
goal "Can we achieve {this}?"
```

{this} is called a macro. It refers to the name of the model element you clicked on. If the model element is a goal, the rule will fire and output the question. The macro {this} will be replaced by the name of the model element clicked on. For example, if you clicked on 'profit' the following text would be output:

```
Can we achieve profit?
```

Here is a rule for an effect in the model:

```
produces(, harmful) "Can we find a way to prevent {source} from
producing harmful {destination}?"
```

The rule would match any effect in the model that is a 'production' where the destination is harmful. It would not care about the source, for that is not specified.

The two forms:

```
produces(*, harmful) and
produces(, harmful)
```

are equivalent.

{source} and {destination} refer to the two agents linked by the effect. If those agents were 'carbon dioxide' and 'global warning', the output would be:

```
Can we find a way to prevent carbon dioxide from producing harmful
global warming?
```

Tip: {from} and {to} can be substituted for {source} and {destination}

Patterns

You can write patterns to match either 1) single agents, 2) pairs of agents connected by an effect, or 3) a group of three connected agents. The latter is called a 'triple'. Here are the three forms:

*	matches any agent
*(,)	matches any effect – between agents or other effects
*(,) *(,)	is used to match a 'triple'

A 'triple' is a pair of effects, sharing an anchor – either an agent or another effect. Triples are used to specify 'contradictions' (as in TRIZ) or other kinds of three agent patterns. (See later section)

* can be replaced by any combination of Southbeach Notation keywords, e.g.:

useful+potential+focus

matches a 'useful' agent that is also 'potential' and is marked as a 'focus' point in the model.

Attributes can also be used in effect patterns or triple patterns to match the attributes of the 'source', 'destination' or 'anchor' point agents or effects. For example:

harmful+questionable(*, harmful)

matches any effect that is questionably harmful and whose output – whatever it is - is also harmful.

The full set of keywords for specifying patterns are listed in the tables below. You will recognize these as they are identical to the attributes you use to develop a Southbeach model. If you already know Southbeach Notation, you also know how to write MyCreativity rules!

Pattern keywords that match agents or effects

In the tables below, examples of agent or effect patterns are used to illustrate the use of the keyword in a typical pattern. The examples are kept deliberately simple only for clarity. Any keyword can also be used in triples (2 effects with an anchor agent) and in composite patterns (e.g. useful+insufficient+focus) as you wish.

Pattern keyword	*Definition*	*Examples*
*	Matches anything	* "I clicked on {this}" *(,) "{from} is effecting {to}"
useful	Matches useful agents or effects (green)	useful "{this} is useful" useful(harmful,) "How is harmful {from} improving the system?"
harmful	Matches harmful agents or effects (red)	harmful "What is the root cause of {this}?" harmful(useful,) "Find an alternative to {from} that does not produce {to}"
neutral	Matches neutral agents or effects (gray)	neutral "Is {this} useful or harmful?"
emphasis	Matches emphasized agents or effects (thicker line)	harmful+emphasis "Why did the model author emphasis {this}?"
highlight	Matches highlighted agents or effects (Highlighter tool in Whiteboard menu)	highlight "Should {this} be highlighted?"
insufficient	Matches insufficient agents or effects (dashed line)	insufficient+useful "Find a way to increase {this}" produces+insufficient(useful, useful) "How can {from} be modified so as to produce more of {to}?"
sufficient	Matches agents or effects that are 'sufficient' (for example, not potential, not insufficient, not dysfunctional)	!sufficient "What is wrong with {this}?"
potential	Matches potential agents or effects (dotted line)	potential+action "Should we invoke action: {this}?" potential+counteracts(, harmful) "How can we avoid {from} harming {to}?"

dysfunctional	Matches dysfunctional agents or effects (irregular line)	causes(, dysfunctional) "Why does {from} lead to dysfunction in {to}?"
@<name>	Matches any text in the model, such as object names, types, effect labels, grid axes labels and user supplied tags Using @ on a grid is powerful, since it matches any agent in that grid cell – whatever its attributes Since users can tag models how they like, @ opens up the creativity and reporting engine to many creative methods (see later section)	@car, @degrades(,), @process, @future @car "{this} is a car" @degrades(, useful+@process) "What about {from} is degrading process: {to}?" decreases(@weakness, @strength) "{this} is a problem for us"
!<keyword or name>	Not the thing (or 'other than') Do not confuse NOT with ! NOT+produces and !produces are <u>not</u> the same (see later section)	e.g. !useful means NOT useful, i.e. harmful or neutral e.g. !produces(,) matches any effect other than 'produces' !potential+action "{this} is something we have decided to do"

Pattern keywords specific to agents

Pattern keyword	*Definition*	*Examples*
agent	Matches normal agents, i.e. not choices, issues, actions or knowledge	agent "{this} is not a choice, issue, action or knowledge"
issue	Matches only issues (lozenge)	produces(, issue+useful) "Does {this} raise any other useful questions?"
choice	Matches only choices (diamond)	choice+harmful "Can we find a way to avoid making the choice: {this}?"
action	Matches only actions (blue)	action "What resources do we need in order to take action: {this}?"
knowledge	Matches only knowledge (hexagons)	knowledge "How can we be sure that {this} is true?"
goal	Matches goals (green filled)	goal "Can any of these: {useful} help reach the goal: {this}?"
risk	Matches risks (red filled)	risk "How can we mitigate {this}?" *(, risk) "How can we change {from} so that the risk {to} does not arise?"
focus	Matches agents where focus has been set (yellow highlight)	harmful+focus "We are particularly concerned about {this}"
historical	Matches agents marked historical (X)	historical+choice "Why do we no longer have to take a choice about {this}?"
surplus	Matches surplus agents (doubled line)	surplus+useful "How can we use surplus {this} as a resource elsewhere in the system?"

start	Matches any agent at the start of a chain, i.e. having outputs but no inputs Think of this as begins, origin, first, root …	start "Does {this} have any outputs other than {outputs}?" start+harmful "Why is {this} a root cause?"
end	Matches any agent at the end of a chain, i.e. having inputs but no outputs Think of this as finish, terminator, last, result …	goal+end "{this} is a final goal" produces(, goal+!end) "Can {from} reach {goal+end} without the need for intervening goal {to}?"
noconnections	Matches agents with no connections, i.e. having no effects, isolated in the model	noconnections "Should {this} have any connections to the rest of the model?" * "Does {this} effect any of these: {noconnections}?"
space@<name>	Matches any agent separated in space, e.g. space@outside	space@below "Should {this} move above?" counteracts(, space@above) "How does {from} counteract {to} in the space above?"
time@<name>	Matches any agent separated in time, e.g. time@future	Ditto
parts@<name>	Matches any agent separated by parts (structure), e.g. parts@thispart	Ditto
perspective@<name>	Matches any agent separated by perspective, e.g. perspective@mine	Ditto
aspect@<name>	Matches any agent separated by aspect, e.g. aspect@quality	Ditto
role@<name>	Matches any agent separated by role, e.g. role@CEO	Ditto
probability@<name>	Matches any agent separated by probability, e.g. probability@certain	Ditto
condition@<name>	Matches any agent separated on a condition, e.g. condition@limit	Ditto
version@<name>	Matches any agent separated on version, e.g. version@next	Ditto

Pattern keywords specific to effects

Pattern keyword	*Definition*	*Examples*
produces	Matches produces effects	produces(,) "…."
counteracts	Matches counteracts effects	counteracts(,) "…."
creates	Matches creates effects	creates(,) "…."
destroys	Matches destroys effects	destroys(,) "…."
stores	Matches stores effects	stores(,) "…."
consumes	Matches consumes effects	consumes(,) "…."
opposed	Matches opposed effects	opposed(,) "…."
is_a	Matches 'Is A' effects	is_a(,) "…."
prevents	Matches prevents effects	prevents(,) "…."
related	Matches related effects	related(,) "…."
becomes	Matches becomes effects	becomes(,) "…."
replaces	Matches replaces effects	replaces(,) "…."
user_defined	Matches user defined effects	user_defined(,) "…."
contributesto	Matches contributes to effects	contributesto(,) "…."
detractsfrom	Matches detracts from effects	detractsfrom(,) "…."
uses	Matches uses effects	uses(,) "…."
implements	Matches implements effects	implements(,) "…."
specifies	Matches specifies effects	specifies(,) "…."
NOT	Matches any effect marked as NOT occurring, e.g. NOT being produced Note: do not confuse this with '!' (see below)	NOT+harmful(,) "…."
necessary	Matches any necessary effect	necessary (,) "…."
inevitable	Matches any inevitable effect	inevitable(,) "…."
delay	Matches any delayed effect	delay(,) "…."
questionable	Matches any questionable effect	questionable(,) "…."
excessive	Matches excessive effects (doubled line)	excessive(,) "…."
increases	Matches any 'increasing' effect, includes: produces, creates, contributes to, causes ...	increases(,) "…"

decreases	Matches any 'decreasing' effect, includes: counteracts, destroys, detracts from, consumes, prevents ...	decreases(,) "…"

Effect patterns such as 'increases' and 'decreases', 'useful' and 'harmful', let you write general rules that match effects that shares the property. They also let you write rules that reflect changes in effect impact arising from changes at other points in the model. For example, suppose the model said:

useful A produces useful B

Then the rule:

harmful (,) "Why does {from} cause harm to {to}?"

would fire if, for example, the user changed B to harmful. This would cause the 'produces' effect to be harmful. The sentence would then be output.

The same thing would happen if the user, instead of changing the agent, changed the 'produces' effect to 'counteracts'. Once again, the effect would then be 'harmful', and the rule would fire.

How to use @

@ is used to match any text in the model. This includes user supplied effect labels, grid axes (separations), tags and even agent names. Let's look at each in turn.

Matching effects via their effect labels

For example, suppose you were labeling your effects consistently, you would be able to write rules that matched your own terminology. For example:

destroys@corrodes(, useful)

would match any 'destroys' effect that you also labeled 'corrodes' (in this case as long as it were degrading something useful).

By not specifying the effect type, you can match just the labels on any effect, for example:

@corrodes(, useful)

would match any effect labeled 'corrodes'.

Matching agents placed on a grid

@ can also be used to match the names of axes of grids, for example on a SWOT chart:

@weaknesses

would match any agent in the model that is placed in the 'weaknesses' quadrant of the SWOT chart.

The capability to refer to grid axes is powerful. For example, it would allow you to write a rule such as:

counteracts(@threats, @opportunities) "How can we avoid {to} from counteracting {from}?"

This rule would match any agent in the 'threats' box of the 2x2 SWOT that is counteracting any agent in the 'opportunities' box. Move the agents across the model, and the output would change. Presto! This works because agents 'pick up' attributes off the grid. An agent 'competitor' in the threats box may be marked 'harmful', but it is also, by implication, a 'threat'. Thus, it has the attribute 'threats' and the attribute 'harmful'.

It is also possible to write less precise rules. For example:

> harmful(@threats, @opportunities) "Does the threat {from} cause a problem for our opportunity: {to}?"

This rule would match any agent in the 'threat' quadrant that has a harmful effect on an agent in the 'opportunity' quadrant. Creating output from consulting models such as this is a powerful and unique capability of Southbeach Modeller.

@ can also be used with separations on grids, for example:

> space@outside "Does {this} need a roof?"
>
> role@CEO+harmful "Is {this} a problem for our CEO?"

Matching user supplied tags

@ can also be used to match 'tags' in the model. Southbeach Modeller supports tagging of any agent, using multiple exclusive or inclusive tag sets and tag values.

Suppose the agents in a model had been tagged as 'processes' or as 'inputs' or as 'outputs', it could be possible to write the following rules:

> produces(@process, @output) "How does the process {from} generate the output {to}?"

Matching the names of agents

While less often used, it is also possible to use @ to precisely match specific agent names, for example:

> @car "This is a car"
> produces(@car, @travel) "This effect shows a car producing travel"

Because @ can be used to match different properties of the agents, it will match a mixture. For example, @car will match an agent tagged 'car' and an agent called 'car'.

Using ! (not, meaning 'other than')

When you are writing a combination of attributes, for example:

useful+potential+goal

the + sign signifies 'and'. All of the attributes must be present for the rule to match. Sometimes, however, it is useful to be able to exclude a specific attribute. This is done as follows:

useful+!goal

This pattern would distinguish a useful agent that is not a goal. Here are more examples:

harmful+!insufficient

matches harmful agents or effects that are not insufficient

!produces

matches any effect other than produces

counteracts(, !goal)

matches a counteracting effect on something other than a goal

The difference between NOT and !

Do not confuse NOT and !

NOT is an attribute of an effect in Southbeach. For example, A does NOT produce B. If you wish to match NOT use:

not+produces(,)

For example:

not+produces(, useful) "Why does {from} not produce useful {to}?"

By contrast: '!' really means 'other than'. For example:

harmful+!risk

matches anything harmful in the model that is not a risk. For example:

harmful+!risk "Could {this} become a risk?"

Macros: the stuff in {} or []

Macros are the commands in brackets that you include in your output text in order to extract information from the model itself. Often this is just the name of the clicked on agent, for example:

* "I have clicked on {this}"

As you click around the model, the sentence is output with {this} substituted by the name of the agent or effect. Many other keywords, and even some patterns, can be used.

The following table sets out the possibilities:

Macro keyword	Definition	Examples
{this}	The description of the agent, or the effect, clicked on where the rule matches and fires	* "I clicked on {this}" *(,) "I clicked on {this}"
{selected}	If scope is 'laser', {this} and {selected} are the same agent or effect Otherwise, {selected} is the clicked on object, and {this} is wherever the pattern falls, within 'extent' (See later section)	harmful "Can {selected} be helped by {this}?"
{from}	The agent (or the effect) at the input end of an effect, e.g. in A produces B, {from} = A	produces(, harmful) "Why does {from} cause harmful {to}?"
{to}	The agent at the output end of an effect, e.g. in A produces B, {to} = B	Ditto
{effects}, {effecting} etc.	The name of the effect clicked on, e.g. 'produces'. If the effect is the 'User Defined' effect, and has a user defined effect label, it outputs the label itself. For the complete table of alternates and how to use them, refer to Appendix C	*(,) "{from} {effects} {to}?" harmful(,) "Why should {from} {effecting} {to} be harmful?"
{source} or {arg1}	Alternate name for {from}	
{destination} or {arg2}	Alternate name for {to}	
{&<argument name>}	A user defined argument name in an effect or triple pattern	produces(&a=*, &b=harmful) "How does {&a} produce harmful {&b}?"

{inputs}	A list of the objects that are an 'input' to {this}, e.g., if B produces A and C counteracts A, the {inputs} are B and C	* "My inputs are: {inputs}"
{outputs}	A list of the objects that are an 'output' of {this}, e.g., if A produces B and A counteracts C the {outputs} are B and C.	* "My harmful outputs are: {outputs+harmful}"
{adjacent}	A list of the objects directly connected to {this}	* "I am connected to: {adjacent}"
{1hop}	Same as {adjacent}	
{2hops} … {5hops}	A list of the names of all agents 2 … 5 hops away from {this} Note - hops are counted by agents and effects, thus the agent B in 'A produces B' is two hops from A. The effect 'produces' is 1 hop from A.	* "Things near me are: {3hops}"
random	A qualifying keyword which selects a random agent from the list returned by another macro Allows for the specification of creativity that includes a degree of randomness (a.k.a. 'creativity') The position of the keyword in the pattern makes no difference to the result, e.g. useful+random is the same as random+useful (See later section)	e.g. {harmful+random} {inputs+random} {5hops+random}
{<any agent pattern>}	A list of the names of all agents in the model that match the pattern Note - If you combine macro keywords with a pattern, the output will vary depending on context, e.g. {produces+goal} means a list of the goals directly produced by {this}	{goal+potential} will output all potential goals in the model as a comma separated list {@vehicle+dysfunctional} will output all agents in the model that are tagged 'vehicle' and which are also dysfunctional

When to use {selected} instead of {this}

Always use {this} where you mean the agent where the rule matched, for example:

> harmful+potential "Under what conditions could {this} be harmful?"

Use {selected} only in cases where you want to distinguish between the agent you 'clicked on' with the mouse and the agent against which the rule is matching. There will only be a difference between {this} and {selected} when you have set an extent greater than 'laser'.

You can therefore write rules such as:

> * "Have we understood how {selected} is effected by {this}?"

Clicking somewhere in the model, with a wide extent, will generate several different sentences.

Using arguments '&='

In a pattern, you may optionally name an argument to an effect, e.g.:

> produces(&a=useful, &b=harmful)

You can then refer to these arguments as macros, and insert them into the output text, thus:

> produces(&a=useful, &b=harmful) "Find a way to limit {&a} so that less of {&b} is generated"

Naming arguments like this is normally unnecessary in a simple pattern like the one above, because there are already built in macros for source {from} and destination {to}, thus:

> produces(useful, harmful) "Find a way to limit {from} so that less of {to} is generated"
>
> produces(useful, harmful) "Find a way to limit {source} so that less of {destination} is generated"

However, arguments are essential to triple patterns (contradictions and other 3-element model patterns), because they play the role of defining the 'anchor' element in the model. See next section.

Triples (and contradictions)

A Southbeach 'triple' is the term Southbeach Solutions uses to refer to a pair of effects connected by an anchor element. For example:

> A (the anchor) is producing useful B, but is also producing harmful C
>
> Useful A (the anchor) is producing useful B, but is also being counteracted by C
>
> A is transitioning to state C, through a harmful intermediate state B (the anchor)
>
> A (the anchor) is producing surplus B, and insufficient C

These, and many others, are patterns found in all typical models. Using MyCreativity, you can create rules to match them, and so generate creative suggestions. TRIZ practitioners know this as 'generating directions', e.g. via a creative algorithm such as ARIZ.

Arguments (&a, &b etc.) are used to refer to the elements of the pattern, for example:

> produces(&a=useful, &b=useful) produces(&a, &c=harmful) "Find a way for {&a} to produce {&b} without producing {&c}"

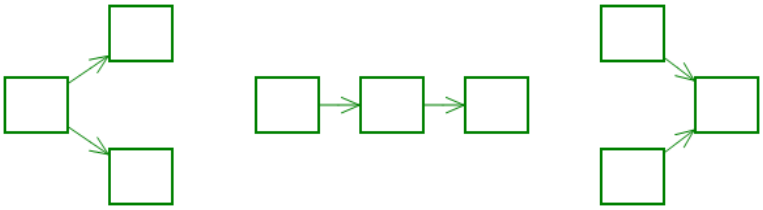
or perhaps:

> produces(&a=useful, &b=useful) produces(&a, &c=harmful) "Do we need {&a} in the system? Is there another way to produce {&b} so as to avoid producing {&c}?"

Note how, in a triple, there are <u>three</u> possible positions for the anchor element:

> effect1(&a, &b) effect2(&a, &c) (outward effects from a)
>
> effect1(&a, &b) effect2(&b, &c) (a chain)
>
> effect1(&b, &a) effect2(&c, &a) (inwards to a)

To be clear, these patterns are shown visually on the next page:



Triples make for a powerful way to add creativity to models. By using rules that match different effect types, their attributes, and even tags of the surrounding agents, it is possible to automate many creative processes.

For example, here is a more complex rule:

> becomes+delay(&a=useful, &b=risk+process) becomes+delay(&b, &c=useful)
>
> "Find a way to speed up process {&b} (which represents a risk) so that we can replace {&a} with {&c}"

Using Tags and creativity (@)

In Modeller, agents can be 'tagged' (similar to the way web pages as tagged on social networks). The software comes with a library of tags for all kinds of consulting and engineering purposes. It also lets you define your own tag groups and tag values.

Tag groups can be exclusive, or inclusive, for example:

Animals (exclusive)
Dog, Cat, Elephant, Giraffe

Traits (inclusive)
Fluffy, Fierce, Friendly, African, Domestic, Tall, Large, Small

MyCreativity and MyReports provide three ways of matching tagged agents.

@tagvalue	(lazy match tag not depending on tag group)
@taggroup:tagvalue	(strict match of tag within the tag group)
@taggroup	(match based on tag group name)

You can use these as a pattern (to specify whether a rule should fire) or as a macro (to extract the names of the corresponding agents into the output generated by the rule).

Here are examples of creativity based on tags:

* "The dogs in the model other than {this} are {@dog}"

* "The cats in the model are {@cat}"

* "The african animals in the model are {@african}"

* "The fierce animals other than {this} are: {@traits:fierce}"

@dog "Is {this} fierce or friendly?"

@dog+@fierce "{this} is a fierce dog that should not be domestic"

increases(, @dog+@fierce)

"Why does {from} increase the aggresion in {to}"?

Using ‘random’ in MyCreativity

‘Random’ is a qualifying keyword that can be added to other patterns. For example:

harmful+random

returns a randomly selected harmful agent.

‘random’ is typically used in macros in order to select and insert a random agent into a line of creativity or into the body of a report. For example, here is a rule which matches on any harmful agent and asks a question:

harmful “Can {useful+random} be used to solve {this}?”

Each time you click on a harmful agent, a different sentence is generated. Here is another similar example:

* “Can {useful+random} be used to solve {harmful+random}?”

In this case, clicking on any element of the model outputs a sentence which includes two randomly selected agents.

Note: ‘random’ can only be used in macros, not in patterns. Think about it, it makes sense!

Like many macro keywords, the ‘random’ keyword can also be used in MyReports. It works slightly differently here. See section below.

Macros to generate correct form of English Language output

Suppose you wished to write a rule that applies to any effect. Take this example:

*(,) “How does {from} {effects} {to}?”

If the model says A produces B, the output would be:

How does A produces B?

Yuk! That is bad English. It should correctly read:

How does A produce B?

To allow you to write rules that generate correct English output, MyCreativity provides variations of the {effects} macro. The variations provided are:

{effect}	How does A produce B?
{effects}	A produces B
{effecting}	A is producing B
{effected}	A produced B in the past
{effectedby}	A was produced by B
{effector}	A is the producer of B
{effection}	The production of B by A

These macros generate the correct English form for *all* of the Southbeach effects. Many of them also deal with negated (NOT) effects (i.e. A is NOT *produced by* B). For a few, providing the NOT form in this automated fashion makes no sense, since the NOT in the output sentence may need to be in a different place in order to make sense in English. Therefore, to write rules for NOT effects, consider using:

<effect>+NOT(,) ".... sentence to include *not* in the correct position ..."

Using the MyCreativity library

As well as embedding sequences of rules in 'creativity tabs' and storing them with your model, you can build up a library of rule-sets as part of your Modeller environment.

The library is displayed in the Toolbox panel as a hierarchy. Check boxes allow you to turn on/off any rule-set. When set on, output is generated and merged with any output from 'model specific' rules.

The library is stored in .txt files in the MyCreativity sub-folder of the Southbeach documents folder, and also appears in the Explorer panel.

You can place any number of .txt files in the MyCreativity folder. They are loaded and merged at start up. An example file _examples.txt is provided as an illustration. (Future versions of Modeller will expand on this, eventually providing many rule-sets for common consulting and engineering tasks)

The syntax for defining a rule-set is:

```
// any comment

#name of rule-set

rule
rule
rule
etc.
```

You can have any number of rule-sets in a file. The names you give rule-sets determine their hierarchical view in the toolbox panel, e.g.

```
#hats
#hats.black
#improve
#improve.increase useful
#improve.decrease harmful
```

We recommend that you define more basic (commonly used) rules at lower levels of the tree, and more unusual rules (less used) at higher levels in the tree. In this way, you can turn on extra rules and generate more output, by drilling down and checking the appropriate boxes at run time.

How you organize your rules is entirely up to you. Whether you work with just a few rules, or hundreds of rule-sets for every possible creative method, is your choice.

The same principle works for the tags library. Some users suppress many of the tag groups they don't need, and/or add new tag groups for 'domain knowledge' relevant to their work.

MyReports Reference Guide

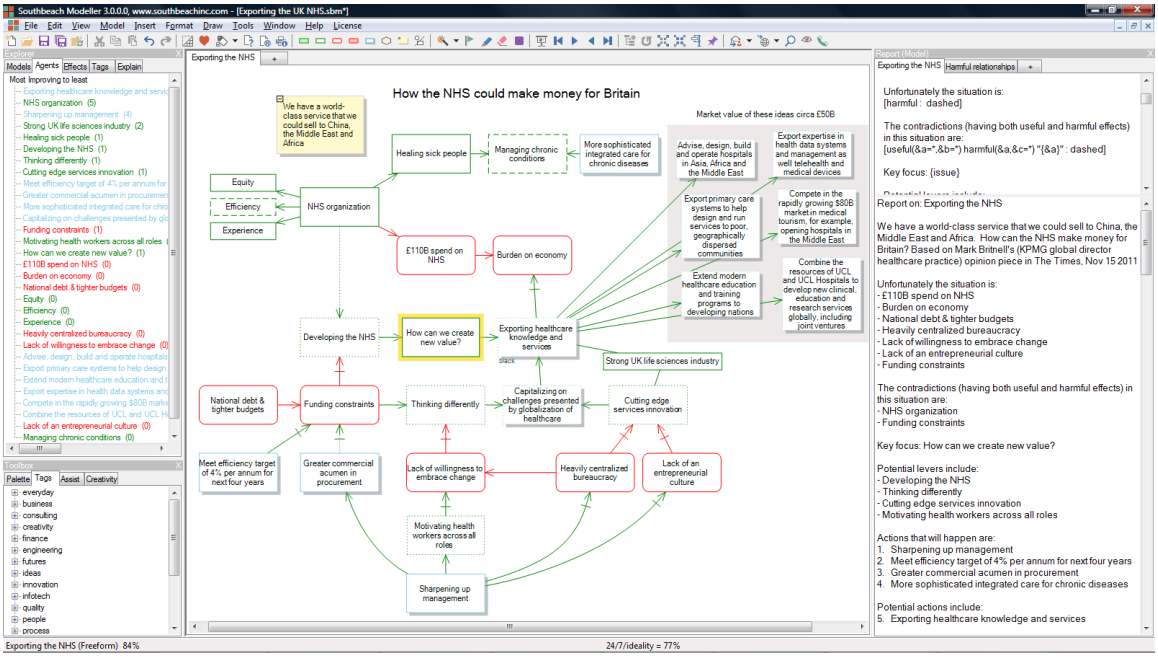
Just as you can add 'creativity' tabs to any model, and then add rules to them, you can also add 'report' tabs, and add report templates to them. When you refresh the report, the output is updated according to the current content of the model.

What is the difference between MyCreativity and MyReports?

MyCreativity is interactive, as you click around the model, the output generated will vary wildly and depends only on which rules and patterns match the point in the model (and its extent) when you click and select objects.

MyReports is different; the structure of the report remains stable (just as you need it to be), and when refreshed, information is inserted into the body of the report. Thus, you can write an entire consulting report and have sections of the report lift information from your model(s).

To get the idea, take a look at this screenshot:



The very simplest report would be something like this:

> My model contains the following agents:
> [*]

The * means 'match all' just as it does in MyCreativity. The angle brackets [] signify a macro to be expanded: they say "replace this at this point in the report". In this case, a list of the names of all agents in the model would be inserted into the report.

Inside the [ ] you can put any MyCreativity pattern, with or without template text. To illustrate, here is a short report template that you would find useful with any model:

> // My report (comment does not appear in output)
> A report on my model [model.name]
>
> The model contains the following harmful functions:
> [harmful]
>
> The risks are:
> [risk]
>
> To find a solution we are focussed on: {focus}
>
> Let's ask the following questions:
>
> [increases(, harmful) "Find a way to prevent {from} increasing {to}"
> [decreases(, useful) "Find a way to prevent {from} decreasing {from}"

Whereas in MyCreativity the rules fire around the 'clicked on' object, in MyReports *all* of the elements of the model are processed every time you refresh the report output. You can always keep your reports in sync with the model, even after significant changes to the model. It is even possible to run the report against multiple models in a file, by setting the report output panel to 'All models'.

Tip - To refresh a report, simply press the F5 key in the report panel, or select from the right mouse menu of the report panel.

In the report example above, did you spot that {focus} was in curly brackets in the report template above, and not in angled [] brackets? The latter generates one line per item output, and the former a comma separated list. Using a different style of bracket is an easy way to specify whether you want lines or a list. You will probably use both at different points in the report.

As you can see, MyReports builds on the MyCreativity syntax you have already learnt. In the following sections, there are some extensions that you can use to format the report output.

Southbeach Modeller makes it very easy to work with reports. Every model file contains several models. Every file can also contain several reports. Simply by selecting a model, and a report, via a single-click on their tabs, invokes the report with the model. Each has a separate output area, so all reports can be applied to all models, and all output is kept separate.

Numbering and lettering the MyReport output

Sometimes it is nice to bullet point, number or letter multiple lines of output in a report. The following outputs a numbered list of all useful functions:

[useful : numeric]

The following:

[increases(, useful) "How does {from} help to increase {to}?" : questions]

generates sentences each starting Q1, Q2, Q3 etc.

The colon : symbol is used to signify the start of arguments to the macro within the angle brackets.

The following bullet, number and letter styles are supported:

: dashed : ddashed	A simple dashed list - A double-dashed list --
: bullets : dbullets	A simple bullet point list Double bullet list
: chevrons : dchevrons	A > style list A >> style list
: numbered	A simple numbered list 1. 2. 3. …
: alpha	A simple alphabetic list A. B. C. …
: assertions	A1, A2, A3 …
: actions	A1, A2, A3 …
: barriers	B1, B2, B3 …
: choices	C1, C2, C3 …
: constraints	C1, C2, C3 …
: decisions	D1, D2, D3 …
: goals	G1, G2, G3 …
: harmfuls	H1, H2, H3 …
: ideas	I1, I2, I3 …
: issues	I1, I2, I3 …
: opportunities	O1, O2, O3 …
: options	O1, O2, O3 …

: proposals	P1, P2, P3 …
: problems	P1, P2, P3 …
: questions	Q1, Q2, Q3 …
: strengths	S1, S2, S3 …
: solutions	S1, S2, S3 …
: threats	T1, T2, T3 …
: tasks:	T1, T3, T3 …
: usefuls	U1, U2, U3 …
: weaknesses	W1, W2, W3 …
: a to z : A to Z	lower case list, a1, a2 to z1, z2 … upper case list, A1, A2 to Z1, Z2 …
: pros	+1, +2, +3 …
: cons	-1, -2, -3 …
: continued	Continue using the last list
: allzero	Zero all counters, all lists start at 1 again
: nonil	A special argument. If a macro generates no output, normally <nil> is output. Use nonil if you would prefer to see no output at all in your report in this case. Note: separate from other arguments using a comma, for example: [useful : numeric, nonil]
: blank	A special argument. The macro runs as normal but all output is suppressed. Read the next section on 'MI' and 'MR' for why this is sometimes very useful. It allows you to run macros and collect up the generated list (MI), but not output it. You can then use it in another macro that does generate output (MR).

The reason for providing so many standard keywords for numbering or lettering report output, is that reports will be more consistent across consultants and engineering working with Southbeach.

Indirect effect patterns - using MEMORY (like a calculator)

You have seen how a model can contain one or more reports, and each can have a template which defines the body of the report. Inside the body you can include macros which call the MyCreativity engine to get information from the model into the report. Given the range of patterns it is possible to specify as rules, reports generated from a model can be very sophisticated. However, there is something critical missing - support for indirect effects, in other words, how one agent influences another, which influences another. That is the subject of this section.

Because the MyCreativity engine matches the patterns at the places ‘clicked on’ in the model, you can never get at indirect effects across the model, even if you increase ‘extent’. Suppose, for example, you wanted output for all agents that were increasing other harmful agents that were counteracting goals that were driving towards a focus object in the model. That would make no sense at all for the type of creative output that the MyCreativity rules engine was designed for. However, in a report, it makes perfect sense!

Building up a report, section by section, by generating creative sentences that speak about (or ask questions about) all of the direct or indirect relationships in a model is a powerful way to explain a model to a client. The output generated can also help to ensure that the whole model makes logical sense.

To find such 'indirect' effects in a model, we could have implemented a complex extension to the pattern matching language. This would have made it impossible for anyone other than a 'programmer' to use. Instead, we came up with a simple device that called MEMORY. If you have ever used the MEMORY IN and MEMORY RECALL keys on a calculator you will be able to use it.

Here is a simple report:

> My report on things effecting our goals.
> The factors limiting our goals are:
>
> [decreases(MI, goal) "{from}"]
>
> The factors that are increasing those are:
>
> [increases(, MR) "{from}"]

By placing the keyword MI in the position of an argument in a pattern, it 'picks up' the matching objects. These can then be referred to in another pattern using MR.

MI and MR can be used in conjunction with any of the MyCreativity keywords, for example:

[counteracts(useful+MI,) "{from} is countering {to}"]
What can we do to remove any of these:
[MR]

MI can also be used to pick up effects in the model, as follows:

produces+MI(,) " ....."

And MI and MR can appear in any position in a triple. Thus, it would be possible to find contradictions in model, and then find indirect around those contradictions, and then further side-effects and causes, and so on, forever.

Remember the ADD TO MEMORY key on your calculator, well that is provided here also. It's called MA. By using MA you can accumulate lists of things as a report is processed, and then use them later in the report for another section, or as part of another pattern.

MI, MA and MR are a powerful, but simple, feature of MyReports. Now you can write reports which use indirect effects and not only direct effect patterns.

Using ‘random’ in MyReports

As we explained in previous sections, ‘random’ is a qualifying keyword that can be added to any macro, e.g. {harmful+random}, in order to select a random agent from the list that would otherwise be returned. In MyCreativity, this can be used to generate sentences that contain randomized output. A typical example might be to ask questions about one agent in relation to other randomly selected agents. In MyReports, ‘random’ serves precisely the same purpose, but because of the way reports work, a line of output will be generated for each matching pattern, each line of which will contain randomized content.

Take this example:

> [* “Can {harmful+random} be solved by {useful+random}?” : numeric]

What do you think this does? The * matches all agents in the model, so there will be one possible line of output for every agent in the model, each numbered. However, the output is a sentence that combines a randomly selected harmful agent with a randomly selected useful agent. The sentence will be tried the number of times that matches the *, e.g. the total number of agents in the model, but if the randomly generated sentences repeat each other, some of the output will be suppressed as ‘duplicates’. (This is the same way that MyCreativity always avoids unnecessary duplicate output.) In fact, you might get different number of lines generated each time you update the report (F5).

Don’t be confused … the * may lead you to think there should be one line output per agent in the model, but fewer lines will actually be generated. If you really want one line per agent, then you should include in the output sentence something that is unique, such as the name of the matching agent, for example:

> [* “Can {this} be helped by {useful+random}?” : numeric]

More logically you probably wanted to say:

> [harmful “Can {this} be helped by {useful+random}?” : numeric]

Here is another example:

> [produces(, harmful) “Can {random+useful} be combined with {from} in order to make a new agent which does not produce {to}?”]

The use of ‘random’ can provide powerful report output, especially when combined with values off a grid or tags (@). As an exercise, write a report which asks intelligent questions about the relationship of randomized pairs of strengths, weaknesses, opportunities and threats (SWOT). Since ‘random’ plucks the agents out of the model as a macro, you don’t even need to connect the model using effects in order to generate sets of useful sentences.

Calling a MyCreativity rule-set from a report

A small extension of syntax also lets a report call a MyCreativity rule-set from your library. For example:

```
My report uses my improve and hats rules

Here is everything we need to do:

[* #improve : numbered]

Let's do some Black Hat Thinking on all the useful elements:

[useful #hats.black : questions]
```

Multi line rule-sets and report macros

Just as multi-line sentences and paragraphs are supported in MyCreativity rules:

```
pattern "output text line 1\n
output line 2\n
output line 3"
```

so are multi-line macros in MyReports. But because a report is a text template, you don't need to put in \n you just lay the report out the way you want it. For example:

```
[harmful "Watch out for:
>>>> {this}
we need to remove its cause."]
```

But remember, you get one line or paragraph generated for each match in the model. This repeated text form may not be right in all circumstances. So how about this form instead:

```
Watch out for:
[harmful ">>>> {this}"]
we need to remove their cause(s).
```

In the first form, all the text is inside the macro. In the second form, some of the text was outside the macro in the body of the report template.

Inserting Model Properties into a report

Since the report you enter in a report panel can be used across any model, simply by clicking on the model tab and clicking on the report tab and then hitting the F5 key (Refresh Report), it is sometimes useful to insert the model name into the report, so that the output corresponding to the report is clear. Macros are provided for this as follows:

model.name

model.description

model.perspective

model.url

These macros extract information from the model Properties (right mouse on the canvas, model notes tab, model tab name, etc.)

Thus, a report could begin:

This is my report on {model.name} from the perspective of {model.perspective}:

Background:
[model.description]

The harmful elements include:
[harmful]

The goal is to solve the following problems:
[produces(useful, harmful) “Find an alternative to {from}” : problems]

Running a report across multiple models

A Southbeach Modeller file (.sbm file) can contain several models, each in their own tab, like some spreadsheet programs. It is sometimes useful to generate a report across all the models in the file. For example, you could have been adding actions to each model, building up an action list to guide an improvement project.

Users develop multiple models for all kinds of reasons. Perhaps they are modelling from the perspective of different stakeholders (separation by role) or different aspects of the problem, or versions of a model. The ability to generate a report across all of the models can be very useful. You could use it to find things such as:

All of the harmful effects
All of the contradictions
All of the potential actions

Etc.

The MyReports panel lets you select the scope as:

This model
All models

When you refresh the report (F5) it applies to the currently selected model, or to all models.

Everything you have learnt in this manual can be used in the template and applied across all of the models. When you do this, it is sometimes (but certainly not always) the case that you might need to pull out information, into the report, from a specific model only. For example, some parts of the report use information from all models, and other parts from a specific model in the set. To support this, each model has what is called 'model separation'. In the example above, if the models applied to different stakeholders such as the CEO, CIO, CFO etc., then you could separate those models by 'role'. (Each model tab has a right mouse menu for specifying a separation). In this way, all of the agents in a model inherit (automatically), a separation 'tag' attribute with the name of the model, e.g. @CEO, @CIO, @CFO.

These extra attributes as a result of separating each model in the set can then be used in a report, exactly like any other tag. For example:

The CEO has these problems:
[harmful+@CEO]
The CFO has initiated an action plan:
[produces(@CFO+harmful, action) "{to} : numeric]

Appendix A - How can I learn more?

Try it. Start simple. Don't be afraid. Nothing can break. This is not programming.

Encourage your colleagues to get involved. Share models among a team. Also share creativity rules and report templates to amplify each other's creativity. Learn from each other as you work.

The main help page for Southbeach Modeller starts here:

http://www.southbeachinc.com/help/index.html

(Help menu – Contents)

Connect with other users in our Southbeach community:

http://www.southbeachinc.com/help/community.html

Write to us with questions at support@southbeachinc.com

Provide feedback to feedback@southbeachinc.com

Appendix B - Trouble shooting

Q1: "I cannot see the Creativity or Report panels and tabs"

Have you hidden them? Check the View menu?

Ctrl+K opens the creativity panel
Ctrl+R opens the reports panels
Ctrl+T opens the toolbox which provides the library

If the panels you need are still not available, you may be using an older version of software. Download the latest version here:

http://www.southbeachinc.com/software/download/index.html

You can check whether you are running the latest version:

menu Help - Check for new releases

Q2: "I cannot generate any output"

Try the following:

- If you are writing rules in .txt files for use across several models, have you loaded the creativity? Use the 'Reload MyCreativity' menu option in the right mouse menu of the Creativity tab in the Toolbox panel (Ctrl-T)
- Do your rules match the places in the model where you are clicking? Try a very simple rule first and get that working, such as:

  * "You clicked on {this}"

- Check that the rule name/category is enabled in the Creativity tab of the Toolbox panel.
- If writing model specific rules, check that the '-model' category is enabled in the Creativity tab of the Toolbox panel. This turns on and off 'model specific' rules.
- Check the format of your rules carefully. Any rules that are not correctly formatted will be reported in the errors.log file. This file can be found in the My Documents/Southbeach/MyCreativity folder on your computer.

Q3: Still not getting any output?

Write to support@southbeachinc.com attaching the model and rules.

Appendix C - Tenses & Correct English Form for Effects

In order that it is possible to generate correct English sentences in MyCreativity output and in MyReports, different variations of the macro {effect} are provided as explained in a previous section.

To list all of the variations possible, we used the following creativity:

*(,) "Does {from} {effect} {to}?"

*(,) "{from} {effects} {to}"

*(,) "{from} is {effecting} {to}"

*(,) "{from} {effected} {to} in the past"

*(,) "{to} {effectedby} {from}"

*(,) "{from} is the {effector} of {to}"

*(,) "The {effection} of {to} by {from}"

And then ran it against a model containing an example of all Southbeach effects. It generated the following:

Produces	**Counteracts**	**Prevents**
Does A produce B?	Does A counteract B?	Does A prevent B?
A produces B	A counteracts B	A prevents B
A is producing B	A is counteracting B	A is preventing B
A produced B in the past	A counteracted B in the past	A prevented B in the past
A is the producer of B	A is the counter of B	A is the preventor of B
The production of B by A	The counteraction of B by A	The prevention of B by A
Opposed	**Consumes**	**Detracts from**
Does A oppose B?	Does A consume B?	Does A detract from B?
A is opposed to B	A consumes B	A detracts from B
A is opposing B	A is consuming B	A is detracting from B
A opposed B in the past	A consumed B in the past	A detracted from B in the past
A is the opposer of B	A is the consumer of B	A is the detractor of B
The opposition of B by A	The consumption of B by A	The detraction of B by A
Creates	**Destroys**	**Is A**
Does A create B?	Does A destroy B?	Does A inherit B?
A creates B	A destroys B	A is a B
A is creating B	A is destroying B	A is being a B
A created B in the past	A destroyed B in the past	A was a B in the past
A is the creator of B	A is the destroyer of B	A is the example of B
The creation of B by A	The destruction of B by A	The description of B by A

Related	**User Defined effect**	**Specifies**
Does A relate to B?	Does A affect B?	Does A specify B?
A is related to B	A affects B	A specifies B
A is relating to B	A is affecting B	A is specifying B
A related to B in the past	A affected B in the past	A specified B in the past
A is the relative of B	A is the affector of B	A is the design of B
The relation of B by A	The effect of B by A	The specification of B by A
Implements	**Contributes to**	**Stores**
Does A implement B?	Does A contribute to B?	Does A store B?
A implements B	A contributes to B	A stores B
A is implementing B	A is contributing to B	A is storing B
A implemented B in the past	A contributed to B in the past	A stored B in the past
A is the implementer of B	A is the contributor of B	A is the store of B
The implementation of B by A	The contribution of B by A	The containment of B by A
Becomes	**Replaces**	**Causes**
Does A become B?	Does A replace B?	Does A cause B?
A becomes B	A replaces B	A causes B
A is becoming B	A is replacing B	A is causing B
A became B in the past	A replaced B in the past	A caused B in the past
A is the past form of B	A is the replacement of B	A is the cause of B
The transformation of B by A	The substitution of B by A	The causation of B by A
Uses		
Does A use B?		
A uses B		
A is using B		
A used B in the past		
A is the user of B		
The use of B by A		

Note - using macros such as {effection} and others cannot fully guarantee to produce the right form of words in every sentence form you use. With care, you can develop creativity sentences that are usable across all Southbeach effects, without repeating the creativity. But this will throw up some 'oddities'. For example:

{from} is the {effector} of {to}

would generate this for a 'contributes to' effect:

A is the contributor of B

It should be:

A is a contributor to B

Whereas the very same rule applied to a 'produces' effect generates this:

A is the producer of B

which is correct! So sometimes, the form of sentences you choose will lead to oddities of expression if you are trying to write the sentences to apply to all possible Southbeach effects.

Where necessary, repeat the creativity rule and build in the match of the effect type, for example:

produces(,) "{from} is the producer of {to}"

contributes(,) "{from} is a contributor to {to}"

Similar considerations apply in NOT forms of effects. Standard NOT forms are impossible because the NOT sometimes has to appear in different places in the sentences. For example:

A does NOT produce B

In that case, the words 'NOT' and 'produce' are next to each other. So you could imagine them being generated automatically for any effect type. But that's not the case here:

A is NOT the store of B
A did NOT cause B in the past

In the first case, the author of the rule wants the word 'the' between the NOT and the 'store' effect. In the second case, the NOT has led to a required change of tense.

Therefore, to encode 'NOT forms', we suggest using separate rules and include the word 'NOT' where you need it in the sentences, for example:

produces+NOT " {to} does NOT {effect} {to}"

In other words: macros such as {effects} and {effection} will not insert 'NOT' automatically for you!