# Table des matières

Basics .................................................................................................................. 3  
Installation ......................................................................................................... 3  
User interface basics .......................................................................................... 3  
Keyboard shortcuts .............................................................................................. 3  
Knowledge representation ................................................................................... 5  
Vocabulary .......................................................................................................... 5  
  Concept types hierarchy ...................................................................................... 10  
    Insert new concept type .................................................................................. 11  
    Concept type hierarchy control ...................................................................... 12  
    Forbidden types ................................................................................................ 15  
    Concept type alteration .................................................................................... 16  
    Graph layout and coloring .............................................................................. 16  
  Relation types hierarchy ..................................................................................... 19  
    Insert new relation type .................................................................................. 19  
    Relation type hierarchy control ..................................................................... 21  
    Relation type alteration ................................................................................... 22  
    Graph layout and coloring .............................................................................. 23  
Individuals ........................................................................................................... 25  
Rules .................................................................................................................... 26  
Constraints ........................................................................................................... 29  
Facts ..................................................................................................................... 31  
  Insert new concept ............................................................................................. 32  
  Insert new relation ............................................................................................. 33  
  Coreference ........................................................................................................ 33  
  Reduced edition ................................................................................................. 33  
Queries ................................................................................................................... 33  
Reasoning .............................................................................................................. 33  
  Inspecting facts .................................................................................................. 33  
  Applying rules .................................................................................................... 33  
  Sum and normalization ...................................................................................... 34  
  Querying .............................................................................................................. 34  
Import, Export and Convert .................................................................................. 34  
  To/From RDF(S) and OWL ............................................................................... 34  
  To/From Datalog+ ............................................................................................... 34  
Building documents ............................................................................................ 34  
  Build vocabulary documentation .................................................................... 34  
  Build vocabulary views ..................................................................................... 34  
Extending CoGui .................................................................................................. 34  
Scripts .................................................................................................................... 34  
Plugins .................................................................................................................. 34
Basics

This new version 3 migrates CoGui from a classic Java application to a completely different architecture based on the NetBeans Platform. If you discover CoGui, a good way is to follow the Getting Started section. CoGui 2 will always be available for download here: http://www.lirmm.fr/cogui/cogui_2.0b6.jar but it is recommended to install the new version. All the instructions about the installation are available in the Installation section.

The application CoGui installed will allow you to automatically update CoGui over the corrections and improvements.

If you have some old CoGui projects from before v3.0 (COGX ML format) you can import them to the new project format, read more about projects import in the Import, Export and Convert section.

Installation

User interface basics

Keyboard shortcuts

Finding, Searching, and Replacing
Ctrl-F3  Search word at insert point
F3/Shift-F3  Find next/previous in file
Ctrl-F/H  Find/Replace in file
Ctrl-Shift-F/H  Find/replace in projects
Alt-Shift-H  Turn off search result highlights
Ctrl-R  Rename
Ctrl-U, then U  Convert selection to uppercase
Ctrl-U, then L  Convert selection to lowercase
Ctrl-U, then S  Toggle case of selection
Ctrl-Shift-V  Paste formatted
Ctrl-Shift-D  Show Clipboard History
Ctrl-I  Jump to quick search field
Alt-Shift-L  Copy file path

Opening and Toggling between Views

Ctrl-Tab  (Ctrl-)  Switch between open documents by order used
Shift-Escape  Maximize window (toggle)
Ctrl-F4/Ctrl-W  Close selected window
Ctrl-Shift-F4  Close all windows
Shift-F10  Open contextual menu
Ctrl-PgUp/PgDown  Switch between open documents by order of tabs
Ctrl-Alt-T  Reopen recently closed file

Editing with graphical editors
Knowledge representation

CoGui works on a model of a knowledge base consisting of:

The ontological part
- A unique and necessary Vocabulary
- A set of Individuals
- A set of Rules
- A set of Constraints

Data organized into
- A set of Facts
- A set of Queries

Vocabulary

CoGui is able to create multilingual ontologies designed for Conceptual Graphs (CGs). A CG Ontology is composed of exact knowledge and contextual knowledge. The vocabulary is one important part of the exact knowledge and consists of two hierarchies:
1. a hierarchy of concept types (also named concept or class or object type)
2. a hierarchy of relation types (also named relation) with arity greater or equal to 1.

The above hierarchies are respectively organized in partially ordered sets (not necessarily a tree or a lattice). The exact knowledge of the ontology, apart from the vocabulary, consists of:
- a collection of individuals
- rules

Editors allow end users to navigate through the ontology and edit graphically its structure and content. The ontology is controlled and, if necessary, tools are provided to correct it.

Graphically, types are displayed as vertices. An arc connecting vertex A to vertex B means that the type A is a kind of type B (or A is a specialization of B or B is a generalization of A):
In most cases the ordered set looks like this:

![Simple ordered set of concept types](image)

In this case, the hierarchical structure is a tree. But the model accepts extra connections. Two examples below illustrate hierarchies that not have a tree structure:

![Hierarchies with extra connections](image)

The edit operation is not heavily constrained by the model, in practice, the only critical error occurs when a circuit is detected. More details can be found in following chapters.

**How to browse through type hierarchies**

When a project is opened (or created) a vocabulary panel appears on the right part of the main window. Concept types, relation types and Individuals are displayed in three separated tabbed panels. An arborescent representation containing every path between maximal type and others. Types are alphabetically sorted, relation types are also sorted by arity.
The tree representation is useful to create vertices in conceptual graphs by dragging types into the graph editor (see Graph Edition chapter). Please remember that the type's order is not necessarily a tree. That's why the same type may be retrieved several times in the tree representation. For the same reasons tree is not automatically expandable if hierarchy contains at least one circuit.
Click right button and choose 'Graphical Editor'

The graphical editor is opened
two synchronized editors for a same type hierarchy (both concepts and relations)

you can navigate between both representations:

- a simple double click on a vertex in left panel select and show the (unique) corresponding vertex in the hierarchy view
- a right click on vertex displays a popup menu: the 'Navigate/Show type in tree' action selects (and scrolls if necessary) the corresponding node(s) int the left panel.
Double click will open graphical editor and scroll to make the vertex visible

From the graphical editor to the tree editor

Two other options show parents or children inside the graph (the scrolling process is automatically performed).

- Shows parent vertex in the graph representation and select them
- Shows children vertex in the graph representation and selected them

Concept types hierarchy

- Insertion
- Graph arrangement
- Concept type hierarchy control
- Forbidden types
- Concept type alteration
Insert new concept type

A newly created concept type hierarchy contains only one type named 'Top'.

Insert button on the toolbar of the graphical editor can be used

'Insert concept type' into the popup menu inserts the desired type at the mouse location.

A concept type can also be created into the tree representation. 'New concept type' option in the type view popup menu creates a new concept type as a type of selected item:

A concept type can be created directly into the tree

Newly created type is kind of previously selected item

Into the graph editor, the following vertex is displayed: conceptType1. Click twice on the vertex to edit
type name. Concept types can be renamed directly on the concept type tree. Click once on the tree item to edit type name. Both actions have same effect and are synchronized.

**Naming convention**

Homonymous types are not allowed in the same type hierarchy. The case is respected but comparisons are case unsensitive. For instance user can decide to write 'Dog' or 'dog' but cannot define both words in the concept type hierarchy. Blank spaces are allowed.

**Concept type hierarchy control**

Action to control of the concept types hierarchy is provided in Debug menu:

Only one critical error can occur with the graph structure: the detection of a circuit. Assume that 2 types A and B on a circuit, i.e. a path exist from A to B and another exist from B to A. It means A is a kind of B and B is a kind of A.
CoGui control detects circuits: all the animals are not dogs

However, it is possible for a project consisting of several pieces of ontologies to work with several synonymous types from multiple equivalent URIs (owl: SameAs). Despite the warnings CoGui is able to work with the circuits, all the concept types belonging to the same circuit are considered as equivalent.

Another model constraint is that the concept types hierarchy must have a maximal concept type. By default CoGui names it 'Top', feel free to change its name or to choose another vertex as the maximal. A warning message occurs if the hierarchy contains more than one maximal element. The tool does not automatically add a maximal abstract type to the hierarchy but it is recommended to respect this constraint.
A single maximal type is required

Another warning can occur when the user draws redundant arcs. If A is a kind of B and B a kind of C, by transitivity CoGui 'knows' that A is a kind of C, hence an arc between A and C is correct but redundant. These extra arcs can obstruct the graph view but extra entries on the tree representation could be used as sort of shortcuts for often used types. That is the reason why CoGui accepts and stores redundant arcs. When a message (error or warning) occurs, it can hold the action to solve it. The Repair box checked indicates that a repair action is available:

In this case all the redundant edges can be removed with the message popup menu with the action named 'Transitive reduction'.

Forbidden types

In a graph, the concept vertices may be associated to a conjunctive types, meaning it has several types. As a result, the model provides a mechanism to prohibit some incongruous associations. For instance, suppose you have defined both "Animal" and "Plant" concept types, you might want to prohibit associations between these types as well as between sub-types of them. It is possible to express this restriction in your concept type hierarchy. To this end, you are going to introduce a forbidden type in the concept type View to express this incompatibility. See below an example of such restriction expressed in the concept type hierarchy triggers and here it triggers an error in a conceptual graph.

Forbidden types (also named disjoint types) can be added in the view placed below the concept type tree:
It is possible to create sets of 3 conjunctive types or more. If the forbidden type (A;B;C) is defined, all subset will be forbidden (A;B) (B;C) (A;C) and of course (A;B;C). This corresponds to the most frequent needs of the users. If you want to specify that only the conjunction (A;B;C) is forbidden a negative constraint can be used.

All subsets of A,B,C with card>1 is forbidden

Negative constraint for A;B;C

Concept type alteration

As for relations, adding concept types does not affect the existing ones. The deletion of a concept type can affect not only the type hierarchies and fact graphs, but also the signatures of relations. All references to this type must first be removed from the base.

Removing a link between two concept types does not create inconsistencies in the knowledge base but can decrease the number of answers to a query, adding a link can increase the number of answers, the forbidden types may change and some constraints may become unsatisfied.

All consequences of these alterations are detected by the CoGui controller and error messages help the user to correct inconsistencies.
Even if the position and color of the vertices of the graph do not matter in the representation model, they can be very useful for the user.

A vertex is moved by dragging its box with the mouse. Another way to place vertices is to run an automatic arrangement with the layout algorithms.

A dynamic force directed layout is also provided:
The default color of the concept types can be changed using Tools/Options/CoGui/Appearance command:

A different color can be selected for each concept type. Contextual menu of the vertex propose a submenu.
'Coloring':

Created with the Personal Edition of HelpNDoc: Full-featured multi-format Help generator

Relation types hierarchy

- Insert new relation type
- Relation type signatures
- Graph layout and coloring
- Relation type hierarchy control
- Relation type alteration

Created with the Personal Edition of HelpNDoc: Produce Kindle eBooks easily

Insert new relation type

A newly created relation type hierarchy contains only one type named 'Link'.
Use button or popup menu to add a new relation type

The relation type can be updated with 'Label & signature...' menu action:

Homonym types are not allowed in the same type hierarchy. The case is respected but comparisons are case insensitive. For instance the user can decide to write 'Binary_rel' or 'binary_rel' but cannot define both words in the relation type hierarchy. Blank spaces are allowed.

A signature must be associated with each relation type. A signature is an ordered list of concept types (numbered from 1 to arity) where arity denotes the arity of the relation type, i.e. its number of arguments. The signature dialog box allows to change the arity and to specialize involved concept types.

Press the assistant button and choose arity or directly edit arity number and press the 'Apply' button to confirm. Lines are added or removed from the table. Each concept type can be changed directly or with the assistance button:
Relation type hierarchy control

Similar to concept type hierarchy, circuits are forbidden. The only difference with concept type hierarchy is due to signatures. The constraints are:

1) relation types are grouped by arity. Each 'arity family' must have a maximal element. It means that the relation type hierarchy is decomposed w.r.t. the arity and a unique maximal element is required for each of these sub-hierarchies.

2) Let A and B be two relation types in the same sub-hierarchy (i.e. A and B have the same arity). If A is a kind of B, it means that every concept type in A signature is respectively a kind of concept type in B signature. For example if graze(herbivore,plant) is kind of eat(animal,food) their signatures respect compatibility if herbivore is a kind of animal and if plant is a kind of food.
graze(herbivore,plant) is kind of eat(animal,food)

Another way to quickly define or pre-define signatures is to use 'Suggest signature...' command on popup menu. Use the command after a link is established between a new relation type and an immediate greater relation type: the parent signature is automatically proposed. But this command is designed for more a complex purpose. In a complex ontology it becomes difficult to define a new relation type signature. The command 'Suggest signature...' can help to find the maximal compatible signature.

Relation type alteration

Relation type labels can be changed. With respect to the signature covariance new relations can also be added without consequences for existing knowledge.

For obvious reasons of referential integrity, the removal of a relation assumes that all occurrences have disappeared from all graphs, both in annotations and within ontology.

The consequences of the change of a signature depends on its nature: if the arity of the signature is changed, all occurrences of the relation will require user's intervention; if only the concept types of the signature are changed, then it will be a different signature.

If a concept type is replaced by a more general type, the content of the knowledge base will not be affected, and no error will occur. However, if a term is specialized, it can have an effect on the content of the knowledge base, and can also trigger errors in the annotations. Removing a link between two relation types does not create inconsistencies in the database but can decrease the number of answers; adding a link can
increase the number of answers (new rules may be applicable) and some constraints may become unsatisfied.

Graph layout and coloring

Even if the position and color of the vertices of the graph do not matter in the representation model, they can be very useful for the user.

A vertex is moved by dragging its box with the mouse. Another way to place vertices is to run an automatic arrangement with the layout algorithms.

A dynamic force directed layout is also provided:
The default color of the concept types can be changed using Tools/Options/CoGui/Appearance command:

A different color can be selected for each concept type. Contextual menu of the vertex propose a sub-menu ‘Coloring’.
Individuals

An individual is an identifier which is a surrogate for a precise entity of the discourse universe. For instance, if Town is a concept type then Budapest is an individual of type Town. A concept type may have subtypes, e.g. SmallTown could be a subtype of Town, an individual cannot have 'subindividuals'. The ontological individuals are the individuals about which all the users agree, i.e. for all users an ontological individual must represent the same entity in the discourse universe. An ontological individual is entered into a COGUI-ontology with a primitive concept type called its privileged type. For instance, if the COGUI-ontology concerns Modern Art, and if Picasso is an ontological individual of privileged type Artist representing the famous artist Pablo Picasso, then it is impossible to use the identifier Picasso for representing a Citroën car (unless the conjunctive type Car, Painter is not forbidden). COGUI checks that an ontological individual appearing in an annotation has a type which is compatible (i.e. not forbidden) with the privileged type of the individual.

All individuals appearing in a COGUI-ontology (e.g. in rules, constraints or prototypical knowledge) must be ontological individuals. Thus, the set of ontological individuals can be completed only whenever all knowledge representing in a COGUI-ontology have been considered.

The individual view lists all individuals in a sortable table. Needless to complete, the list of individuals automatically updates when the user references individuals in different graphs. Select individuals in the list to drop them to the graphs and right click to popup the contextual menu to rename, change the privileged type.
Individuals tab in vocabulary view displays the complete list of individuals

Since 3.0 CoGui integrates the notion of namespace. Then a namespace can be associated the each individuals. It can be selected directly on the list:

Rules

To create a new rule or edit an existing rule go on the projects view and use the popup menu (right-click).

As other graphs, rules can be organized in folders. It is particularly interesting to classify the rules on families, especially for testing purposes. Editing rules is very similar to editing Facts. But the split editor is divided in two parts: hypothesis and conclusion. By default hypothesis is placed on the left part and conclusions on right part of the editor. The split bar can be oriented with the mouse to horizontal position, in this case hypothesis is placed on top and bottom is reserved for conclusion.

Create a new rule:

If “Rule...” action does not already appears in menu choose “Other...” and select the type of CoGui object that you want to add in your project:

Rules are used to represent implicit (common sense) knowledge. For instance, let us assume that the fact that Eve is the mother of Abel is represented in a fact graph. If the ontology contains a rule saying that if \( x \) is the mother of \( y \) then \( y \) is a child of \( x \) then the system can automatically add the information that Abel
is a child of Eve.
Such a rule is represented by two simple graphs. One represents the hypothesis (e.g. \([\text{woman}]-1-(\text{mother of})-2-\text{[human]}\)) the other represents the conclusion (\([\text{human}]-1-(\text{child of})-2-\text{[woman]}\)). Furthermore, there is a link between the first person in the hypothesis and the second person in the conclusion, and a link between the second person in the hypothesis and the first person in the conclusion. A rule “if A then B” is used as follows: if an annotation contains A then B can be added to the graph. See below the graphical representation of this rule:

A bi-colored representation of this rule could be simplest with just a conclusion relation “child of” added to the hypothesis. The advantage of this representation is that it allow specialization of a concept type in the conclusion. For instance, from the hypothesis \([\text{human}]-1-(\text{mother of})-2-\text{[human]}\) it can be deduce that the first person is a woman. Thus, rule represented below is more powerful than previous example:
CoGui User Guide

rules could generate some specializations. Here Person is eventually specialized in Woman.

But it is better to express each deduction in a separate rule and add a new rule to the first one:

Do not worry about rule applying order, even if the first rule is not applied in a first step, it can be applicable after applying another one. All rules are tried until saturation (when no rule is applicable).

Due to saturation, rules must be built carefully when at least one rule can create new concepts: The rules may loop and cause an infinite production of concepts. Example below is trivial but combinations of rules can generate loops very difficult to detect.
This rule will cause a loop

Fortunately it is possible to interrupt the saturation running task if it did not end after a "reasonable" time. And the knowledge base can be queried without using saturation. See Applying rules section and Querying for more about the use of rules.

Created with the Personal Edition of HelpNDoc: Produce electronic books easily

**Constraints**

Constraints allow to define pieces of information that are forbidden in the facts (negative constraints) or mandatory in the facts (positive constraints).

To create a new constraint or edit an existing constraint go on the projects view and use the popup menu (right-click).

You will find all the information about the use of the constraints are in section Inspecting facts.

Create a new constraint:

Select the type of Constraint that you want to add in your project:
**Negative constraints**

A negative constraint is a simple graph expressing a condition that must not appear in checked facts. Checking a negative constraint is similar to query facts. Facts are validated if no homomorphism of the constraint graph can be found into them.

A negative constraint:

```
<table>
<thead>
<tr>
<th>hasFlavor</th>
<th>WineFlavor : Strong</th>
</tr>
</thead>
<tbody>
<tr>
<td>wine : *</td>
<td></td>
</tr>
<tr>
<td>hasSugar</td>
<td>WineSugar : Sweet</td>
</tr>
</tbody>
</table>
```

There is no wine that is both sweet and strong.

**Positive constraints**

A positive constraint is a structured as a rule with a condition part and the obligation part. A fact satisfies a positive constraint if every homomorphism from the condition part to the fact can be extended to a homomorphism of the obligation part to the fact. The example below expresses the fact that "a wine is necessarily associated with a winery". Positive constraints will be triggered each time a wine appears without a Winery attached to it with hasMaker relation.
Facts

A Fact is labeled bipartite graph. One class of nodes (the concept nodes) is used to represent entities of the discourse universe. A concept node is labeled by a concept type (e.g. Painter, or a conjunctive type such as Painter,Catalan) and, possibly, by an individual (e.g. Picasso). A concept node which is labeled by a concept type without an individual is called a generic concept node. Such a node represents an unidentified element of the type. For instance, contrarily to a node labeled [Painter : Picasso] representing the painter Picasso, a concept node labeled [Painter] represents a painter.

The second class of nodes represents the relationships between the entities (represented by the concept nodes). For instance, if Guernica is an individual representing the well-known painting realized by Picasso then a relation node labeled (hasPainted) could relate the concept node [Painter : Picasso] to the concept node [Painting : Guernica]. The edge between (hasPainted) and [Painter : Picasso] is labeled by 1, and the edge between (hasPainted) and [Painting : Guernica] is labeled by 2. This edge labeling is used to represent different roles (e.g. to distinguish the subject from the complement). It is also possible to say that two different concept nodes represent the same entity by linking them by a coreference link.

Picture below is described by a fact graph:
Create a new fact:

If "Fact...” action does not already appears in menu choose "Other...” and select the type of CoGui object that you want to add in your project:

Concept Insertion
Relation insertion

Created with the Personal Edition of HelpNDoc: Easily create PDF Help documents

**Insert new concept**

There are many ways to create new concepts.

- Insert concept button on toolbar
- Insert concept from editor popup menu
- Drag a concept from the type hierarchy

The third way (drag and drop) is the most effective since it informs in a single action the position and type of the new concept.
Click twice on the concept vertex or use the popup menu to edit the newly created concept:

**Insert new relation**

In

**Coreference**

**Reduced edition**

**Queries**

**Reasoning**

**Inspecting facts**

**Applying rules**
CoGui User Guide

Created with the Personal Edition of HelpNDoc: What is a Help Authoring tool?

**Sum and normalization**

Created with the Personal Edition of HelpNDoc: Easily create Web Help sites

**Querying**

Created with the Personal Edition of HelpNDoc: Full-featured EBook editor

**Import, Export and Convert**

Created with the Personal Edition of HelpNDoc: Free PDF documentation generator

**To/From RDF(S) and OWL**

Created with the Personal Edition of HelpNDoc: Create HTML Help, DOC, PDF and print manuals from 1 single source

**To/From Datalog+**

Created with the Personal Edition of HelpNDoc: Easy CHM and documentation editor

**Building documents**

Created with the Personal Edition of HelpNDoc: News and information about help authoring tools and software

**Build vocabulary documentation**

Created with the Personal Edition of HelpNDoc: IPhone web sites made easy

**Build vocabulary views**

Created with the Personal Edition of HelpNDoc: Write EPub books for the iPad

**Extending CoGui**

Created with the Personal Edition of HelpNDoc: Free EPub and documentation generator

**Scripts**

Created with the Personal Edition of HelpNDoc: Free EPub and documentation generator

**Plugins**