Short introduction to the software package "Check-Coherence"

vers. Epsilon

Andrea Capotorti & Barbara Vantaggi

August 30, 2005

1 Introduction

Check-Coherence software have been designed to check the coherence (consistency) and to perform extensions (inferences) of probabilistic models based on partial conditional lower-upper probability assessments. Of course precise probabilistic models are included as special cases.

The package mainly consists of two parts:

- a user interface, designed in Visual-C++, to input by keyboard or to load by file the model to check/extend;
- an inferential "engine", designed in Xlisp-Stat, that actually performs the computations.

Even if the user cannot interact with the computations, a window will appear to give a trace of the operations there are going on and it will be closed at the end.

The algorithm on which the software has been designed, is that one based on the theory recently fully described in Coletti & Scozzafava's book [2]. In particular, the specific algorithm has been described in its more general version in [1] and the messages that appear in the computation window refer to that notation.

The optimizations tasks are performed by a GNU LGPL (Lesser General Public Licence) procedure named `lp_solve` (Version 5.1.0.0 dated 1 september 2004). It is based on the revised simplex method and the Branch-and-bound method for the integers. It has been developed by Michel Berkelaar, Kjell Eikland and Peter Notebaert. Please refer to `http://groups.yahoo.com/group/lp_solve/files/Version5.1/` for its specific distribution and license policies.

This short guide is intended to help those who are going to use the interface for the first time. The figures shown in this guide refer to the example presented in [1].

2 Hardware requirements and installation

The package has been implemented to run under any MS Windows 32-bit platform. Anyhow, the most recent tests have been performed only on PC under Winwos XP OS.

To install the package it suffices to transfer the compressed archive Setupckc.zip to a suitable location and to extract all the files.

The archive contains both the "check-coherence" package (Epsilon version) and the public-domain Xlisp-Stat[1] MS Windows 32-bit version.

To run the program double-click on the "interf.exe" icon. An information "About CHK" window will appear and after few seconds the user-interface window will be available.

3 User interface description

The user-interface window mainly consists of four parts (as shown in Fig. 1)

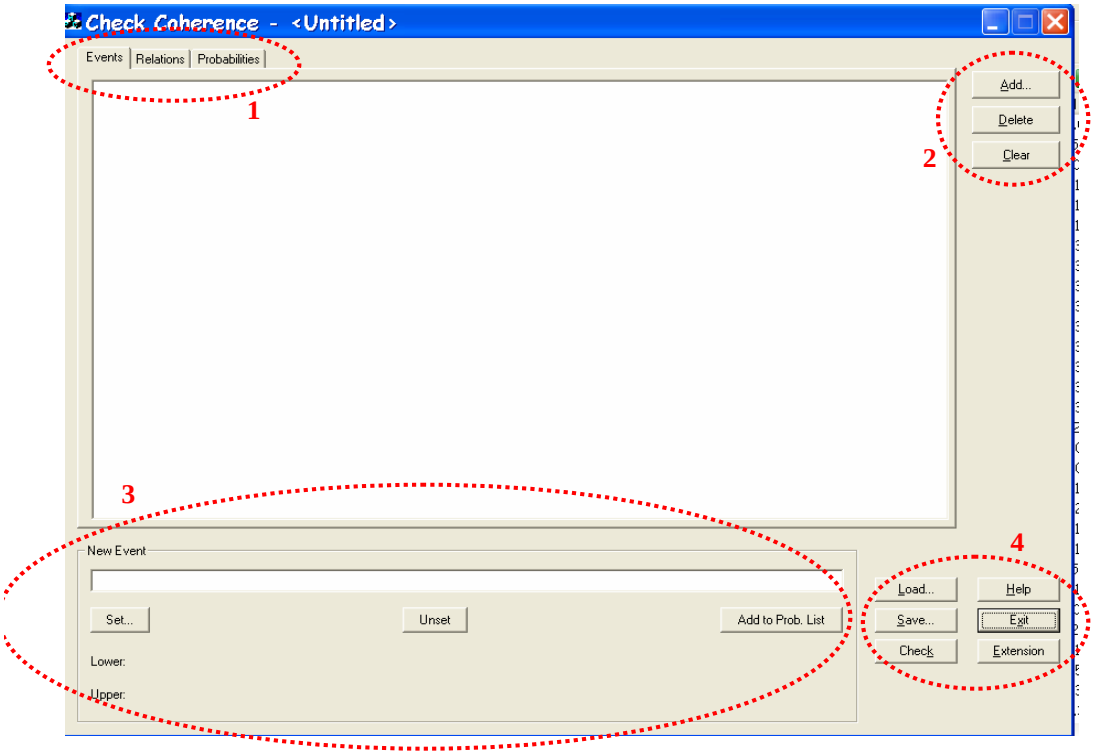

Figure 1: User-interface window with the four main components highlighted

1. the three tabs `Events, Relations` and `Probabilities` containing the model data, where
   - `Events` contains the elements through which the conditional events of the domain are built;
   - `Relations` contains the logical constraints among the elements of `Events`
   - `Probabilities` contains the conditional events representing the domain of the assessment with associated lower/upper probability values;
2. the editing buttons `Add, Delete, Clear`, shared by all the tabs, and the `Edit` button , present in the `Relations` and `Probability` tabs, where
   - `Add` is to adjoint any new item to the active tab;

[1]XLISP-PLUS version 3.04 Portions Copyright ©1988, by David Betz. Modified by Thomas Almy and others. XLISP-STAT Release 3.52.17 (Beta). Copyright ©1989-1999, by Luke Tierney

- `Delete` is to erase a selected item from the active tab;
- `Edit` is to modify a selected item in the active tab;
- `Clear` is to erase *all* the items listed in the active tab;

3. the extension I/O part in the bottom-left corner, where
   - `New event` is a field where the element of interest for inference appears;
   - `Set` is to input the new conditional event;
   - `Unset` is to erase `New event` field;
   - `Lower` and `Upper` are the fields where the result of the extension appears;
   - `Add to Prob. List` is to automatically adjoint the result of the extension (inference) to `Probabilities` tab;
4. the function buttons `Load, Help, Save, Exit, Check` and `Extension` on the bottom-right corner, where
   - `Load` permits to input by file an already prepared example (the package contains a subfolder "examples" with some models taken from the respective papers listed in Section 5);
   - `Help` does not work (in the future it will launch an on-line tutorial);
   - `Save` is to save[2] the active example on any suitable location;
   - `Check` is to start the computation to check the coherence of the model[3];
   - `Extension` is to simultaneously check the coherence of the model and to compute coherent lower/upper bounds for the element in `New event`. It must be used only if `New event` field has been previously set.
   - `Exit` is to close the interface;

4 Model input and functions use

The first part of the model to input is `Events` tab because all other parts will refer to the items listed on it. Anyhow, at any time it will be possible to go back to modify such list.

4.1 How to input the elements of the "Events" tab

By clicking on `Add`, a dialog window appears with a field where to input an event label (name). Labels can be composed with any standard character, except for special symbols and blank spaces. Hence multiple names labels must input with some auxiliary character, e.g. the event "Low pressure" must be input as `Low_pressure`.

Note that the user can choose if to input or not specific labels for composed events like $(e_1 \wedge \neg e_2) \vee (\neg e1 \wedge e_2)$. This because in the other parts it will be always possible to compose any logical combinations of the elements listed in `Events` tab. The choice will mainly depend on the readability of the model.

Warning: deleting a label by the `Delete` button, all the logical relations and the probabilities assessments involving it will be erased in the other tabs.

[2]Note that examples must be saved with the extension .ckc otherwise they will not be recognized by the Load function

[3]The Check function will ignore the New Event field, if present

4.2 How to input the logical relations

This is the only part that differs from usual inputs procedures. Anyhow, once the user will become familiar with it, he/she would appreciate the possibility to input also very nested logical constraints among the elements listed in the `Events` tab.

By clicking on `Add`, a wide dialog window appears, as shown in Fig. 2. The first step

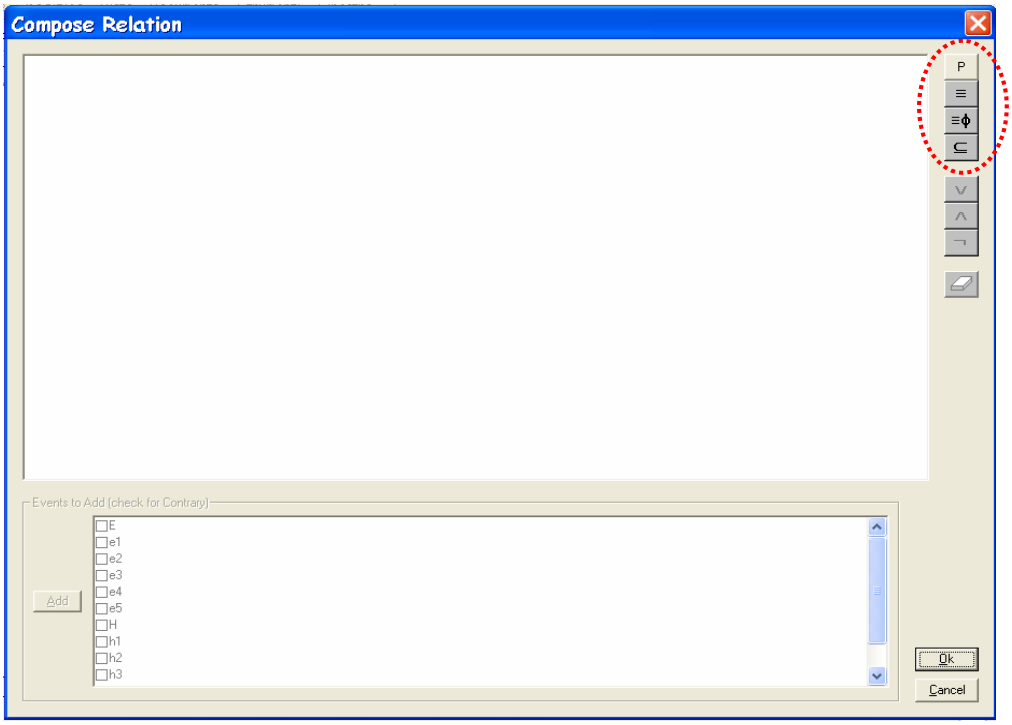

Figure 2: Dialog window to input the highlighted logical relations

consists in choosing one of the possible relations among those listed on the up-right corner of the window:

- `P` Partition (to be used whenever there is a list of events forming a partition of the sure event)
- $\equiv$ Identity (to be used whenever there is an event actually equivalent to a combination of others, e.g. $e_i = e_j \vee e_k$)
- $\equiv \phi$ Incompatibility (to be used whenever a particular combination of events is logically impossible, e.g. $e_l \wedge e_m = \emptyset$)
- $\subseteq$ Inclusion (to be used whenever there is an event or a combination of events that logically implies an other event or an other combination of events, e.g. $h_i \subseteq h_j$)

Once a relation has been chosen, a tree with one or two branches appears, depending on the number of inputs requested by each relation. Simultaneously the logical connectives listed on the right become active.

In each branch of the tree it is possible to input

- a list of events among those listed at the bottom of the window (for negations just select the check-boxes on the left of the labels) by selecting them and by clicking `Add`;
- a logical connective ($\neg$ for negation, $\vee$ for disjunction and $\wedge$ for conjunction) by clicking the respective button listed on the right.

Note that in each branch there is an empty leaf to add other sub-branches or events.

The logical relation that is going to be built must be read following the tree structure. For example, the tree shown in Fig. 3 is to represent the logical equivalence

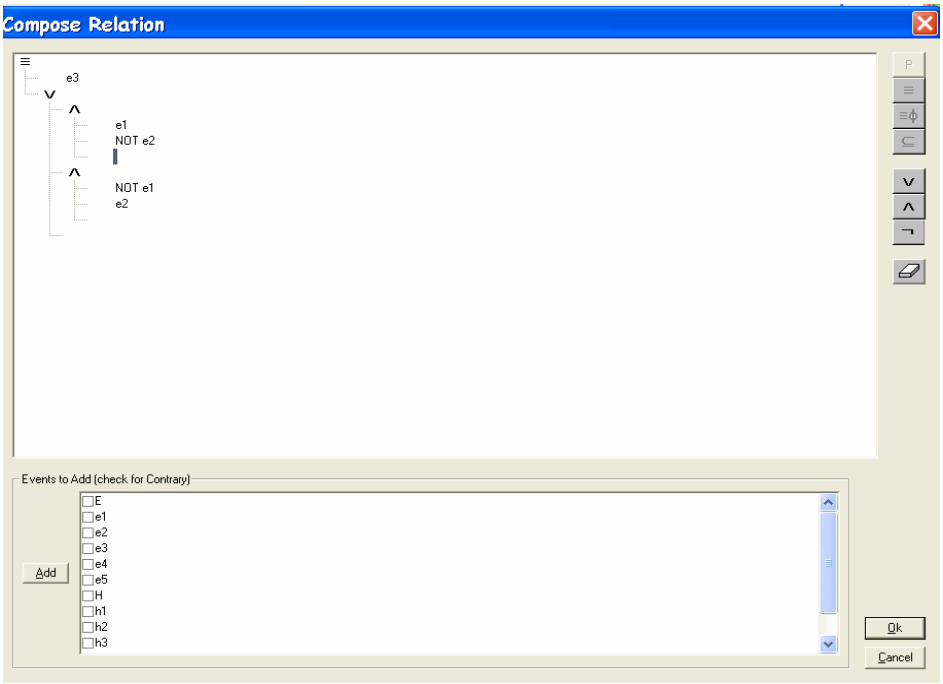

Figure 3: An example of a logical relation tree

$$e_3 \equiv (e_1 \wedge e_2^c) \vee (e_1^c \wedge e_2).$$

Of course, using iteratively the different connectives it is possible to build much more nested relations.

The "eraser" icon (listed just below the logical connectives) is to prune the branch where the prompt has been previously located.

When the tree is complete, click on `OK` at the bottom-right corner to automatically obtain on the `Relations` tab the desired logical constraint, as shown in Fig. 4

Any logical constraint can be modified or deleted by selecting it and clicking on `Delete` or `Edit` , respectively.

4.3 How to input the initial assessment

In the `Probabilities` tab it is possible to add, modify or delete a probability assessment. Each element of the assessment can be conditional or unconditional, precise or imprecise (lower/upper).

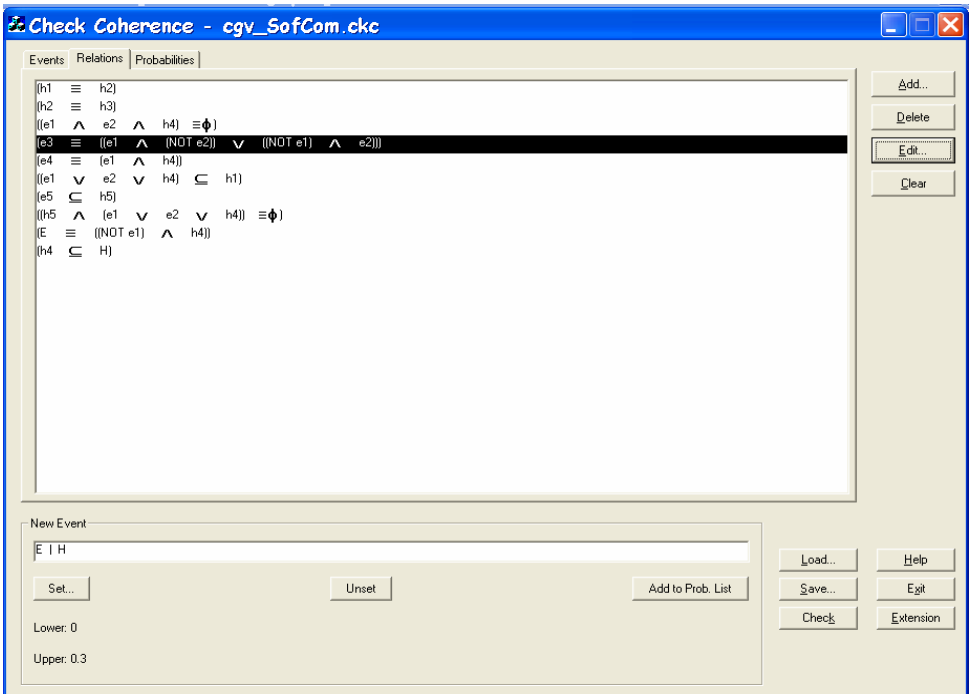

Figure 4: As the logical relation of Fig. 3 appears in the "Relations" tab

By clicking `Add` , the "Probability" dialog window shown in Fig. 5 appears.

If we refer to a generic conditional event $H|E$, for the conditioned element H you can either select one of the basic events listed in the `Hypothesis` field (once again for negation select the check-box) or compose a more complex event by clicking `Composed Ev.`. In the latter case, a dialog window similar to that one of the logical relations, will appear. The way to use the logical connectives is the same of that already described in the previous subsection.

For the choice of the conditioning element E, the rules are the same, with the addition that in the `Assumed info` list it is available the <Unset> label to be used for unconditional values.

In the fields `Lower prob.` and `Upper prob.` it is possible to input the respective values. For a precise probability it is enough to input only the first field. The numerical values can be either in decimal or in rational form.

At the end of the values input, click on `OK` and the element will be added to the `Probabilities` tab.

`Delete` and `Edit` work like in the other tabs.

4.4 Check and extension functions

When the model has been fully input, it is possible to check its coherence by clicking the function button `Check`. The Xlisp-Stat will automatically start the computation loading the needed modules and showing, time to time, the flow of the inner elaboration. At the end the Xlisp-Stat will automatically close and return the result ("Coherent" or not "Not Coherent") to the interface.

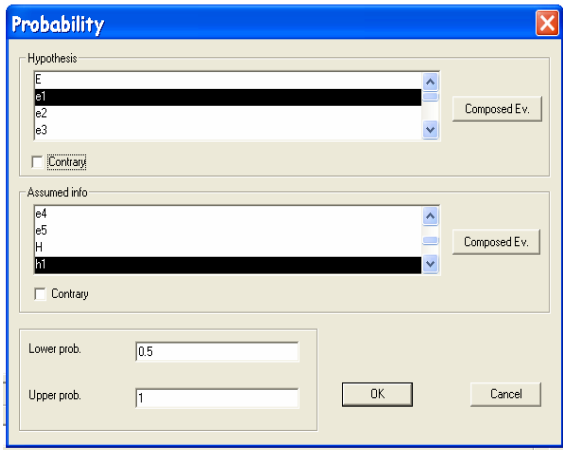

Figure 5: The probability dialog window

Otherwise, it is possible to simultaneously check the coherence of the model and, in the case of coherence, to compute the extension for an inference target. The inference target must be set into the `New Event` field by clicking `Set` button. The input rules are the same of the input of the initial assessment, with the obvious absence of the lower/upper probability fields.

By clicking `Extension` the computation will start in a similar way of that one depicted before . At the end now the possible result can be either a message of incoherence or the coherent bounds for the probability of the new event. Such bounds will appear in the bottom-left corner. The initial assessment can be enlarged with these values just by clicking on `Add to Prob. List`.

4.5 Other functions

The Save, Load and Exit functions work like in all the standard Windows applications.

5 Examples' list

In the package there is a folder contain some already prepared examples. All of them are taken from the relevant literature. In particular:

- `biaz_gil.ckc` from V. Biazzo, A. Gilio: A generalization of the fundamental theorem of de Finetti for imprecise conditional probability assessments. *International Journal of Approximate Reasoning*, 24, (2000), pp. 251–272.

- `cgv_SofCom.ckc` from A. Capotorti, L. Galli and B. Vantaggi: How to use locally strong coherence in an inferential process based on upper-lower probabilities. *Soft Computing*, 7(5), (2003), pp.280–287.

- `colscozz_infsci.ckc` and `colscozz_infsci_up2.ckc` from G. Coletti and R. Scozzafava: The role of coherence in eliciting and handling imprecise probability and its application to medical diagnosis. *Information Sciences*, 130 (2000) 41–65.

- `cv_AMAI.ckc` from A. Capotorti, B. Vantaggi: Locally strong coherence in an inference process. *Annals of Mathematics and Artificial Intelligence*, 35(1-4) (2002), 125–149.

- `cv_ipmu2000.ckc` from A. Capotorti, B. Vantaggi: A computational strategy for inference by strong local coherence. *Proc. of IPMU2000, Madrid, Spain*, (2000).

- `dickey.ckc` from J. Dickey: Convenient Interactive Computing for Coherent Imprecise Prevision Assessments, *proc. of the 3rd International Symposium on Imprecise Probabilities and Their Applications ISIPTA'03*, J-M Bernard, T Seidenfeld & M Zaffalon Eds., Lugano, Switzerland (2003) pp.218–230.

- `gist.ckc` from A. Capotorti, S. Fagundes Leite: Reliability of GIST diagnosis based on partial information. in *Applied Bayesian Statistical Studies In Biology And Medicine*, (M. Di Bacco, G. D'Amore, F. Scalfari editors), Kluwer Academic Publishers, Boston/Dordrecht/London, (2003).

- `isipta01.ckc` from A. Capotorti , B. Vantaggi: A simplified algorithm for inference by lower conditional probabilities, *proc. of the 2nd International Symposium on Imprecise Probabilities and Their Applications ISIPTA'01*, Shaker Publishing 2000, The Netherlands, (2001) pp.68–76.

- `Mathware_coer.ckc`, `Mathware_ex4.ckc` and `Mathware_ex5.ckc` from G. Coletti: Coherence principles for handling qualitative and quantitative partial probabilistic assessments. *Mathware & Soft Computing*, 3 (1996), pp. 159–172.

References

[1] A. Capotorti, L. Galli and B. Vantaggi. How to use locally strong coherence in an inferential process based on upper-lower probabilities. *Soft Computing*, 7(5), (2003), pp.280–287.

[2] G. Coletti, R. Scozzafava. *Probabilistic Logic in a Coherent Setting*, Dordrecht: Kluwer, Series "Trends in Logic", (2002).