The following are examples of conjectures:

- symmetric and \( k \) connected and alpha0 \( \leq b \) and \( k \geq 2^{b*(b+2)} \) \( \implies \) hamiltonian.

where \( 2^b \) means \( 2^{b^b} \)

- I connected and meyniel \( 2^n \) nodes-3 \( \implies \) hamiltonian.

After having formulated a conjecture \( H \implies T \), one can try to prove or refute it. In proving a conjecture, advantage is taken of known relations between invariants and theorems in graph theory. In order to refute a conjecture, the negation of the conjecture \( \neg (H \implies T) \) equivalently \( H \text{ and } \neg T \), is to be proved, what amounts to provide a counterexample to the conjecture, i.e. a digraph which belongs to the digraph family \( H \) and \( \neg T \). By failing in refuting or proving it, the conjecture is declared open. To leave a conjecture open means that the knowledge at hand is not sufficient to prove or reject it. Moreover, the process used to study a conjecture may conduce to the formulation of new conjectures, enriching the existing knowledge. The researcher can also produce conjectures while drawing digraph families with similar behaviour. One of the graphist's goal is "to send the good conjectures (not trivially true or false) to the market", in the sense that they are made public to be studied by researchers in the field. Tests must be made with existing digraphs, in order to select the "good" conjectures, for preserving the graphist's prestige. The graphist reasoning is fascinating, but at the same time it may easily conduce to a long and sterile work, if tools and methods are not available. We will call Graph Conjecture Processing (GCP) such graph-theoretic practice.

Several efforts have been made in building computational systems to aid researchers in GCP. From our perspective the following systems deserve to be mentioned: the INGRID software system [2,4] has been designed in order to bound all invariants once the ranges of selected invariants are specified. The system relies upon a collection of relations between invariants involving 36 graph invariants and a set of graphs. The GRAFFITI system [6] generates a conjecture on a negative way, i.e. by verifying that the system can not afford a graph as a counterexample to the conjecture and deciding after that whether the conjecture is or not interesting for being the object of further research. Later work on GRAFFITI [7] has been motivated by designing more powerful heuristics for telling whether the conjectures made by the system are of interest.

In this paper, we describe the AGORA tool, basically constituted by a constraint-based terminological reasoning subsystem for solving digraph conjectures, a knowledge base specialized on paths and circuits and a subsumption algorithm for consistency, proving and refutation. A first prototype of this system, based on NLOG[17] a full first-order logic programming languages that can deal with non-Horn clauses and negative clauses, was designed for C++ [19] and Quintus Prolog\(^2\). Due to the poor performance of this first prototype, a stand alone version for C++/SICStus Prolog\(^3\) [9] was written by the Artificial Intelligence Group of the La Habana University, Cuba. The user-interface for this version was implemented using Tcl/Tk [18] under a Windows95/LINUX platform. In this paper, the last AGORA version for workstations under a UNIX/MWWToolkit++/C++/SICStus Prolog platform [15] will be presented. This version offers an extended inference power for solving more complex conjectures and new reasoning strategies applied by the graphists when proving conjectures. Explanations of the evaluation's results are also given as an additional feature. Moreover, the AGORA architecture and graphical user-interface (GUI) will be described. AGORA can be used as a stand alone system and it is now employed for studying the construction of interconnection networks, which are modeled by digraphs. It can also be used as the integrated reasoning system facility of the GREAt (Graph Researcher Assistant) environment [16], developed at the Software Engineering Center (ISYS) of the Universidad Central de Venezuela.

This paper is structured as follows: besides this introduction: Section 2 describing the AGORA global architecture, Section 3 presenting the reasoning system and illustrating it with several examples and Section 4 devoted to the description of the object-oriented GUI. Finally, the conclusion concerns the AGORA limitations and the future research.

\(^2\)Quintus Prolog is a registered trademark of Quintus Corporation, An Intergraph Company, Palo Alto, CA

\(^3\)Copyright 1995 SICS, Swedish Institute of Computer Science, Sweden

\(^4\)MotifWidgetWrapperToolkit, developed at ISYS research center, Universidad Central de Venezuela
2. The Architecture

The AGORA architecture is shown in Figure 1. The system is constituted by two main components, the AGORA Interface System, which will be denoted by AUI and the AGORA Reasoning System, denoted by ARS. AUI captures the user inputs and adequately deliver them to ARS. AUI is constituted by four main modules: - Conjecture Editor, - Conjecture Browser, - Knowledge Base Browser and - Help System. The interface layer between both systems is constituted by two temporary text files, one for storing the conjecture in Prolog form, and the other for storing the results of the ARS evaluation.

![Figure 1. Architecture of the AGORA tool](image)

The Conjecture Editor is responsible for the AGORA main functionalities. Its graphical interface allows the user to enter a conjecture, expressed in LEC (Conjecture Specification Language) [13]; this expression is submitted to a semantic and syntactic analysis before being translated into an internal form, which is understood and evaluated by ARS. For example the LEC expression:

\[
\text{woodall} \geq \text{nodes} \implies \text{hamiltonian}.
\]

is a conjecture written in LEC. A text file, produced as a result of the translation of the conjecture in a PROLOG internal form, is read by ARS to evaluate the conjecture, under the responsibility of the Deductive Mechanism. The results of the evaluation are stored by the Explanation module in another text file. This file is used by the Conjecture Editor for showing these results to the user. Both temporary files maintain the independence of the reasoning system from the graphical user interface.

The Conjecture Browser is used to store and show the persistent information on conjectures and the results of the evaluation. Information such as date of evaluation, results and explanation, is registered for each conjecture. The browser can handle Proved, Refuted and Failed Conjectures through a powerful graphical interface.

The Knowledge Base Browser allows to edit the AGORA Knowledge Base (AKB), located in ARS. Through this browser, specialized information on theorems, invariant relations and digraphs can be stored or retrieved. Records are used to represent AKB data. New AKB records can be added and existing ones can be modified or deleted. An AKB record contains the following main fields: identifier, references, comments, text. The