# Proposal of PhD Thesis

## Using Concurrency for First-Order Analytic Tableaux

## Context and Position

Research on parallel satisfiability solving, meaning satisfiability in propositional logic (SAT), began in the early 1990's and is still actively pursued today [7]. Research on parallel theorem proving, meaning automatic theorem proving (ATP) in first-order logic, began in the mid and late 1980's, flourished in the 1990's, and came pretty much to a halt in the early 2000's [10]. Concerning first-order logic, most of the work concerns model-based methods, where a theorem-proving method is model-based, if the state of a derivation contains a representation of a candidate partial model that unfolds with the derivation. For the tableau method for example, there are model-elimination tableaux, which are based on clausal tableaux.

Regarding the tableau method [6], as initially designed by Beth and Hintikka [1, 3], which manipulates first-order formulas (and not clauses), no work appears in the literature as far as the authors know. This kind of tableaux is interesting because it does not transform the initial formula (no Skolemisation, no clausification), and it produces a genuine proof at the end in the sequent calculus without cut. In addition, this method could naturally and advantageously extend to concurrency. Indeed, the tableau method is based on a tree-like proof search in which all branches are to be closed (i.e. we look for a contradiction in each branch). Thus, we could imagine trying to close all the branches in parallel. Nevertheless, the problem is more complex than it seems because in first-order logic, there are dependencies between branches. We look for a substitution of metavariables (also called free variables in the literature, and which are introduced by existential quantifiers for which we look for an instantiation) such that all branches are closed simultaneously.

## Objectives

In this PhD thesis, we would like to focus on concurrency for analytic tableaux, which deals with first-order formulas. As we are interested in proof search, we will consider tableaux with metavariables, which allow us to find (by unification) instances for existential formulas. As said previously, this will consist in closing all the branches of a proof search tree in parallel while taking into account the dependencies introduced by the metavariables, and we will present, in the following, the different advantages and problems of such a proof search.

Exploring all branches of a proof search tree in parallel has several advantages. One of them is that some branches can provide possible instances for metavariables while others will not. A sequential exploration of the branches must choose a branch to be explored and if the latter does not provide instances for metavariables, a long search may be done in this branch before the conditions of fairness require us to change the branch to be explored. This is the case with the formula: $\exists x.x = a \wedge \mathcal{F}(x)$, where $a$ is a constant and $\mathcal{F}(x)$ any formula. If we start exploring the branch corresponding to $\mathcal{F}(x)$, we may not find a proof, while the first branch immediately gives us the instance $a$ for $x$. A parallel exploration of the branches would allow us to find the instance $a$ for $x$ quickly. With such a parallel exploration, there is the problem of knowing if we instantiate immediately as soon as we find an instance and how we avoid redoing work already done in the proof search.

Another advantage of parallel branch exploration is that it will be possible to better deal with purely classical theorems, which require several metavariables for an existential variable in order to be able to find the proof. For example, if we consider the formula $\exists x.P(x) \Rightarrow P(a) \wedge P(b)$, where $a$ and $b$ are constants and $P$ a predicate, two instantiations are required (one with $a$ and one with with $b$). In the case of destructive management of metavariables, we will first try to find the proof with one metavariable, which will produce a failure. We will then try with two metavariables and find the proof. A parallel exploration of the branches could show us very quickly that two separate instances are suitable for $x$, and we could see right away that two metavariables are necessary without waiting for a failure of the proof search. It will be necessary to look carefully at how this method is combined with the one mentioned previously, which restarts the proof search as soon as an instance has been found in a branch.

Currently, there are several parallel programming approaches. These approaches can be classified according to targeted hardware infrastructures (taxonomy of Flynn [2]). We are in particular interested in MIMD (Multiple Instruction, Multiple Data) architectures, including multi-core processors and distributed systems, with a shared or a distributed memory space, and SIMD (Single Instruction, Multiple Data) computers, like GPUs. In each classification, many programming models are also proposed, with different levels of abstraction. A model with a low level of abstraction, like multithreading, usually combines functional aspects with non-functional ones (parallelism management, synchronization, data sharing/transfer, scheduling, etc.), while a high level of abstraction, like workflow programming [5, 12, 11], allows a developer to focus on functional aspects and delegate the management of other aspects to a middle-ware.

A main objective of this PhD thesis is to propose a strategy and a programming model that are suitable and efficient for solving problems with the tableau method, while enabling execution traceability. For this, it will be necessary to well understand the tableau method to determine in particular how data are used, how they can be shared between branches to explore, how they can be exploited for result traceability, and which parallel structures should be used. A meticulous study of existing parallel programming approaches is also necessary to identify those close to meet a suitable level of abstraction, ideally an infrastructure independent model enabling execution on multi-core processors as well as on distributed systems. The aim is to ensure scalability, portability, and extensibility. At first glance, starting from workflow based models could meet our aim.

Finally, it will be necessary to ensure that the parallel strategy of exploration of proof search space remains sound and complete. To do so, it will be necessary to use a formalism able to deal with processes (a process is created for each new branch) and communication between processes (for example, a process that has found an instance for a metavariable must report it to the other processes to possibly interrupt their proof search). Many process calculi have existed since the 1980s, such as Milner's CCS [8] (or its extension with the $\pi$-calculus [9]) or Hoare's CSP [4]. Note that we will use these formalisms as frameworks to prove the soundness and completeness of our method, and in particular, we will not be interested in using implementations of these formalisms.

## Additional Remarks

The PhD thesis will be co-supervised by:

- David Delahaye (LIRMM, Univ. de Montpellier, CNRS, `David.Delahaye@lirmm.fr`);
- Hinde Bouziane (LIRMM, Univ. de Montpellier, CNRS, `Hinde.Bouziane@lirmm.fr`).

## References

[1] E. W. Beth. *Formal Methods: An Introduction to Symbolic Logic and to the Study of Effective Operations in Arithmetic and Logic*, volume 4 of *Synthese Library*. D. Reidel Pub. Co., 1962.

[2] M. J. Flynn. Some Computer Organizations and Their Effectiveness. *IEEE Transactions Computers*, 21(9):948–960, Sept. 1972.

[3] J. Hintikka. Two Papers on Symbolic Logic: Form and Content in Quantification Theory and Reductions in the Theory of Types. *Societas Philosophica, Acta philosophica Fennica*, 8:7–55, 1955.

[4] C. A. R. Hoare. Communicating Sequential Processes. *Communications of the ACM (CACM)*, 21(8):666–677, Aug. 1978.

[5] D. Hollingsworth. Workflow Management Coalition – The Workflow Reference Model. Technical report, Workflow Management Coalition, Jan. 1995.

[6] R. Letz. First-Order Tableau Methods. In M. D'Agostino, D. M. Gabbay, R. Hähnle, and J. Posegga, editors, *Handbook of Tableau Methods*, pages 125–196. Springer, 1999. ISBN 978-94-017-1754-0.

[7] R. Martins, V. M. Manquinho, and I. Lynce. An Overview of Parallel SAT Solving. *Constraints*, 17(3):304–347, July 2012.

[8] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science (LNCS)*. Springer, 1980. ISBN 3-540-10235-3.

[9] R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, I/II. *Information and Computation*, 100(1):1–77, 1992.

[10] J. Schumann. Parallel Theorem Provers – An Overview. In *Parallelization in Inference Systems*, volume 590 of *Lecture Notes in Computer Science (LNCS)*, pages 26–50, Dagstuhl Castle (Germany), 1992. Springer.

[11] I. Taylor, E. Deelman, D. Gannon, and M. Shields, editors. *Workflows for e-Science: Scientific Workflows for Grids*. Springer, 2007.

[12] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(3):5–51, July 2003.