LE MATEMATICHE
Vol. LXIII (2008) – Fasc. I, pp. 85–105

USING ÆTNANOVA TO FORMALLY PROVE THAT THE DAVIS-PUTNAM SATISFIABILITY TEST IS CORRECT

EUGENIO G. OMODEO - ALEXANDRU I. TOMESCU

This paper reports on using the ÆtnaNova/Referee proof-verification system to formalize issues regarding the satisfiability of CNF-formulae of propositional logic. We specify an "archetype" version of the Davis-Putnam-Logemann-Loveland algorithm through the THEORY of recursive functions based on a well-founded relation, and prove it to be correct. Within the same framework, and by resorting to the Zorn lemma, we develop a straightforward proof of the compactness theorem.

1. Introduction

ÆtnaNova, aka Referee (see [2, 11, 12]), or Ref for short, is a broad gauge proof-verification system based on set theory; its design and implementation evolved hand-in-hand with the development of a fully formalized proof scenario which will culminate in a proof of the Cauchy integral theorem on analytic functions [5, 7]. One virtue of Ref is that the syntax of proofs is very close to natural mathematical notation: this ensures readable and reusable proofware.[1]

Entrato in redazione 27 novembre 2007

AMS 2000 Subject Classification: 03E75,03F07,03-02,03C62,03E30

Keywords: Proof checking, program-correctness verification, set theory, computable set theory, cumulative hierarchy, satisfiability decision procedures, proof modularization.

Research partially funded by the INTAS project *Algebraic and deduction methods in non-classical logic and their applications to Computer Science*, and by the Italian MUR/PRIN project *Large-scale development of certified mathematical proofs* n. 2006012773.

[1] *Proofware* [8] is the peculiar scripting code used to specify in absolute rigor proofs and proof schemes. Its primary application is not to describe algorithms.

Formal methods of algorithm verification are a natural yield of proof checking and automated deduction. Algorithms have often been examined formally with systems such as Isabelle/HOL [6, 10] or Coq [9], but the experience on which we will report is the first verification of an algorithm carried out with Ref. Although Ref does not, up until today, encompass specific programming notation, this paper suggests that an integration of such notation with Ref's logical notation is easy to conceive.

We will formalize the notion of model for a formula in conjunctive normal form, and will introduce an "archetype" version of the Davis-Putnam-Logemann-Loveland procedure, DPLL. To encapsulate the relevant concepts in the Ref system, we will use the THEORY mechanism [12]. Like procedures in a programming language, theories have lists of formal parameters. Each theory requires its parameters to meet a set of assumptions. When applied to a list of actual parameters that have been shown to meet the assumptions, a theory will instantiate several additional output set, predicate, and function symbols, and then supply a list of theorems initially proved explicitly (relative to the formal parameters) by the user inside the theory itself. These theorems will generally involve the new symbols. Roughly speaking, the assumptions met by the formal parameters can be regarded as preconditions, while the theorems supplied in the "theory" are postconditions. By the term *interface* we refer to the list of formal parameters of a THEORY, the assumptions they meet, and the statements of the theorems proved within the THEORY.

M. Davis and H. Putnam [4], and later M. Davis, G. Logemann, and D. Loveland [3], proposed an algorithm for determining whether a propositional formula given in conjunctive normal form is satisfiable or not. A vast literature has evolved from this seminal paper [4] — before it, researchers in the Automated Deduction field thought they had to reduce ground sentences to equivalent *dis*junctive normal form to be tested for validity [14], whereas nowadays reduction to conjunctive normal form and satisfiability test have become rather standard steps (see, e.g. [1]).

The version of DPLL underlying our work is best specified in rather conventional imperative terms, like those seen in Figure 1 where the programming language used is SETL [15]. But, in order to comply with the Ref notation, we must formalize DPLL in purely logical, set-theoretical terms, as will be shown and explained at due time.

This paper is organized as follows. In Section 2 we discuss how to create, out of a given set of so-called "atoms", a set of affirmative and negative *literals* (i.e., sets which can act conveniently as pure symbols) of which the affirmative ones are equi-numerous with the atoms. Section 3 introduces *CNF-formulae* along with the pertaining notions of *model* and *satisfiability*. It also states lem-

mas on how to *reduce* a CNF formula S, relative to a literal X occurring within it: this reduction either produces a single CNF S', simpler but equi-satisfiable with S, or it produces two simpler CNFs, S' and S'', such that S is satisfiable if and only if either S' or S'' is satisfiable. In Section 4, the algorithm *DPLL* is specified and shown to be correct. The specification is provided in such terms as to ensure termination, but grossly, in the sense that no criterion is indicated for selecting the literal X relative to which the CNF-formula S will be reduced (such a criterion, in fact, would have a bearing on efficiency but is immaterial as far as correctness is concerned). The variant of DPLL which discards, at the outset, all tautological clauses from the input formula is also shown to be correct. In Section 5 we briefly report on how the compactness theorem about propositional logic was proved with Ref.

The issues discussed in Sections 2, 3, 4, and 5 reflect into three Ref theories. As these consist of 1161 lines of code altogether, we cannot afford discussing in full the various constructions and proofs which support our mathematical discourse: only the interfaces of the theories will be shown, in a dedicated Appendix. Notwithstanding, we will provide examples and stress sensible points of the proof of correctness.

The interested reader should turn to [11], [12] and [16] for syntax, basic terminology and features of Ref.

2. Representing literals via a global "toggle switch"

The signedSymbols theory, whose interface is shown in Appendix A, receives via its formal parameter a set atms not subject to any assumption. Out of this set, it creates a set lits_Θ whose elements shall be regarded as *literals*.[2] Internally, this theory defines an injective affirmation operation, e.g.

$$\mathsf{aff}_\Theta(\mathsf{X}) =_{\mathrm{Def}} \{\{\mathsf{X}\}\},$$

converting every element x of atms into a positive literal. A negation operation, e.g.

$$\mathsf{neg}_\Theta(\mathsf{X}) =_{\mathrm{Def}} \big(\mathsf{X} \setminus \{\emptyset\}\big) \cup \big(\{\emptyset\} \setminus \mathsf{X}\big),$$

is also defined internally. Thus, if we regard each $\mathsf{neg}_\Theta(\mathsf{aff}_\Theta(\mathsf{x}))$ as a negative literal and stick to the above definitions, the collection lits_Θ of all positive and negative literals constructed inside the signedSymbols theory turns out to be

$$\mathsf{lits}_\Theta =_{\mathrm{Def}} \big\{b \cup \{\{x\}\} : b \subseteq \{\emptyset\}, x \in \mathsf{atms}\big\}.$$

[2] The Θ subscript is attached to formal output parameters of a THEORY, to be actualized whenever the THEORY will be invoked (by means of the construct named APPLY, cf. [12]).

Figure 1: A procedural specification of the "archetype" DPLL algorithm

```
procedure dp0(s)
   -- s is a finite (conjunctive) set of disjunctive clauses of literals

   t := {c: c in s, h in c | neg(h) in c};
   -- tautological clauses within input conjunction
   m := dp1(s - t);
   -- analyze the rest; if unsatisfiable, propagate {fail},
   -- else enlarge its model
   return
      if fail in m then {fail}
      else m + {arb({h,neg(h)}): c in s, h in (c - m) | neg(h) in (c - m)} end if;

   procedure dp1(s);
      -- s is a finite set of disjunctive non-tautological clauses
      -- the model of s under construction will be
      -- enlarged repeatedly, and s will be simplified
      -- until s is either satisfied or blatantly false

      if s = {} then return {} end if; -- obvious
      if {} in s then return {fail}; end if; -- absurd

      h := selectLiteral(s);
      -- if the selected literal appears in a singleton clause,
      -- apply the unit literal rule

      if {h} in s then
         -- the literal h was chosen within a unit clause
         return dp1({c - {neg(h)}: c in s | h notin c}) + {h};
      end if;

      -- otherwise, proceed either to pure literal rule
      -- or to splitting rule

      if (FORALL d in s | neg(h) notin d) then
         -- include the pure literal h in the model
         return dp1({c in s | h notin c}) + {h};
      end if;

      -- split
      m1 := dp1({c - {neg(h)}: c in s | h notin c}) + {h};
      return
         if fail notin m1 then m1
         else dp1({c - {h}: c in s | neg(h) notin c}) + {neg(h)} end if;
   end dp1;

   procedure selectLiteral(s);
      -- Here we are assuming that s is a non-null set of non-null clauses
      -- we pick a literal in one of the clauses of s, and return it
      return arb(arb(s)); -- refined selection criteria can enhance efficiency
   end selectLiteral;

end dp0;
```

$$\{\mathsf{rk}(\mathsf{x}) : \mathsf{x} \in \mathsf{lits}_\Theta\} \subseteq \{\mathsf{rk}(\mathsf{false}_\Theta)\}$$
$$\langle \forall \mathsf{n} \in \mathbb{N} \cup \{\mathbb{N}\} \mid \mathsf{n} \in \mathsf{rk}(\mathsf{false}_\Theta)\rangle$$
$$\langle \forall \mathsf{x} \mid \mathsf{rk}(\mathsf{x}) \notin \mathsf{rk}(\mathsf{false}_\Theta) \rightarrow \mathsf{rk}\big(\mathsf{neg}_\Theta(\mathsf{x})\big) = \mathsf{rk}(\mathsf{x})\rangle$$

Figure 2: Addendum to the interface of the signedSymbols theory

Along with $\mathsf{aff}_\Theta(\_)$, $\mathsf{neg}_\Theta(\_)$, and lits_Θ, the signedSymbols theory returns a designation for falsehood, e.g.

$$\mathsf{false}_\Theta =_{\text{Def}} \emptyset,$$

such that the pair false_Θ, $\mathsf{neg}_\Theta(\mathsf{false}_\Theta)$ of complementary *truth values* does not intersect lits_Θ. Note that the theorems externalized by this theory include

$$\langle \forall x \mid \mathsf{neg}_\Theta\big(\mathsf{neg}_\Theta(x)\big) = x \;\&\; \mathsf{neg}_\Theta(x) \neq x\rangle,$$

stating that the global function neg_Θ is a GALOIS CORRESPONDENCE.

We remark in passing that the above-shown precise definitions of $\mathsf{aff}_\Theta(\_)$, $\mathsf{neg}_\Theta(\_)$, lits_Θ, and false_Θ, are devoid of interest outside the theory we are considering and, as such, do not deserve appearing explicitly on the theory interface. Speaking in general, a theory should be designed in such a way that reworking of its internals does not propagate outside the theory in question, to other theories that make use of it.

Actually, a second-release implementation of the signedSymbols theory—motivated by the authors' desire to make signedSymbols more widely usable, e.g. for the development of a theory of freely generated groups—insists on the fact that all of the syntactic entities in the collection

$$\{x : x \in \mathsf{lits}_\Theta\} \cup \{\mathsf{false}_\Theta, \mathsf{neg}_\Theta(\mathsf{false}_\Theta)\}$$

have the same, transfinite, set-theoretic rank (see Figure 2).

It may be worth recalling here that the global rank function (intuitively speaking, a measure of how deeply nested every set is) has the recursive definition

$$\mathsf{rk}(\mathsf{X}) =_{\text{Def}} \bigcup \big\{\mathsf{rk}(y) \cup \{\mathsf{rk}(y)\} : y \in \mathsf{X}\big\}.$$

The new constraint about rank (entailing, among others, that false_Θ can no longer equal $\emptyset$), as well as the modest changes needed inside the theory to meet this constraint, have no bearing whatsoever on the exploitation of signedSymbols discussed in the ongoing.

3. Modeling propositional CNF-formulae

Any CNF-formula

$$\mathsf{S} = \mathsf{c}_1 \wedge \mathsf{c}_2 \wedge \cdots \wedge \mathsf{c}_n$$

of propositional calculus, where each clause c_i is a disjunction

$$\mathsf{c}_i = \mathsf{h}_{i,1} \vee \mathsf{h}_{i,2} \vee \cdots \vee \mathsf{h}_{i,\#\mathsf{c}_i} \qquad (\#c \text{ being the cardinality of } c)$$

of distinct literals, is conveniently represented as the set

$$\mathsf{S} = \big\{\{\mathsf{h}_{1,1}, \mathsf{h}_{1,2}, \ldots, \mathsf{h}_{1,\#\mathsf{c}_1}\}, \ldots, \{\mathsf{h}_{n,1}, \mathsf{h}_{n,2}, \ldots, \mathsf{h}_{n,\#\mathsf{c}_n}\}\big\}.$$

In what follows, as we think of a "CNF-formula" in these set-theoretical terms, we shall say that the set M is a MODEL of S if and only if:

i) M is not contradictory (i.e., M contains no complementary literals h, $\backsim\mathsf{h}$),

ii) every set in S shares at least one literal with M.

We say that a formula S is SATISFIABLE iff there exists a model of S.

The cnfModels theory (see Appendix B) receives a global Galois correspondence $h \mapsto \backsim h$, and introduces the notions of model and satisfiability of a CNF-formula as just explained:

$$\mathsf{Is\_cnfModel}_\Theta(\mathsf{M}) \leftrightarrow_{\text{Def}} \{h \in \mathsf{M} \mid \backsim h \in \mathsf{M}\} = \emptyset;$$

$$\mathsf{Has\_cnfModel}_\Theta(\mathsf{S},\mathsf{M}) \leftrightarrow_{\text{Def}} \mathsf{Is\_cnfModel}_\Theta(\mathsf{M}) \;\&\; \{c \in \mathsf{S} \mid \mathsf{M} \cap c = \emptyset\} = \emptyset;$$

$$\mathsf{Is\_cnfSat}_\Theta(\mathsf{S}) \leftrightarrow_{\text{Def}} \langle \exists m \mid \mathsf{Has\_cnfModel}_\Theta(\mathsf{S}, m)\rangle.$$

The main purpose of this theory is to prove various lemmas enabling the reduction of an instance of the satifiability/modeling problem for formulae in conjunctive normal form to a simpler instance of the same problem.

Relative to a CNF-formula S, when c is a clause in S and h is a literal in c, we shall say that

- c is a TAUTOLOGICAL CLAUSE iff

$$\langle \exists h' \in \mathsf{c} \mid \backsim h' \in \mathsf{c}\rangle \qquad \text{(tautological clause)}$$

- c is a UNIT CLAUSE and h is a UNIT LITERAL iff

$$\mathsf{c} = \{\mathsf{h}\} \qquad \text{(unit clause, unit literal)}$$

- h is a PURE LITERAL iff

$$\langle \forall c' \in \mathsf{S} \mid \backsim \mathsf{h} \notin c'\rangle \qquad \text{(pure literal)}$$

Let us now examine the algorithm in Figure 1. The wrapper procedure $\mathsf{dp0}(s)$ receives in input a CNF-formula s represented as mentioned at the beginning of this section. It will return a model for s, if any exists, and will return the unsatisfiability indication $\{fail\}$ otherwise. It first builds the set t of all tautological clauses in s and calls the recursive function dp1 on the input formula deprived of these tautologies, in order to get a model m for it. If this simpler formula is unsatisfiable, the same holds for the original s; otherwise, m gets extended into a model for s by addition, for each clause in t, of one of the complementary literals which render it tautological, unless either already belongs to m.

The backtracking function $\mathsf{dp1}(s)$ begins by picking a literal h in s; then it judges whether to put h in the model of s. If not, it tentatively adds first h and then—if needed— $\backsim h$ to the model, simplifying s accordingly, and recursively checking whether either simplified formula has a model.

More precisely: if the literal h was taken from a unit clause (and hence must belong to any model of s), or is a pure literal (and hence its insertion into the model is unproblematic), then h simply goes into the model. Otherwise, s gets simplified in two ways, reflecting the possible truth-value assignments for h:

$$\{\{\mathsf{h}\}\} \cup \{\mathsf{c} \setminus \{\backsim\mathsf{h}\} : \mathsf{c} \in \mathsf{s} \,|\, \mathsf{h} \notin \mathsf{c}\}\,, \quad \{\{\backsim\mathsf{h}\}\} \cup \{\mathsf{c} \setminus \{\mathsf{h}\} : \mathsf{c} \in \mathsf{s} \,|\, \backsim\mathsf{h} \notin \mathsf{c}\}\,.$$

If either of these is unsatisfiable, an empty disjunctive clause $\{\}$ will eventually appear during its recursive treatment, and will serve as a base case to close, with a failure, a branch of the recursion. On the other hand, if the parameter s becomes $\{\}$ along any branch of the recursion, then it is satisfiable (because it has the model $\{\}$); this will indicate that the s given at the outset is satisfiable as well.

To support the claims which we will provide as explanations during the analysis that follows, we will produce the statements of various theorems whose proofs have been written and verified with Ref; note that variables written in upper case within such statements are, by convention, universally quantified. To be short, we denote by $\bigcup\mathsf{S}$ the union-set $\{x : y \in \mathsf{S}, x \in y\}$, and by $\mathbf{arb}(\mathsf{X})$, where X is any set, an arbitrary member of X (conventionally, $\mathbf{arb}(\emptyset) = \emptyset$).[3] Relative to S, c, and h as above, we must consider the following cases:

(i) Both h and $\backsim$h belong to c, and hence c is a tautological clause. In this case, if M is a model for $\mathsf{S} \setminus \{\mathsf{c}\}$ then if M contains h or $\backsim$ h, it is a model for S as well. If it does not contain h or $\backsim$ h, then adding either of h, $\backsim$ h will make

[3]Technically, **arb**(_) originates from Skolemization of the sentence

$$\left\langle \forall x \exists y \forall v \,|\, \left(v \in x \rightarrow (v \notin y \;\&\; y \in x)\right) \;\&\; \left(v \in y \rightarrow y \in x\right)\right\rangle,$$

which is a version of the *regularity* axiom of the Zermelo-Fraenkel set theory.

M into a model for S. To state this, one can introduce the following theorem:

THEOREM. $\mathsf{Has\_cnfModel}_\Theta(S,M) \to \mathsf{Has\_cnfModel}_\Theta(S \cup \{c : c \in S_0, h \in c \mid \backsim h \in c\}, M \cup \{\mathbf{arb}(\{h, \backsim h\}) : c \in S_0, h \in c \backslash M \mid \backsim h \in c \backslash M\})$.

(ii) h is a unit literal. Using the definition of $\mathsf{Has\_cnfModel}_\Theta$, we have that h belongs to any model M of S. Therefore, any model of the simplified formula $\{\{h\}\} \cup \{c \setminus \{\backsim h\} : c \in s \mid h \notin c\}$ will also be a model of S.

THEOREM. $\{H\} \in S$ & $\mathsf{Has\_cnfModel}_\Theta(S,M) \to H \in M$ & $\backsim H \notin M$;
THEOREM. $\mathsf{Has\_cnfModel}_\Theta(\{\{H\}\} \cup \{c \backslash \{\backsim H\} : c \in S \mid H \notin c\}, M) \to \mathsf{Has\_cnfModel}_\Theta(S,M)$.

(iii) h is a pure literal. By the theorem below, establishing the satisfiability of S amounts to establishing the satisfiability of the reduced formula $\{\{h\}\} \cup \{c \in S \mid h \notin c\}$. This formula, stripped of all clauses containing h, requires h to belong to its model, as discussed above.

THEOREM. $\backsim H \notin \bigcup S \to \big(\mathsf{Is\_cnfSat}_\Theta(S) \leftrightarrow \mathsf{Is\_cnfSat}_\Theta(\{\{H\}\} \cup \{c \in S \mid H \notin c\})\big)$.

(iv) h is neither a unit literal nor a pure literal. Then $\backsim h \in \bigcup S$. Splitting the formula S on h means checking whether either of the two formulae

$$\{\{h\}\} \cup \{c \backslash \{\backsim h\} : c \in S \mid h \notin c\}, \quad \{\{\backsim h\}\} \cup \{c \backslash \{h\} : c \in S \mid \backsim h \notin c\}$$

has a model. As h or $\backsim h$ are unit literals here, this amounts to checking whether h or $\backsim h$ can be put in a model of S.

THEOREM. $\mathsf{Is\_cnfSat}_\Theta(S) \leftrightarrow \mathsf{Is\_cnfSat}_\Theta(\{\{H\}\} \cup \{c \backslash \{\backsim H\} : c \in S \mid H \notin c\}) \vee \mathsf{Is\_cnfSat}_\Theta(\{\{\backsim H\}\} \cup \{c \backslash \{H\} : c \in S \mid \backsim H \notin c\})$.

To exemplify the Ref proof-checking mechanism and its inference steps, we illustrate below how to prove the above splitting principle by resorting to the following theorems, which must have been proved already:

THEOREM $\mathsf{cnfModels}_0$. $\backsim\backsim H = H$ & $\backsim H \neq H$;
THEOREM $\mathsf{cnfModels}_{12}$. $\mathsf{Has\_cnfModel}_\Theta(S,M) \to \mathsf{Has\_cnfModel}_\Theta(\{\{H\}\} \cup \{c \backslash \{\backsim H\} : c \in S \mid H \notin c\}, M \cup \{H\}) \vee \mathsf{Has\_cnfModel}_\Theta(\{\{\backsim H\}\} \cup \{c \backslash \{H\} : c \in S \mid \backsim H \notin c\}, M \cup \{\backsim H\})$;
THEOREM $\mathsf{cnfModels}_{14}$. $\mathsf{Is\_cnfSat}_\Theta(\{\{H\}\} \cup \{c \backslash \{\backsim H\} : c \in S \mid H \notin c\}) \to$

$\mathsf{Is\_cnfSat}_\Theta(S)$.

The proof of the splitting principle runs as follows:

THEOREM $\mathsf{cnfModels}_{15}$. $\mathsf{Is\_cnfSat}_\Theta(S) \leftrightarrow$
$\mathsf{Is\_cnfSat}_\Theta(\{\{H\}\} \cup \{c \setminus \{\backsim H\} : c \in S \,|\, H \notin c\}) \vee$
$\mathsf{Is\_cnfSat}_\Theta(\{\{\backsim H\}\} \cup \{c \setminus \{H\} : c \in S \,|\, \backsim H \notin c\})$. PROOF:
$\mathsf{Suppose\_not}(s, h_0) \Rightarrow$ AUTO

Let s, h_0 be a counterexample. We consider the direct implication first:

$\mathsf{Suppose} \Rightarrow \mathsf{Is\_cnfSat}_\Theta(s)$ &
$\neg\big(\mathsf{Is\_cnfSat}_\Theta(\{\{h_0\}\} \cup \{c \setminus \{\backsim h_0\} : c \in s \,|\, h_0 \notin c\}) \vee$
$\mathsf{Is\_cnfSat}_\Theta(\{\{\backsim h_0\}\} \cup \{c \setminus \{h_0\} : c \in s \,|\, \backsim h_0 \notin c\})\big)$
$\mathsf{Use\_def}(\mathsf{Is\_cnfSat}_\Theta) \Rightarrow$ *Stat1* : $\big\langle \exists m \,|\, \mathsf{Has\_cnfModel}_\Theta(s, m)\big\rangle$ &
Stat2 : $\neg\big\langle \exists m \,|\, \mathsf{Has\_cnfModel}_\Theta(\{\{h_0\}\} \cup \{c \setminus \{\backsim h_0\} : c \in s \,|\, h_0 \notin c\}, m)\big\rangle$ &
Stat3 : $\neg\big\langle \exists m \,|\, \mathsf{Has\_cnfModel}_\Theta(\{\{\backsim h_0\}\} \cup \{c \setminus \{h_0\} : c \in s \,|\, \backsim h_0 \notin c\}, m)\big\rangle$
$\big\langle m_1\big\rangle \hookrightarrow$*Stat1*$(*Stat1*\star) \Rightarrow$ *Stat4* : $\mathsf{Has\_cnfModel}_\Theta(s, m_1)$

According to Theorem $\mathsf{cnfModels}_{12}$, *we can construct a model for one of the two sets in the hypothesis.*

$\big\langle s, m_1, h_0\big\rangle \hookrightarrow T\,\mathsf{cnfModels}_{12}$ $(*Stat4*\star) \Rightarrow$
$\mathsf{Has\_cnfModel}_\Theta(\{\{h_0\}\} \cup \{c \setminus \{\backsim h_0\} : c \in s \,|\, h_0 \notin c\}, m_1 \cup \{h_0\}) \vee$
$\mathsf{Has\_cnfModel}_\Theta(\{\{\backsim h_0\}\} \cup \{c \setminus \{h_0\} : c \in s \,|\, \backsim h_0 \notin c\}, m_1 \cup \{\backsim h_0\})$

Both cases get discarded, as they contradict Stat2 and Stat3, respectively.

$\mathsf{Suppose} \Rightarrow$ *Stat5* :
$\mathsf{Has\_cnfModel}_\Theta(\{\{h_0\}\} \cup \{c \setminus \{\backsim h_0\} : c \in s \,|\, h_0 \notin c\}, m_1 \cup \{h_0\})$
$\big\langle m_1 \cup \{h_0\}\big\rangle \hookrightarrow$*Stat2*$(*Stat5*\star) \Rightarrow$ false
$\mathsf{Discharge} \Rightarrow$ *Stat6* :
$\mathsf{Has\_cnfModel}_\Theta(\{\{\backsim h_0\}\} \cup \{c \setminus \{h_0\} : c \in s \,|\, \backsim h_0 \notin c\}, m_1 \cup \{\backsim h_0\})$
$\big\langle m_1 \cup \{\backsim h_0\}\big\rangle \hookrightarrow$*Stat3*$(*Stat6*\star) \Rightarrow$ false; $\mathsf{Discharge} \Rightarrow$ AUTO

We now consider the reverse implication, and suppose that the first statement holds. By Theorem $\mathsf{cnfModels}_{14}$, *we arrive at a contradiction.*

$\mathsf{Suppose} \Rightarrow \mathsf{Is\_cnfSat}_\Theta(\{\{h_0\}\} \cup \{c \setminus \{\backsim h_0\} : c \in S \,|\, h_0 \notin c\})$
$\big\langle h_0, s\big\rangle \hookrightarrow T\,\mathsf{cnfModels}_{14}$ $(\star) \Rightarrow$ false; $\mathsf{Discharge} \Rightarrow$ AUTO

It remains to be shown that the second statement holds. Given that $\backsim\backsim h_0 = h_0$, *we can exploit Theorem* $\mathsf{cnfModels}_{14}$ *again. We have discarded all possible cases, and hence our proof is complete.*

$\big\langle h_0\big\rangle \hookrightarrow T\,\mathsf{cnfModels}_0$ $(\star) \Rightarrow$ $\backsim\backsim h_0 = h_0$
EQUAL $\Rightarrow$ $\mathsf{Is\_cnfSat}_\Theta(\{\{\backsim h_0\}\} \cup \{c \setminus \{\backsim\backsim h_0\} : c \in s \,|\, \backsim h_0 \notin c\})$
$\big\langle \backsim h_0, s\big\rangle \hookrightarrow T\,\mathsf{cnfModels}_{14}$ $(\star) \Rightarrow$ false; $\mathsf{Discharge} \Rightarrow$ QED

Some readers may be perplexed about an apparent terminology misuse in what precedes: in spite of our indication that the notions of model and satisfiability refer to formulae in conjunctive normal form, the formal definitions of the

$$
\begin{array}{l}
\textsc{Theory}\ \mathsf{globalizeTog}\ (\mathsf{T}) \\
\quad \mathsf{Svm}(\mathsf{T})\ \&\ \mathsf{T}^{\leftarrow} = \mathsf{T}\ \&\ \left\{\mathsf{p} \in \mathsf{T} \mid \mathsf{p}^{[1]} = \mathsf{p}^{[2]}\right\} = \emptyset \\
\Rightarrow\ \left(\mathsf{tog}_{\Theta}\right) \\
\quad \left\langle \forall \mathsf{x} \in \mathbf{domain}(\mathsf{T}) \mid \mathsf{T}\restriction\mathsf{x} \neq \mathsf{x}\ \&\ \mathsf{T}\restriction(\mathsf{T}\restriction\mathsf{x}) = \mathsf{x} \right\rangle \\
\quad \left\langle \forall \mathsf{x} \in \mathbf{domain}(\mathsf{T}) \mid \mathsf{tog}_{\Theta}(\mathsf{x}) = \mathsf{T}\restriction\mathsf{x} \right\rangle \\
\quad \left\langle \forall \mathsf{x} \mid \mathsf{tog}_{\Theta}(\mathsf{x}) \neq \mathsf{x}\ \&\ \mathsf{tog}_{\Theta}\left(\mathsf{tog}_{\Theta}(\mathsf{x})\right) = \mathsf{x} \right\rangle \\
\textsc{End}\ \mathsf{globalizeTog}
\end{array}
$$

Figure 3: A tool for globalizing a toggling map T to the entire universe of sets

predicates $\mathsf{Has\_cnfModel}_{\Theta}(\mathsf{S}, \_)$ and $\mathsf{Is\_cnfSat}_{\Theta}(\mathsf{S})$ do not enforce any typing restriction reflecting the idea that S should be a set of sets of literals. The very requirement that the correspondence $\backsim$ be defined globally (i.e., that $\backsim X$ yield a value for every set X) may look over-demanding, if one thinks that literals should be drawn from a specific set (say the set of non-null integer numbers) instead of from the class of all sets. The theory displayed in Figure 3 accommodates things for those who think that a better way of modeling negation would be by means of a self-inverse function whose domain is the specific set whose elements represent literals and truth constants. As a matter of fact, it provides a mechanism for extending any such "toggling map" into a global Galois correspondence. Once this patch is available, one will probably convene that working without the encumbrance of a typing discipline not only is safe but also offers some advantages to the proof designer.[4]

4. Specifying and checking the Davis-Putnam procedure

After having developed the ability to construct a global Galois correspondence, a set of signed symbols, and an encoding of falsehood (to be passed on to our next theory via its formal parameters $\backsim\mathsf{X}$, lits and fail), we can now proceed to writing an "archetype" version of the Davis-Putnam algorithm for propositional

[4]To clarify the theory interface shown in Figure 3, we must say that a pairing function $x,y \mapsto \langle x,y\rangle$ and projections $p \mapsto p^{[1]}$, $p \mapsto p^{[2]}$ have been defined globally so that $\left\langle \forall x,y \mid \langle x,y\rangle^{[1]} = x\ \&\ \langle x,y\rangle^{[2]} = y \right\rangle$. By the notation $\mathsf{Svm}(F)$ we indicate that F is a *map*, i.e. a set satisfying the property $F = \left\{\langle p^{[1]}, p^{[2]}\rangle : p \in F\right\}$, and is also *single-valued* in the sense that $\left\{\langle p,q\rangle : p \in F, q \in F \mid p \neq q\ \&\ p^{[1]} = q^{[1]}\right\} = \emptyset$. The notation $F\restriction X$, where F normally is a map—not a global function!—designates the *application* of F to X, defined as follows: $F\restriction X =_{\mathrm{Def}} \mathbf{arb}\left(\left\{p^{[2]} : p \in F \mid p^{[1]} = X\right\}\right)$. Finally, the *inverse* $F^{\leftarrow}$ and the *domain* of a map F are defined to be the respective sets $\left\{\langle p^{[2]}, p^{[1]}\rangle : p \in F\right\}$ and $\left\{p^{[1]} : p \in F\right\}$.

formulae in conjunctive normal form. We will define a recursive function dp_Θ that, given a set of clauses S, returns a model of S if S is satisfiable, otherwise returns a fictitious model containing the element fail. Let us note that the null set $\emptyset$ will represent falsehood when viewed as a clause, whereas it will represent truth when viewed as a set of clauses.

Within the davisPutnam theory (see Appendix C), we will apply the cnfModels theory submitting as input parameter to it the same function $\backsim\mathsf{X}$, and renaming its output predicates as $\mathsf{Has\_dpModel}_\Theta$ and $\mathsf{Is\_dpSat}_\Theta$, respectively.

A selection function $\mathsf{sl}(\mathsf{S})$ will be supplied to davisPutnam as a fourth parameter: this is supposed to provide a literal appearing in S, unless S is a trivial formula (i.e., a blatantly satisfiable or unsatisfiable one). The simplest such function is

$$\mathsf{sl}(\mathsf{S}) =_{\mathrm{Def}} \mathbf{arb}(\mathbf{arb}(\mathsf{S})).$$

A more efficient choice would be to select first literals within unit clauses, then pure literals, and finally arbitrary literals from the formula. Ouyang [13] analyzes the choice of branching rules, but such a discourse is beyond the scope of our paper.

The formulae which make sense as input for the satisfiability decision algorithm contain finitely many literals; accordingly, we define $\mathsf{clauSets}_\Theta$ to be the set of all formulae with a finite number of literals, all belonging to lits:

$$\mathsf{clauSets}_\Theta =_{\mathrm{Def}} \{\mathsf{s} \subseteq \mathcal{P}\,\mathsf{lits} \,|\, \mathsf{Finite}(\bigcup \mathsf{s})\}.$$

Let us introduce the notation $\mathsf{settled}_\Theta$ to represent, for any $\mathsf{s} \in \mathsf{clauSets}_\Theta$, the set of all literals h which occur as unit literals and also as pure literals in s and for which $\{h\}$ is the only clause in s that involves h:

$$\mathsf{settled}(\mathsf{S})_\Theta =_{\mathrm{Def}} \big\{\mathsf{u} : \mathsf{u} \in \mathsf{S}, \mathsf{h} \in \mathsf{u} \,|\, \mathsf{u} = \{\mathsf{h}\} \;\&\; \{\mathsf{h}, \backsim\mathsf{h}\} \cap \bigcup(\mathsf{S} \setminus \{\mathsf{u}\}) = \emptyset\big\}.$$

Using the definition of $\mathsf{Has\_dpModel}_\Theta$, we can deduce that $\bigcup\mathsf{settled}_\Theta(\mathsf{S})$ is included in any model of S; therefore, in order to simplify S, it suffices to select literals from $\mathsf{S} \setminus \mathsf{settled}_\Theta(\mathsf{S})$. Hence, we define the "picking" function:

$$\mathsf{pk}_\Theta =_{\mathrm{Def}} \big\{\big[\mathsf{x}, \mathsf{sl}\big(\mathsf{x} \setminus \mathsf{settled}_\Theta(\mathsf{x})\big)\big] : \mathsf{x} \in \mathsf{clauSets}_\Theta\big\}.$$

In sight of defining the Davis-Putnam procedure recursively, we must single out a well-founded relation over the family $\mathsf{clauSets}$ of all sets of clauses. (Incidentally, this approach will automatically ensure *termination* of the algorithm). A set of clauses is regarded as being "smaller" than another if its unsettled part involves fewer literals than the unsettled part of the other. The relationship just introduced is indeed *well-founded* over the family $\mathsf{clauSets}_\Theta$ of sets of clauses:

THEOREM. $\mathsf{G} \subseteq \mathsf{clauSets}_\Theta \;\&\; \mathsf{G} \neq \emptyset \rightarrow$
$$\Big\langle \exists \mathsf{m} \in \mathsf{G}, \forall \mathsf{v} \in \mathsf{G} \,|\, \neg \Big(\bigcup\big(\mathsf{v} \backslash \mathsf{settled}_\Theta(\mathsf{v})\big) \subseteq \bigcup\big(\mathsf{m} \backslash \mathsf{settled}_\Theta(\mathsf{m})\big) \;\&\; \bigcup\big(\mathsf{v} \backslash \mathsf{settled}_\Theta(\mathsf{v})\big) \neq \bigcup\big(\mathsf{m} \backslash \mathsf{settled}_\Theta(\mathsf{m})\big) \Big) \Big\rangle .$$

To be able to invoke the wellfounded_recursive_fcn theory already available in Ref (cf. [12]), we define an operator $\mathsf{f3\_dp}_\Theta$ and a predicate $\mathsf{P3\_dp}_\Theta$, both ternary, which will play the role of formal input parameters. The third argument of these is meant to designate a "picking" function like the one just defined. As regards the operator $\mathsf{f3\_dp}_\Theta$, its first and second argument generally designate a set s of clauses and a doubleton collection of models, one modeling $\mathsf{s} \cup \{\{\mathsf{p} \restriction \mathsf{s}\}\}$ and one modeling $\mathsf{s} \cup \{\{\backsim(\mathsf{p} \restriction \mathsf{s})\}\}$; the pseudo-model $\{\mathsf{fail}\}$ being used in place of either when a genuine model does not exists. When s is trivially satisfiable (e.g. null), as it coincides with its obvious part, then $\mathsf{f3\_dp}_\Theta(\mathsf{s}, \_, \_)$ supplies its model $\bigcup\mathsf{s}$; when s is manifestly false, as flagged by the null clause present in it, $\mathsf{f3\_dp}_\Theta(\mathsf{s}, \_, \_)$ returns $\{\mathsf{fail}\}$; when s is neither trivially satisfiable nor manifestly false, $\mathsf{f3\_dp}_\Theta(\mathsf{s}, \mathsf{m}, \mathsf{p})$ draws a model of s (if any) from m, giving priority to the model of $\mathsf{s} \cup \{\{\mathsf{p} \restriction \mathsf{s}\}\}$ if this exists but picking the model of $\mathsf{s} \cup \{\{\backsim(\mathsf{p} \restriction \mathsf{s})\}\}$ when the former does not exist (of course if neither has a model, $\mathsf{m} = \{\{\mathsf{fail}\}\}$ will hold).

$$\begin{array}{l} \mathsf{f3\_dp}_\Theta(\mathsf{S}, \mathscr{M}, \mathsf{P}) =_{\text{Def}} \text{if } \emptyset \in \mathsf{S} \text{ then } \{\mathsf{fail}\} \text{ else} \\ \quad \text{if } \mathsf{settled}_\Theta(\mathsf{S}) \supseteq \mathsf{S} \text{ then } \bigcup\mathsf{S} \text{ else} \\ \qquad \mathbf{arb}\big(\{\mathsf{w} \in \mathscr{M} \,|\, (\mathsf{fail} \notin \bigcup\mathscr{M} \rightarrow \mathsf{P} \restriction \mathsf{S} \in \mathsf{w}) \;\& \\ \qquad\qquad\qquad (\mathsf{fail} \in \mathsf{w} \rightarrow \langle \forall \mathsf{v} \in \mathscr{M} \,|\, \mathsf{fail} \in \mathsf{v} \rangle)\}\big) \\ \quad \text{fi} \\ \text{fi} \end{array}$$

As regards the predicate $\mathsf{P3\_dp}_\Theta$, its first argument again designates a set s of clauses, and its second designates the set s' resulting from the simplification of s when either the literal h or its complement $\backsim\mathsf{h}$ is assumed to be true, where h is selected from s by the picking function. It would be pointless to carry out the simplification relative to $\backsim\mathsf{h}$ when h is a pure literal, namely one whose complement does not occur in s: in this case the above-discussed operator $\mathsf{f3\_dp}_\Theta$ will receive a singleton, instead of a doubleton, set of models. There are situations in which the picking function is not guaranteed to return anything significant, but they cause no problem because in such cases the satisfiability check for s is obvious and can be performed directly by the operator $\mathsf{f3\_dp}_\Theta$ without reduction of s to a simpler s'.

$$\begin{aligned}&\mathsf{P3\_dp}_\Theta(S,S',P) \leftrightarrow_{\text{Def}} \big(\backsim(P\restriction S)\in\bigcup S\ \&\\ &\quad S'=\{\{\backsim(P\restriction S)\}\}\cup\{c\setminus\{P\restriction S\}: c\in S\,|\,\backsim(P\restriction S)\notin c\}\big)\ \vee\\ &\quad S'=\{\{P\restriction S\}\}\cup\{c\setminus\{\backsim(P\restriction S)\}: c\in S\,|\,P\restriction S\notin c\}\end{aligned}$$

Let us note that in the case when $P\restriction S$ is a pure literal, we have $\{\{P\restriction S\}\}\cup\{c\setminus\{\backsim(P\restriction S)\}: c\in S\,|\,P\restriction S\notin c\}=\{\{P\restriction S\}\}\cup\{c\in S\,|\,P\restriction S\notin c\}$.

By applying the THEORY well_recursive_fcn, we obtain:

$$\begin{aligned}&\textsc{Theorem}.\ \big\langle\forall x,p\,|\,x\in\mathsf{clauSets}_\Theta\to\mathsf{dp0}_\Theta(x,p)=\\ &\mathsf{f3\_dp}_\Theta(x,\{\mathsf{dp0}_\Theta(y,p): y\in\mathsf{clauSets}_\Theta\,|\,(\bigcup(y\setminus\mathsf{settled}_\Theta(y))\subseteq\bigcup(x\setminus\mathsf{settled}_\Theta(x))\\ &\qquad\qquad \&\ \bigcup(y\setminus\mathsf{settled}_\Theta(y))\neq\bigcup(x\setminus\mathsf{settled}_\Theta(x)))\ \&\ \mathsf{P3\_dp}_\Theta(x,y,p)\},p)\big\rangle.\end{aligned}$$

The following theorem states that the set resulting from the splitting rule is a finite set of literals and that it is smaller with respect to the well-founded relation introduced above.

$$\begin{aligned}&\textsc{Theorem}.\ X\in\mathsf{clauSets}_\Theta\ \&\ H\in\bigcup\big(X\setminus\mathsf{settled}_\Theta(X)\big)\ \&\\ &Y=\{\{H\}\}\cup\{c\setminus\{\backsim H\}: c\in X\,|\,H\notin c\}\to\\ &\quad Y\in\mathsf{clauSets}_\Theta\ \&\ \bigcup\big(Y\setminus\mathsf{settled}_\Theta(Y)\big)\neq\bigcup\big(X\setminus\mathsf{settled}_\Theta(X)\big)\ \&\\ &\quad\bigcup\big(Y\setminus\mathsf{settled}_\Theta(Y)\big)\subseteq\bigcup\big(X\setminus\mathsf{settled}_\Theta(X)\big).\end{aligned}$$

We are now ready to state the key theorem of our correctness verification:

$$\begin{aligned}&\textsc{Theorem}.\ S\in\mathsf{clauSets}_\Theta\to\mathsf{if}\ \mathsf{fail}\in\mathsf{dp0}_\Theta(S,\mathsf{pk}_\Theta\ \mathsf{then}\\ &\quad\neg\mathsf{Is\_dpSat}_\Theta(S)\ \mathsf{else}\ \mathsf{Has\_dpModel}_\Theta\big(S,\mathsf{dp0}_\Theta(S,\mathsf{pk}_\Theta)\big)\ \mathsf{fi}.\end{aligned}$$

This assertion gets proved by mathematical induction on the number $n=\#\bigcup\big(s\setminus\mathsf{settled}_\Theta(s)\big)$. If $n=\emptyset$, then $s\setminus\mathsf{settled}_\Theta(s)=\emptyset$ or $s\setminus\mathsf{settled}_\Theta(s)=\{\emptyset\}$. If the former alternative holds, then the value returned by $\mathsf{dp0}_\Theta$ will be $\bigcup s$, and actually we know from cnfModels (recalling the definition of $\mathsf{settled}_\Theta$) that this is a model of s. If the latter alternative holds, we have also $\emptyset\in s$ and thus $\mathsf{dp0}_\Theta$ will return the answer $\{\mathsf{fail}\}$ which is again correct by the definition of a model.

If $\emptyset\in n$, then pk_Θ will pick a literal h from one of the clauses in $s\setminus\mathsf{settled}_\Theta(s)$. If $\backsim(\mathsf{pk}_\Theta\restriction s)\notin\bigcup s$, then the only set that satisfies the predicate $\mathsf{P3\_dp}_\Theta$ is $s_1=\{\{\mathsf{pk}_\Theta\restriction s\}\}\cup\{c\in s\,|\,(\mathsf{pk}_\Theta\restriction s)\notin c\}$. As it has one fewer literal than s, we can apply the induction hypothesis and deduce that the resulting set m_1 is a model for s_1.

Otherwise, both $s_1=\{\{\backsim(\mathsf{pk}_\Theta\restriction s)\}\}\cup\{c\setminus\{\mathsf{pk}_\Theta\restriction s\}: c\in s\,|\,\backsim(\mathsf{pk}_\Theta\restriction s)\notin c\}$ and $s_2=\{\{\mathsf{pk}_\Theta\restriction s\}\}\cup\{c\setminus\{\backsim(\mathsf{pk}_\Theta\restriction s)\}: c\in s\,|\,(\mathsf{pk}_\Theta\restriction s)\notin c\}$ satisfy $\mathsf{P3\_dp}_\Theta$. As they have two fewer literals than s, by the induction hypothesis we deduce

that the models m_1 and m_2 returned by $\mathsf{dp0}_\Theta(\mathsf{s}_1)$ and $\mathsf{dp0}_\Theta(\mathsf{s}_2)$ are indeed models of s_1 and s_2, respectively.

For all the cases above, using the theorems on the pure literal rule, unit literal rule and the splitting rule in cnfModels, we can prove the statement for s, and show that the induction holds, which completes our proof.

To refine our results, we define the function dp_Θ that applies $\mathsf{dp0}_\Theta$ to the input formula deprived of all tautological clauses, and then enriches the model (if any) thus obtained, so that it becomes also a model of the initial formula.

DEF. $\mathsf{modelTaut}_\Theta(\mathsf{S},\mathsf{M}) =_{\text{Def}} \text{if } \mathsf{fail} \in \mathsf{M} \text{ then } \{\mathsf{fail}\} \text{ else}$
$\quad \mathsf{M} \cup \{\mathbf{arb}(\{\mathsf{h}, \backsim\mathsf{h}\}) : \mathsf{c} \in \mathsf{S}, \mathsf{h} \in \mathsf{c} \backslash \mathsf{M} \mid \backsim\mathsf{h} \in \mathsf{c} \backslash \mathsf{M}\};$
DEF. $\mathsf{dp}_\Theta(\mathsf{S}) =_{\text{Def}} \mathsf{modelTaut}_\Theta\big(\mathsf{S}, \mathsf{dp0}_\Theta(\mathsf{S} \setminus \{\mathsf{c} : \mathsf{c} \in \mathsf{S}, \mathsf{h} \in \mathsf{c} \mid \backsim\mathsf{h} \in \mathsf{c}\}, \mathsf{pk}_\Theta)\big);$
THEOREM. $\mathsf{S} \in \mathsf{clauSets}_\Theta \rightarrow \big(\mathsf{Is\_dpSat}_\Theta(\mathsf{S}) \leftrightarrow \mathsf{fail} \notin \mathsf{dp}_\Theta(\mathsf{S})\big);$
THEOREM. $\mathsf{S} \in \mathsf{clauSets}_\Theta \rightarrow \Big(\mathsf{Is\_dpSat}_\Theta(\mathsf{S}) \leftrightarrow \mathsf{Has\_dpModel}_\Theta\big(\mathsf{S}, \mathsf{dp}_\Theta(\mathsf{S})\big)\Big).$

5. Proof of the Compactness Theorem

It is remarkable that a single proof-development environment can provide support for the specification of a terminating algorithm (the Davis-Putnam procedure in our case-study), for the proof of its correctness, and also for the proof of such a general fact as the compactness theorem. For propositional logic, this fact can be stated by first introducing the notion of FINITE SATISFIABILITY, which refers to a set Ψ of sets S of finite clauses: in order that Ψ be finitely satisfiable, for every finite subset F of Ψ there must exist a model M which simultaneously satisfies all S in F. The compactness theorem states that any Ψ which is finitely satisfiable in this sense admits a model M which simultaneously satisfies all S in Ψ. A short, well-known proof of this fact can be derived from the Zorn lemma.

In rough outline, the proof goes as follows: let us consider the set Σ of all finitely satisfiable supersets Φ of Ψ such that

$$\big\{ x : s \in \Phi, c \in s, y \in c, x \in \{y, \backsim y\} \big\} = \big\{ x : s \in \Psi, c \in s, y \in c, x \in \{y, \backsim y\} \big\}.$$

It can be shown that every subset of Σ which is totally ordered by $\subseteq$ has an upper bound (relative to $\subseteq$) in Σ; therefore, by the Zorn lemma, Σ has at least one $\subseteq$-maximal element Φ_0. It can be shown that for every finitely satisfiable set Φ and for each $X \in \big\{ x : s \in \Phi, c \in s, y \in c, x \in \{y, \backsim y\} \big\}$, either $\Phi \cup \{\{\{X\}\}\}$ or $\Phi \cup \{\{\{\backsim X\}\}\}$ is finitely satisfiable: which means, when Φ is maximal (e.g. when $\Phi = \Phi_0$), that either $\{\{X\}\}$ or $\{\{\backsim X\}\}$ (and only one of the two, else the finite satisfiability would be contradicted) belongs to it. Hence we can define a model M_0 by collecting all those X for which $\{\{X\}\} \in \Phi_0$. Consider now an $S \in \Phi_0$ and a clause $c \in S$. Assuming by contradiction that c does not intersect

M_0, we would have $\{\{\backsim x\}\} \in \Phi_0$ for every $x \in c$, but then $\{S\} \cup \{\{\{\backsim x\}\} : x \in c\}$ would not be satisfiable, contradicting the finite satisfiability of Φ_0.

Carrying this argument out in Ref requires two proofs (one concerning the extensibility of finitely satisfiable sets, and one culminating in the compactness theorem). We found that the natural placement for such proofs is within the cnfModels theory discussed two sections ago (see boxed parts of Appendix B). 232 lines of proofware were needed to formalize these proofs; their verification takes about 2 seconds.

Conclusions and Further Work

In this paper we have described how to prove the correctness of a well-known algorithm in the ÆtnaNova theorem checker. The three theories introduced constitute the actual proof, showing that Ref can be used to tackle algorithm verification problems. Many algorithms can be specified, very naturally and in compact, high-level terms, by means of an executable language grounded on set theory (cf. [15]). Hence it would be desirable to enhance Ref with programming-specific notation, so that proving that a procedure behaves as desired could be done, in full, under the surveillance of the automated verifier.

This paper has also addressed, for the first time, the issue of how to "arithmetize" the syntax and the symbolic manipulations of a formal language; i.e., how to encode by means of set-theoretic constructions the formation rules, designation rules, deduction and/or rewriting rules of a formal system. The first step having now been done, we are confident that many situations will arise which can be tackled similarly. Indeed, thanks to the set-theoretic counterpart provided by the THEORY signedSymbols for the basic symbols of a language, $\in$-recursion can mimic structural recursion over the terms of its signature.

For a quick example, let us denote by $\mathscr{V}_\omega$ the collection of all the hereditarily finite sets (namely, those sets whose rank is a finite ordinal). We can specify the algorithm which reduces an arbitrary formula of classical propositional logic to *negation normal form* as follows:

$$
\begin{aligned}
rev\_opc(P) &=_{\text{Def}} \{\,\mathbf{arb}(P \cap \mathscr{V}_\omega)\,\} \cap 1\,;\\
flip(P) &=_{\text{Def}} \text{if } P \in \mathsf{lits} \text{ then } \backsim P\\
&\qquad \text{elseif } P \setminus \mathscr{V}_\omega = \{\,\mathbf{arb}(P \cap \mathsf{lits})\,\} \text{ then } flip\big(\mathbf{arb}(P \cap \mathsf{lits})\big)\\
&\qquad \text{else } \{\,rev\_opc(P)\,\} \cup \{\,flip(y) : y \in P \setminus \mathscr{V}_\omega\,\} \text{ fi}\,;\\
nnf(P) &=_{\text{Def}} flip\big(flip(P)\big).
\end{aligned}
$$

The idea, here, consistently with what has been proposed at the end of Section 2 (where we were focusing exclusively on literals), is that formulae (now also conjunctions and disjunctions, arbitrarily nested) are encoded by sets of transfinite rank. When we view a set P as the encoding of a formula, we regard $P \cap \mathscr{V}_\omega$ and $P \setminus \mathscr{V}_\omega$ as being its opcode and its operand; for specificity we can think that P encodes a disjunction, respectively a conjunction, depending on whether $\mathbf{arb}(P \cap \mathscr{V}_\omega)$ is 0 or is nonnull: correspondingly, since $0 = \emptyset$ and $1 = \{\emptyset\}$ hold by definition, the value of $rev\_opc(P)$ will be 1 or 0. In either case, we understand the operands of the disjunction/conjunction to be the formulae encoded by the elements of $P \setminus \mathscr{V}_\omega$. In the light of this, it should be clear that $flip$ simply implements the De Morgan's rules, plus a rule for unwrapping unit literals.

To see another example, referring now to propositional *mono-modal* logic, let us specify how to evaluate a formula P relative to a Kripke frame consisting of

- a set W of worlds and an accessibility relation R between worlds, and to
- a model M assigning a set of worlds to each propositional letter.

The evaluation rules are:

$$\mathsf{m\_vl}(Op,U,W,R) \quad =_{\text{Def}} \quad \mathsf{if}\ \ Op = 0\ \ \mathsf{then}\ \ U\ \ \mathsf{elseif}\ \ Op = 1\ \ \mathsf{then}\ \ W \setminus U \ \ \mathsf{else}\ \ \big\{ x \in W \,|\, U \supseteq R^{\triangleright} \{x\} \big\}\ \ \mathsf{fi}\,;$$

$$m\_eval(P,W,R,M) \quad =_{\text{Def}} \quad \mathsf{if}\ \ P \in \mathsf{domain}(M)\ \ \mathsf{then}\ \ M \restriction P\ \ \mathsf{else}$$
$$\mathsf{m\_vl}\big(\mathsf{arb}(P \cap \mathscr{V}_\omega), \bigcup \{ m\_eval(y,W,R,M) : y \in P \setminus \mathscr{V}_\omega \}, W, R, M\big)\ \ \mathsf{fi}\ .$$

Here we view again $P \cap \mathscr{V}_\omega$ as being the opcode of a formula, and we interpret $P \setminus \mathscr{V}_\omega$ as a disjunction, a negated disjunction, or a necessitated disjunction, when the value of $\mathbf{arb}(P \cap \mathscr{V}_\omega)$ is 0, 1, or any set different from these, respectively. The set of worlds where a disjunction is true is the union of the sets of worlds making its disjuncts true; a necessitated formula holds true in those worlds x such that every world accessible from x belongs to the set U of worlds that make the operand of necessitation true.

Acknowledgements

The authors would like to thank Jacob T. Schwartz (New York University) for his assistance with the development of the ÆtnaNova system.

We had pleasant and fruitful discussions with Giovanna D'Agostino (Univ. of Udine), who contributed to the "arithmetization" (set-theoretic rendering in our case) of semantic notions referring to propositional (even non-classical) logics, and with Marianna Nicolosi Asmundo (Univ. of Catania), who contributed to the proof of compactness.

REFERENCES

[1] M. Baaz - U. Egly - A. Leitsch, *Normal Form Transformations*, A. Robinson and A. Voronkov (eds.), Handbook of Automated Reasoning 275-333, Elsevier, 2001.

[2] D. Cantone - E.G. Omodeo - J.T. Schwartz - P. Ursino, *Notes from the logbook of a proof-checker's project*, N. Dershowitz ed. International symposium on verification (Theory and Practice) celebrating Zohar Manna's 1000000_2-th birthday. Springer-Verlag, LNCS 2772, 182-207, 2003.

[3] M. Davis - G. Logemann - D. Loveland, *A machine program for theorem-proving*, Comm. of the ACM 5 (7) (1962), 394-397.

[4] M. Davis - H. Putnam, *A Computing Procedure for Quantification Theory*, J. ACM 7 (3) (1960), 201-215.

[5] N. Dunford - J. T. Schwartz, *Linear operators. Part I: General theory*, Interscience Publishers, New York, 1958.

[6] J. Harrison, *Stålmarck's algorithm as a HOL derived rule*, J. von Wright, J. Grundy, and J. Harrison (Eds.), Theorem Proving in Higher Order Logics: 9^{th} International Conference, TPHOLs'96 (26–30 August 1996), Springer-Verlag, LNCS 1126, 221-234, 1996.

[7] J. Harrison, *Formalizing Basic Complex Analysis*, R. Matuszewski and A. Zalewska (Eds.), From Insight to Proof: Festschrift in Honour of Andrzej Trybulec, University of Białystok, Studies in Logic, Grammar and Rhetoric, vol. 10 (23), 151-165, 2007.

[8] J.-P. Keller - R. Paige, *Program derivation with verified transformations - A case study*, Comm. Pure Appl. Math. 48 (9-10) (1995), 1053-1113. *Special issue in honor of J.T. Schwartz.*

[9] P. Letouzey - L. Théry, *Formalizing Stålmarck's algorithm in Coq*, J. Harrison and M. Aagard (Eds.), Theorem Proving in Higher Order Logics: 13^{th} International Conference, TPHOLs 2000, Springer-Verlag, LNCS 1869, 387-404, 2000.

[10] T. Nipkow - L. C. Paulson - M. Wenzel, *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, Springer-Verlag, LNCS 2283, 2002.

[11] E.G. Omodeo - D. Cantone - A. Policriti - J.T. Schwartz, *A Computerized Referee*, O. Stock and M. Schaerf (Eds.), Reasoning, Action and Interaction in AI Theories and Systems, Essays Dedicated to Luigia Carlucci Aiello, LNAI 4155, 114-136, 2006.

[12] E.G. Omodeo - J.T. Schwartz, *A 'Theory' mechanism for a proof-verifier based on first-order set theory*, A. Kakas and F. Sadri (eds.), Computational Logic: Logic Programming and beyond, Essays in honour of Robert Kowalski, part II, Springer-Verlag, LNAI 2408, 214-230, 2002.

[13] M. Ouyang, *How good are branching rules in DPLL?*, Discrete Applied Mathematics 89 (1-3) (1998), 281-286.

[14] D. Prawitz - H. Prawitz - N. Voghera, *A Mechanical Proof Procedure and its Realization in an Electronic Computer*, J. ACM 7 (2) (1960), 102-128. Reprinted (with a commentary) in Automation of Reasoning 1, Classical Papers on Computational

Logic, 202-228, J. Siekmann and G. Wrightson (eds), Springer-Verlag, 1983.

[15] J. T. Schwartz - R. K. B. Dewar - E. Dubinsky - E. Schonberg, *Programming with Sets: An introduction to SETL*, Texts and Monographs in Computer Science. Springer-Verlag, New York, 1986.

[16] `http://www.settheory.com/Setl2/Ref_user_manual.html`

Appendix

A. The signedSymbols THEORY

THEORY signedSymbols(atms)
$\Rightarrow (\mathsf{aff}_\Theta, \mathsf{neg}_\Theta, \mathsf{lits}_\Theta, \mathsf{false}_\Theta)$
$\langle \forall x, y \mid x \neq y \rightarrow \mathsf{aff}_\Theta(x) \neq \mathsf{aff}_\Theta(y) \rangle$
$\langle \forall x \mid \mathsf{neg}_\Theta(\mathsf{neg}_\Theta(x)) = x \;\&\; \mathsf{neg}_\Theta(x) \neq x \rangle$
$\langle \forall x, y \mid \mathsf{aff}_\Theta(x) \neq \mathsf{neg}_\Theta(\mathsf{aff}_\Theta(y)) \rangle$
$\{\mathsf{aff}_\Theta(x) : x \in \mathsf{atms}\} \subseteq \mathsf{lits}_\Theta$
$\{\mathsf{neg}_\Theta(x) : x \in \mathsf{lits}_\Theta\} = \mathsf{lits}_\Theta$
$\mathsf{false}_\Theta \notin \mathsf{lits}_\Theta$
END signedSymbols

B. The cnfModels THEORY

THEORY cnfModels($\backsim x$)
$\langle \forall x \mid \backsim\backsim x = x \;\&\; \backsim x \neq x \rangle$
$\Rightarrow (\mathsf{Has\_cnfModel}_\Theta, \mathsf{Is\_cnfSat}_\Theta$, $\mathsf{Is\_cnfFinSat}_\Theta$)
$\langle \forall s \mid \mathsf{Has\_cnfModel}_\Theta(\emptyset, \emptyset) \;\&\; \mathsf{Is\_cnfSat}_\Theta(\emptyset) \;\&\; (\emptyset \in s \rightarrow \neg \mathsf{Is\_cnfSat}_\Theta(s)) \rangle$
$\langle \forall s \mid \{h \in \bigcup s \mid \backsim h \in \bigcup s\} = \emptyset \;\&\; \emptyset \notin s \rightarrow \mathsf{Has\_cnfModel}_\Theta(s, \bigcup s) \;\&\; \mathsf{Is\_cnfSat}_\Theta(s) \rangle$
$\langle \forall s \mid \{u : u \in s, h \in u \mid u = \{h\} \;\&\; \{h, \backsim h\} \cap \bigcup(s \setminus \{u\}) = \emptyset\} \supseteq s \rightarrow$
$\mathsf{Has\_cnfModel}_\Theta(s, \bigcup s) \;\&\; \mathsf{Is\_cnfSat}_\Theta(s) \rangle$
$\langle \forall t, s, m \mid t \subseteq s \;\&\; \mathsf{Has\_cnfModel}_\Theta(s, m) \rightarrow \mathsf{Has\_cnfModel}_\Theta(t, m) \rangle$
$\langle \forall t, s \mid t \subseteq s \;\&\; \mathsf{Is\_cnfSat}_\Theta(s) \rightarrow \mathsf{Is\_cnfSat}_\Theta(t) \rangle$
$\langle \forall s, m \mid \mathsf{Has\_cnfModel}_\Theta(s, m) \rightarrow \mathsf{Has\_cnfModel}_\Theta(s, m \cap \bigcup s) \rangle$
$\langle \forall s, m, n \mid \mathsf{Has\_cnfModel}_\Theta(s, m) \;\&\; \{h \in n \setminus m \mid \backsim h \in n\} = \emptyset \;\&\; m \subseteq n \rightarrow$
$\mathsf{Has\_cnfModel}_\Theta(s, n) \rangle$
$\langle \forall s, m, t \mid \mathsf{Has\_cnfModel}_\Theta(s, m) \rightarrow$
$\mathsf{Has\_cnfModel}_\Theta(\emptyset, m \cup \{\mathbf{arb}(\{k, \backsim k\}) : c \in t, k \in c \setminus m \mid \backsim k \in c \setminus m\}) \rangle$
$\langle \forall s, m, t \mid \mathsf{Has\_cnfModel}_\Theta(s, m) \rightarrow$
$\mathsf{Has\_cnfModel}_\Theta(s \cup \{c : c \in t, h \in c \mid \backsim h \in c\},$
$m \cup \{\mathbf{arb}(\{h, \backsim h\}) : c \in t, h \in c \setminus m \mid \backsim h \in c \setminus m\}) \rangle$
$\langle \forall s, m, h \mid \mathsf{Has\_cnfModel}_\Theta(s, m) \;\&$
$\neg \mathsf{Has\_cnfModel}_\Theta(\{\{h\}\} \cup \{c \setminus \{\backsim h\} : c \in s \mid h \notin c\}, m \cup \{h\}) \rightarrow$
$\backsim h \in m \rangle$
$\langle \forall s, m, h \mid \mathsf{Has\_cnfModel}_\Theta(s, m) \rightarrow$
$\mathsf{Has\_cnfModel}_\Theta(\{\{h\}\} \cup \{c \setminus \{\backsim h\} : c \in s \mid h \notin c\}, m \cup \{h\}) \vee$
$\mathsf{Has\_cnfModel}_\Theta(\{\{\backsim h\}\} \cup \{c \setminus \{h\} : c \in s \mid \backsim h \notin c\}, m \cup \{\backsim h\}) \rangle$

$$\Big\langle \forall h,s,m \,|\, \mathsf{Has\_cnfModel}_\Theta(\{\{h\}\} \cup \{c \setminus \{\backsim h\} : c \in s \,|\, h \notin c\}, m) \to \mathsf{Has\_cnfModel}_\Theta(s,m) \Big\rangle$$

$$\Big\langle \forall h,s \,|\, \mathsf{Is\_cnfSat}_\Theta(\{\{h\}\} \cup \{c \setminus \{\backsim h\} : c \in s \,|\, h \notin c\}) \to \mathsf{Is\_cnfSat}_\Theta(s) \Big\rangle$$

$$\Big\langle \forall s,h \,|\, \mathsf{Is\_cnfSat}_\Theta(s) \leftrightarrow \mathsf{Is\_cnfSat}_\Theta(\{\{h\}\} \cup \{c \setminus \{\backsim h\} : c \in s \,|\, h \notin c\}) \vee \mathsf{Is\_cnfSat}_\Theta(\{\{\backsim h\}\} \cup \{c \setminus \{h\} : c \in S \,|\, \backsim h \notin c\}) \Big\rangle$$

$$\Big\langle \forall h,s \,|\, \backsim h \notin \bigcup s \to \big(\mathsf{Is\_cnfSat}_\Theta(s) \leftrightarrow \mathsf{Is\_cnfSat}_\Theta(\{\{h\}\} \cup \{c \in s \,|\, h \notin c\})\big) \Big\rangle$$

$$\Big\langle \forall h,s,m \,|\, \{h\} \in s \;\&\; \mathsf{Has\_cnfModel}_\Theta(s,m) \to h \in m \;\&\; \backsim h \notin m \Big\rangle$$

$$\Big\langle \forall \Psi \,|\, \mathsf{Is\_cnfFinSat}_\Theta(\Psi) \leftrightarrow \Big\langle \forall f \subseteq \Psi \,|\, \mathsf{Finite}(f) \to \big\langle \exists m, \forall s \in f \,|\, \mathsf{Has\_cnfModel}_\Theta(s,m) \big\rangle \Big\rangle \Big\rangle$$

$$\Big\langle \forall \Psi \,|\; \mathsf{Is\_cnfFinSat}_\Theta(\Psi) \;\&\; \big\langle \forall s \in \Psi, c \in s \,|\, \mathsf{Finite}(c) \big\rangle \to \big\langle \exists m, \forall s \in \Psi \,|\, \mathsf{Has\_cnfModel}_\Theta(s,m) \big\rangle \Big\rangle$$

END cnfModels

C. The davisPutnam THEORY

THEORY $\mathsf{davisPutnam}\big(\backsim x, \mathsf{lits}, \mathsf{fail}, \mathsf{sl}(x)\big)$

$$\big\langle \forall x \,|\, \backsim\backsim x = x \;\&\; \backsim x \neq x \big\rangle$$

$$\{\backsim x : x \in \mathsf{lits}\} = \mathsf{lits}$$

$$\mathsf{fail} \notin \mathsf{lits}$$

$$\big\langle \forall s \,|\, s \neq \emptyset \;\&\; \emptyset \notin s \to \mathsf{sl}(s) \in \bigcup s \big\rangle$$

$$\Rightarrow (\mathsf{clauSets}_\Theta, \mathsf{Has\_dpModel}_\Theta, \mathsf{Is\_dpSat}_\Theta, \mathsf{dpModel}_\Theta, \mathsf{settled}_\Theta, \mathsf{pk}_\Theta, \mathsf{f3\_dp}_\Theta, \mathsf{P3\_dp}_\Theta, \mathsf{dp0}_\Theta, \mathsf{modelTaut}_\Theta, \mathsf{dp}_\Theta)$$

$$\big\langle \forall h,s \,|\, \backsim h \notin \bigcup s \to \{c \in s \,|\, h \notin c\} = \{c \setminus \{\backsim h\} : c \in s \,|\, h \notin c\} \big\rangle$$

$$\big\langle \forall s \,|\, s \in \mathsf{clauSets}_\Theta \to \big(\mathsf{Finite}(s) \;\&\; \mathsf{Finite}(\bigcup s) \;\&\; \bigcup s \subseteq \mathsf{lits} \;\&\; \mathsf{fail} \notin \bigcup s \;\&\; \backsim \mathsf{fail} \notin \bigcup s\big) \big\rangle$$

$$\big\langle \forall s,c \,|\, \big(s \in \mathsf{clauSets}_\Theta \;\&\; c \in s\big) \to \mathsf{Finite}(c) \big\rangle$$

$$\big\langle \forall s,t \,|\, \big(s \in \mathsf{clauSets}_\Theta \;\&\; (t \subseteq s \vee \bigcup t \subseteq \bigcup s)\big) \to t \in \mathsf{clauSets}_\Theta \big\rangle$$

$$\big\langle \forall s,m \,|\, \mathsf{Has\_dpModel}_\Theta(s,m) \leftrightarrow \{h \in m \,|\, \backsim h \in m\} = \emptyset \;\&\; \{c \in s \,|\, m \cap c = \emptyset\} = \emptyset \big\rangle$$

$$\big\langle \forall s \,|\, \mathsf{Is\_dpSat}_\Theta(s) \leftrightarrow \big\langle \exists m \,|\, \mathsf{Has\_dpModel}_\Theta(s,m) \big\rangle \big\rangle$$

$$\big\langle \forall g \,|\, (g \subseteq \mathsf{clauSets}_\Theta \;\&\; g \neq \emptyset) \to (\exists m \in g \,|\, \forall v \in g \,|\, \neg\big(\bigcup(v \setminus \mathsf{settled}_\Theta(v)) \subseteq \bigcup(m \setminus \mathsf{settled}_\Theta(m)) \;\&\; \bigcup(v \setminus \mathsf{settled}_\Theta(v)) \neq \bigcup(m \setminus \mathsf{settled}_\Theta(m))\big) \big\rangle$$

$$\big\langle \forall x,p \,|\, x \in \mathsf{clauSets}_\Theta \to \mathsf{dp0}_\Theta(x,p) = \mathsf{f3\_dp}_\Theta\Big(x, \{\mathsf{dp0}_\Theta(y,p) : y \in \mathsf{clauSets}_\Theta \,|\, \Big(\bigcup\big(y \setminus \mathsf{settled}_\Theta(y)\big) \subseteq \bigcup\big(x \setminus \mathsf{settled}_\Theta(x)\big) \;\&\; \bigcup\big(y \setminus \mathsf{settled}_\Theta(y)\big) \neq \bigcup\big(x \setminus \mathsf{settled}_\Theta(x)\big)\Big) \;\&$$

$$\text{P3\_dp}_\Theta(\text{x},\text{y},\text{p})\},\text{p}\Big)\Big\rangle$$

$$\big\langle \forall \text{s} \,|\, \text{s} \in \text{clauSets}_\Theta \rightarrow \#\bigcup\big(\text{s}\backslash\text{settled}_\Theta(\text{s})\big) \in \text{Za}\big\rangle$$

$$\big\langle \forall \text{x},\text{h},\text{y} \,|\, \text{x} \in \text{clauSets}_\Theta \;\&\; \text{h} \in \bigcup\big(\text{x}\backslash\text{settled}_\Theta(\text{x})\big) \;\&$$
$$\text{y} = \{\{\text{h}\}\} \cup \{\text{c}\backslash\{\backsim\text{h}\} : \text{c} \in \text{x} \,|\, \text{h} \notin \text{c}\} \rightarrow$$
$$\text{y} \in \text{clauSets}_\Theta \;\&\; \bigcup\big(\text{y}\backslash\text{settled}_\Theta(\text{y})\big) \neq \bigcup\big(\text{x}\backslash\text{settled}_\Theta(\text{x})\big) \;\&$$
$$\bigcup\big(\text{y}\backslash\text{settled}_\Theta(\text{y})\big) \subseteq \bigcup\big(\text{x}\backslash\text{settled}_\Theta(\text{x})\big)\big\rangle$$

$$\big\langle \forall \text{s} \,|\, \text{s} \in \text{clauSets}_\Theta \;\&\; \emptyset \notin \text{s} \;\&\; \text{s} \not\subseteq \text{settled}_\Theta(\text{s}) \rightarrow$$
$$\text{pk}_\Theta \restriction \text{s} \in \bigcup\big(\text{s}\backslash\text{settled}_\Theta(\text{s})\big) \;\&\; \text{pk}_\Theta \restriction \text{s} \neq \text{fail} \;\&$$
$$\backsim(\text{pk}_\Theta \restriction \text{s}) \neq \text{fail} \;\&\; \Big(\backsim(\text{pk}_\Theta \restriction \text{s}) \in \bigcup \text{s} \rightarrow \backsim(\text{pk}_\Theta \restriction \text{s}) \in \bigcup\big(\text{s}\backslash\text{settled}_\Theta(\text{s})\big)\Big)\Big\rangle$$

$$\big\langle \forall \text{s} \,|\, \text{s} \in \text{clauSets}_\Theta \;\&\; \emptyset \in \text{s} \vee \text{s} \subseteq \text{settled}_\Theta(\text{s}) \vee \#\bigcup\big(\text{s}\backslash\text{settled}_\Theta(\text{s})\big) = \emptyset \rightarrow$$
$$\text{if fail} \in \text{dp0}_\Theta(\text{s},\text{pk}_\Theta) \text{ then } \neg\text{Is\_dpSat}_\Theta(\text{s})$$
$$\text{else Has\_dpModel}_\Theta\big(\text{s},\text{dp0}_\Theta(\text{s},\text{pk}_\Theta)\big) \text{ fi}\big\rangle$$

$$\big\langle \forall \text{s} \,|\, \text{s} \in \text{clauSets}_\Theta \rightarrow$$
$$\text{if fail} \in \text{dp0}_\Theta(\text{s},\text{pk}_\Theta) \text{ then } \neg\text{Is\_dpSat}_\Theta(\text{s})$$
$$\text{else Has\_dpModel}_\Theta\big(\text{s},\text{dp0}_\Theta(\text{s},\text{pk}_\Theta)\big)\big\rangle \text{ fi}$$

$$\big\langle \forall \text{s} \,|\, \text{s} \in \text{clauSets}_\Theta \rightarrow \big(\text{Is\_dpSat}_\Theta(\text{s}) \leftrightarrow \text{fail} \notin \text{dp}_\Theta(\text{s})\big)\big\rangle$$

$$\big\langle \forall \text{s} \,|\, \text{s} \in \text{clauSets}_\Theta \rightarrow \Big(\text{Is\_dpSat}_\Theta(\text{s}) \leftrightarrow \text{Has\_dpModel}_\Theta\big(\text{s},\text{dp}_\Theta(\text{s})\big)\Big)\Big\rangle$$

END davisPutnam

EUGENIO G. OMODEO
Dipartimento di Matematica e Informatica
Università di Trieste
e-mail: eomodeo@units.it

ALEXANDRU I. TOMESCU
University of Bucharest
Faculty of Mathematics and Computer Science
e-mail: alexandru.tomescu@gmail.com