TouIST, a Teacher-and Student-Friendly Language for Propositional Logic and Discrete Mathematics

Olivier Gasquet, Dominique Longin, Emiliano Lorini, Frédéric Maris, Pierre Régnier, Sergei Soloviev

Computer assisted mathematics, 2021
№ -: 1–12
http://cte.eltech.ru

TouIST, a Teacher-and Student-Friendly Language for Propositional Logic and Discrete Mathematics

Gasquet O. [1], Full Professor, Olivier.Gasquet@irit.fr
Longin D. [2], Research Scientist, Dominique.Longin@irit.fr
Lorini E. [2], Research Director, Emiliano.Lorini@irit.fr
Maris F. [1], Associate Professor, Frederic.Maris@irit.fr
Régnier P. [1], Associate Professor, Pierre.Regnier@irit.fr
Soloviev S. [1], Full Professor, Sergei.Soloviev@irit.fr

[1]IRIT - University Toulouse 3, 118 route de Narbonne, Toulouse, FR 31062, France
[2]IRIT - CNRS, 118 route de Narbonne, Toulouse, FR 31062, France

Abstract

This work deals with logical formalization and problem solving using automated solvers. We present the automatic translator TouIST that provides a simple language to generate logical formulas from a problem description. Our tool allows us to model many static or dynamic combinatorial problems. All this can be very helpful as a teaching support for logics and discrete mathematics. Users will benefit from the regular improvements of SAT, QBF or SMT solvers in order to solve concrete logical and combinatorial problems efficiently, e.g., different classes of planning tasks in Artificial Intelligence.

Keywords: *keywords in English, more keywords in English*

Citation: Gasquet O. , Longin D. , Lorini E. , Maris F. , Régnier P. , Soloviev S. . TouIST, a Teacher- and Student-Friendly Language for Propositional Logic and Discrete Mathematics. Computer assisted mathematics, 2021. № -. P. 1–12. .

Acknowledgements: *here we may thank everybody who helped authors with this paper, and name grants that supported the research and the paper.*

1. INTRODUCTION

Every university teacher teaching logic or discrete mathematics to undergraduates would probably appreciate a good user-friendly program with simple interface that may be used during the lectures for simple (but not too simple) demos and explanations. For example, tape some semi-formal text such as “not a or b and c” on the keyboard, then display it using projector or zoom on the screen together with the formula $\neg a \vee b \wedge c$ and in a few seconds show its truth table. Or, another example, define and display the graph of the membership relation $\in \subseteq X \times 2^X$ for some finite set X.

All this is easily done with TouIST, as well as many others more advanced examples. On the one hand, the tool can be used by researchers for various tasks involving propositional calculus ranging from computation of models to computation of logical encodings of problems, for example of symbolic AI problems such as planning tasks. On the other hand, TouIST is a pedagogical tool to show the power of propositional logic to students who have been trained a couple of hours to formalize sentences in logic and who have acquired basic notions of validity and satisfiability:

it allows them to automatically solve some combinatorial puzzles. Simplicity of logical encoding (e.g., indexed propositional variables like $p_{i,j}$ may be used to represent predicates over finite domain) permits also to use TouIST to illustrate many algebraic and combinatorial notions and even doing some higher order logic and dependent types in finite case.

2. TouIST: AN OUTLINE

TouIST offers a high-level friendly language for logically modeling various problems in a very compact way using solvers. It consists of: a graphical interface allowing interactive input of the target model; a translation module (compiler) from the input language of TouIST into a language directly understandable by different solvers; a module for viewing models calculated by the solvers. This flexible design permits to use TouIST with different solvers including those that admit, for example, extended resolution or Frege proof systems [3].

Currently, TouIST can call on three different types of solvers: SAT solvers (propositional logic or logic of predicates on finite domain), QBF (authorizing quantification on propositional formulas), SMT (SAT Modulo Theories, for the treatment of problems involving numeric calculus on integer or rational numbers). The goal of the platform is to allow the user to concentrate on the modeling of a given problem without worrying about the technical details related to the use of solvers: linear space translation of formulas into prenex and conjunctive normal form using extension rules [12], then translation into language DIMACS, QDIMACS or SMT-LIB depending on the selected solver (languages used as standard as input to solvers but not very easy to handle directly). Beyond the Boolean connectives of propositional logic, the input language of TouIST has sets, conjunctions and disjunctions parametrized by sets, abbreviations…We can directly express complex propositional formulas such as:

$$\bigwedge_{i\in\{1..N\}} \bigvee_{X\in S(i)} \bigwedge_{n\in \mathrm{X}} \bigwedge_{m\in \mathrm{Y}|m\neq n} (p_{i,X,n} \Rightarrow \neg p_{i,X,m})$$

where we can define the variable N as a particular integer, the $S(i)$ as sets of sets of symbols for each $i \in \{1,\ldots,N\}$, and Y as a set of symbols. For example, if $N = 2$, $S(1) = \{\{blue, red\}, \{red\}\}$, $S(2) = \{\{red\}, \{blue\}, \{white, blue, red\}\}$, and $Y = \{white, red\}$, we write in the TouIST input language:

```
$N = 2
$S(1) = [[blue, red], [red]]
$S(2) = [[red], [blue], [white, blue, red]]
$Y = [white, red]
bigand $i in [1..$N]:
  bigor $X in $S($i):
    bigand $n in $X:
      bigand $m in $Y when $m!=$n:
        p($i,$X,$n) => not p($i,$X,$m)
      end
    end
  end
end
```

We can also use multiple binding of indexes as in $\bigwedge_{i\in A, j\in B}$ and rich computations on indexes as well as on domain sets as in $\bigwedge_{i\in(A\cup(B\cap C))}$, expressed in the TouIST input language as:

```
bigand $i,$j in $A,$B:
  ...
end
bigand $i in $A union ($B inter $C):
  ...
end
```

One may remember that $\bigwedge$ and $\bigvee$ represent also universal quantifier $\forall$ and existential quantifier $\exists$ over finite sets of indexes.

Running the solver only consists in clicking a button and the tool displays the models successively computed by the solvers in the syntax of the input formula. Literals of interest can be filtered by regular expressions. Moreover, TouIST can also be used entirely from the command line and/or batch modus for interfacing with intelligent agent architectures capable of reasoning and planning actions: typically for example, checking the validity of an argument, determining the executable actions, checking that a plan is valid or even calculate a complete plan of actions to satisfy a goal, etc.

TouIST is publicly available for download from the following site: https://www.irit.fr/TouIST/

In the sequel, we introduce features of TouIST for propositional and higher-order logics, and then several examples of static and dynamic reasoning that will serve to demonstrate our tool.

3. PROPOSITIONAL AND HIGHER-ORDER LOGICS IN TouIST

3.1. Basic features for propositional logic

First, we present how to use TouIST to easily verify essential properties of logical formulas, such as satisfiability, validity or logical consequence. Checking the satisfiability of a set of formulas is reduced to determining the existence of a model satisfying each formula from the set. In TouIST, a set of formulas is materialized by newlines. So, to verify that the set $\{coffee \vee tea, \neg tea\}$ is satisfiable, we simply write:

```
coffee or tea
not tea
```

If we wanted to check the validity of the previous formula, we would have to iterate through the models one by one and verify that the set of models is equal to the set of possible valuations for that formula. For this formula, there are only three propositional variables, that is $2^3 = 8$ valuations, so the task is still possible. But as soon as the formula contains more variables, we proceed indirectly by refutation. Thus, the formula is valid if and only if its negation is unsatisfiable.

Concretely, the validity test makes it possible for example to verify that a reasoning is formally valid (but this does not exclude the possibility of false premises or conclusions). For example :

$$rain \wedge (rain \Rightarrow wetRoad) \wedge (wetRoad \Rightarrow danger) \Rightarrow danger$$

In TouIST, proving the validity of such a formula by testing the unsatisfiability of its negation gives:

```
not (rain and (rain => wetRoad)
     and (wetRoad => danger) => danger)
```

Suppose now that $H = \{H_1, \ldots, H_n\}$ is a set of n assumptions (logical formulas) and C a formula. Again, it would be tedious to verify that C is true for all models of H. Here again we proceed indirectly by using the property: $H \models C$ if and only if $H \cup \{\neg C\}$ is unsatisfiable. in other words, to check if $H \models C$, we will test if the formulas H_1, H_2, …, H_n, $\neg C$ taken *all together* are satisfiable. If this is not the case we can conclude that $H \models C$. If this is the case, we will have at least one counter-model which will tell us in which situation we have the true assumptions and the false conclusion.

For example, suppose we have the following set H of rules and facts:

H_1. If the patient has measles, he or she has fever.
H_2. If the patient has hepatitis, but not measles, he or she has a yellow complexion.
H_3. If the patient has fever or a yellow complexion, he or she has hepatitis or measles.
H_4. The patient does not have a yellow complexion.
H_5. The patient has fever.

We want to verify that $H \models measles$. For that, we can verify that in all models of H, the variable *measles* (corresponding to "the patient has measles") is true. But it is much more easier with TouIST to verify that the following set of formulas is unsatisfiable:

```
measles => fever
hepatitis and (not measles) => yellowcomplexion
fever or yellowcomplexion => hepatitis or measles
not yellowcomplexion
fever
not measles
```

3.2. Higher-Order Logics and Dependent Quantifiers over Finite Sets

Predicates over a finite set X may be encoded using indexation by subsets of X. For example, for unary predicates on the set $\{1,...,3\}$ one may define:

```
$N = 3
$Y = powerset([1..$N])
bigand $j in $Y:
  (p($j)<=> (bigand $i in $j: q($i) end) and
  (bigand $i in $j when (not ($i in $j)): not q($i) end))
end
```

That will produce the formula

$$\bigwedge_{j\in Y}(p_j \iff \bigwedge_{i\in j} q_i \wedge \bigwedge_{i\notin j} \neg q_i).$$

It says that p_j is true exactly on elements of $j \subset [1..N]$. Using such definition of variables for predicates one may express the formulas of second order logic, e.g. the constant $False$ can be represented by $\bigwedge_{j\in Y} p_j$ (that corresponds to $\forall P.P$).

Iterating the *powerset* construction we may represent formulas of higher order logics over finite sets.

E.g., using already defined variables one may consider the formula

```
bigand $i in [1...$N]:
bigor $j in $Y when $i in $j:
p($j)
end
end
```

Notice that j depends on i, dropping the condition that i belongs to j another (non-equivalent) formula will be obtained.

Also, to the previous definition one may add one more "level":

```
$Z = powerset($Y)
bigand $k in $Z when $k!=[]:
  (s($k)<=> (bigor $j in $k: p($j) end)
end
```

That is, the indexes k are the families of subsets of X (empty family is excluded) and s are defined as disjunctions of p over subsets $j \in k$.

These examples and the examples of section 2 show how to obtain dependent types $S(i)$. Respectively, $\bigwedge$ and $\bigvee$ with appropriate dependencies between indexes will represent dependent $\forall$ and $\exists$ over finite sets.

Applications may be quite unexpected. As one of less evident let us mention modern linguistics. In [7] some applications of dependent types to linguistic analysis are considered. In particular, dependencies between linguistic quantifiers (such as “all”, “some”, “no one”) may influence the order of words in phrases of natural language or their meaning interpretation. Finite models (supported by `TouIST`) are usually sufficient there and `TouIST` may be used for illustrative purposes or as a tool of analysis.

4. STATIC REASONING WITH `TouIST`

4.1. Solving Puzzles with SAT

`TouIST` allows us to encode and solve static generalized games such as the well known Sudoku for a $N \times N$ grid (composed by N regions, i.e. grids of size $R \times R$ with $N = R^2$). For example, to express that each cell must have at least one value we write the formula:

$$\bigwedge_{i \in [1..N]} \bigwedge_{j \in [1..N]} \bigvee_{k \in [1..N]} p(i,j,k)$$

where $p(i,j,k)$ means that cell (i,j) has value k.
This formula is expressed in the `TouIST` input language as:

```
bigand $i,$j in [1..$N],[1..$N]:
  bigor $k in [1..$N]:
    p($i,$j,$k)
  end
end
```

Then we can express the fact that a cell has exactly one value by adding that each cell must have at most one value:

```
bigand $i in [1..$N]:
  bigand $j in [1..$N]:
    bigand $k1 in $L:
      bigand $k2 in $L$ when $k1!=$k2:
        not p($i,$j,$k1) or not p($i,$j,$k2)
      end end end end
```

Two very similar rules can be added to impose that each value is present at most once in each row or column. Finally, a last rule constraints that each value is present at most once in each region of the grid:

```
bigand $ir in [0..$R-1]:
  bigand $jr in [0..$R-1]:
    bigand $ic1 in [1..$R]:
      bigand $jc1 in [1..$R]:
        bigand $ic2 in [1..$R]:
          bigand $jc2 in [1..$R] when $ic1!=$ic2 or $jc1!=$jc2:
            bigand $k in $L:
              not p($R*$ir+$ic1,$R*$jr+$jc1,$k) or not p($R*$ir+$ic2,$R*$jr+$jc2,$k)
            end end end end end end end
```

It also allows us to solve well-known puzzles and games involving epistemic deductive reasoning, given existing polynomial embeddings of fragments of epistemic logic into propositional logic. This includes “Guess Who?” and the muddy children puzzle [1].

4.2. Solving Puzzles with SMT

In a similar way to Sudoku, the Binario (binary game) consists in filling a grid by deduction with only 0s and 1s. It is possible to model it in propositional logic, but to obtain a more compact encoding one can use SMT (SAT Modulo Theories) with atoms of QF-LIA (linear arithmetic on integers).[1]

In particular we can encode the rule "each row and column must contain as many 0s as 1s" by

$$\bigwedge_{i=1}^{N_R}\left(\sum_{j=1}^{N_C} x_{i,j} = \frac{N_C}{2}\right) \quad \wedge \quad \bigwedge_{j=1}^{N_C}\left(\sum_{i=1}^{N_R} x_{i,j} = \frac{N_R}{2}\right)$$

where N_R is the number of rows of the grid and N_C is the number of columns.

Another rule is that "there is no more than two of either number adjacent to each other", and is expressed by

$$\bigwedge_{i=1}^{N_R}\bigwedge_{j=1}^{N_C-2}\left(\left(\bigvee_{k=0}^{2}(x_{i,j+k}\neq 0)\right)\wedge\left(\bigvee_{k=0}^{2}(x_{i,j+k}\neq 1)\right)\right)$$

$$\bigwedge_{j=1}^{N_C}\bigwedge_{i=1}^{N_R-2}\left(\left(\bigvee_{k=0}^{2}(x_{i+k,j}\neq 0)\right)\wedge\left(\bigvee_{k=0}^{2}(x_{i+k,j}\neq 1)\right)\right)$$

```
bigand $i,$j in [1..$NR-2],[1..$NC]:
  x($i,$j)!=x($i+1,$j) or x($i+1,$j)!=x($i+2,$j)
end
bigand $i,$j in [1..$NR],[1..$NC-2]:
  x($j,$i)!=x($j,$i+1) or x($j,$i+1)!=x($j,$i+2)
end
```

And finally, there can be no identical rows or columns:

$$\left(\bigwedge_{i_1=1}^{N_R}\bigwedge_{\substack{i_2=1\\ i_1\neq i_2}}^{N_R}\bigvee_{j=1}^{N_C}(x_{i_1,j}\neq x_{i_2,j})\right)\wedge\left(\bigwedge_{j_1=1}^{N_C}\bigwedge_{\substack{j_2=1\\ j_1\neq j_2}}^{N_C}\bigvee_{i=1}^{N_R}(x_{i,j_1}\neq x_{i,j_2})\right)$$

```
bigand $i,$j in $N,$N when $i!=$j:
    bigor $k in $N:
      (x($i,$k)!=x($j,$k))
    end
  and
    bigor $k in $N:
      (x($k,$i)!=x($k,$j))
    end
end
```

[1] The theories QF-IDL, QF-RDL (difference logic on integers/rationals) and QF-RDL (linear arithmetic on rationals) are also available in TouIST.

4.3. Applications to Algebra

Since indexed propositional variables in TouIST may be routinely used to represent predicates over finite sets, it is convenient to use predicates to represent algebraic operations and express their properties via $\bigwedge$ and $\bigvee$. For exemple, $x_{i,j,k}$ may represent $i \times j = k$ for some binary operation $\times : X \times X \to X$ on a finite set X. Then the formula

$$\bigwedge_{i\in X}\bigwedge_{j\in X}\bigwedge_{k\in X} x_{i,j,k} \Rightarrow x_{j,i,k}$$

will represent commutativity of $\times$. Associativity of $\times$ will be expressed by

$$\bigwedge_{i\in X}\bigwedge_{j\in X}\bigwedge_{k\in X}\bigwedge_{l\in X}\bigwedge_{m\in X}\bigwedge_{n\in X}(x_{i,j,k} \wedge x_{j,l,m} \wedge x_{k,l,n} \Rightarrow x_{i,m,n}).$$

The formulas

$$\bigwedge_{i\in X}\bigwedge_{j\in X}\bigvee_{k\in X} x_{i,j,k}$$

and

$$\bigwedge_{i\in X}\bigwedge_{j\in X}\bigwedge_{k\in X}\bigwedge_{l\in X l\neq k}(x_{i,j,k} \Rightarrow \neg x_{i,j,l})$$

express the fact that $\times$ is defined for all i, j and has unique value k. They may be easily written using TouIST.

TouIST is sufficiently powerful to produce all such operations $\times$ for the sets X that are large enough for teaching purposes. Each operation will be displayed as a line in the truth table. Moreover, TouIST may be used to verify quickly simple equalities modulo theory, associativity and commutativity in this example.

5. DYNAMIC REASONING WITH TouIST

5.1. Simulating a 2-players game

The principle of the Nim's game is as follows: we have at the start a non-zero number NM of matches and a number NP of players can take 1 or more match(es). The player who loses is the one who, first, can no longer take a match. [2] The number of possible turns of play is at most equal to that of matches (at least, each player takes only one match at each turn). Thus, the set of indices of the possible turns is $T = \{0, \ldots, NM\}$ where 0 is the index of the initial state. Likewise, the set of possible numbers of matches still available is $M = \{0, 1, \ldots, NM\}$.

5.1.1. Logical description of Nim's game

In order to simplify the language used as much as possible, we are modeling here a variant where $NM = 4$ and $NP = 2$. The players are denoted by 0 and 1 and it is 0's turn to play on turn t if $turn\_of\_0(t)$ is true (considering that if it is not the turn of 0 then it is that of 1). Moreover, $left(t, n)$ is true iff on turn t there are n matches remaining.

[2] There are different variations of this game, in particular by varying the numbers of matches and players, but also by varying the possible actions or by introducing constraints (for example, one cannot take the same number of matches as the previous player).

Thus, the initial state of the game is as follows:

$$left(0, NM) \wedge turn\_of\_0(0) \tag{5.1}$$

meaning that in turn 0 there is still NM matches available and that it is 0's turn to play.

In the version presented here of the Nim's game, we also limit the number of possible actions to two: a player can take either 1 match, or 2 matches. Thus, $takes\_2(t)$ is true iff a player takes 2 matches on turn t (considering that $takes\_2(t)$ is false iff she takes only one).

So:

$$\bigwedge_{\substack{t\in T\\ n\in M\\ n\geq 2}} \Big(\Big(left(t,n) \wedge takes\_2(t) \Rightarrow left(t+1, n-2) \Big) \wedge \Big(left(t,n) \wedge \neg takes\_2(t) \Rightarrow left(t+1, n-1) \Big) \Big) \tag{5.2}$$

captures the fact that if on turn t there are at least 2 matches left and a player takes 2 then in the next turn 2 less remains, and if she takes only one then on the next round there is 1 less.

On the other hand, if in turn t there is exactly 1 match left, then necessarily the player will take 1 and there will be 0 left in the following turn:

$$\bigwedge_{t\in T} \Big(left(t,1) \Rightarrow \neg takes\_2(t) \wedge left(t+1, 0) \Big) \tag{5.3}$$

Our model then specifies that:

$$\bigwedge_{t\in T} \bigvee_{n\in M} left(t,n) \tag{5.4}$$

$$\bigwedge_{\substack{t\in T\\ n_1,n_2\in M\\ n_1\neq n_2}} \Big(left(t,n_1) \Rightarrow \neg left(t,n_2) \Big) \tag{5.5}$$

The first formula states that on each turn t there is at least one number n of matches remaining, and the second that this number is unique.

We must now define when a player has lost:

$$0\_lost \Leftrightarrow \bigvee_{\substack{t\in T\\ t>0}} \Big(turn\_of\_0(t) \wedge left(t,0) \Big) \tag{5.6}$$

means that player 0 has lost iff there is a turn t where there are 0 matches remaining when on the previous turn there was at least one.

Finally, at each turn t, it is not for the player 0 to play iff it is for her to play the following turn:

$$\bigwedge_{t\in T\setminus\{NM\}} \Big(\neg turn\_of\_0(t) \Leftrightarrow turn\_of\_0(t+1) \Big) \tag{5.7}$$

5.1.2. Finding a Winning Strategy

The language of QBF allows us to express naturally and concisely the existence of winning strategies as described in [6]. The moves of player 0 (for whom we are searching for a winning strategy) will be existentially quantified while those of his opponent will be universally quantified: we look for the moves of player 0 which will lead him to victory regardless of the moves made by player 1. `TouIST` natively integrates the QBF solver *Quantor 3.2* [2] and can be interfaced with other solvers supporting the QDIMACS format. Selecting this prover in `TouIST` allows us to use quantifiers ∀ and ∃ on propositional variables.

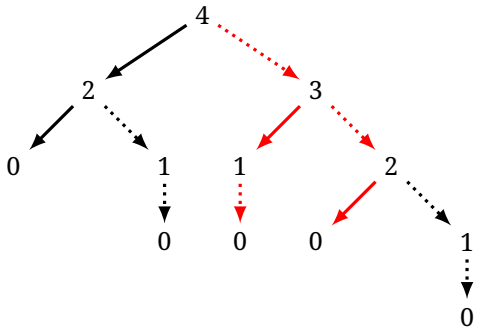

Figure 1. Solutions for Nim's game with 4 matches and 2 players. The winning strategy for player 0 is in red.

Figure 1 shows the exhaustive set of solutions in a Nim's game with four matches. The root of the tree represents the initial number of matches and each arrow represents the action of removing 1 (.....▸) or 2 (——▸) matches. We see that there is a winning strategy for player 0 if she starts. We are leveraging QBF to write this strategy in `TouIST`. The variable $takes\_2(i)$ is true if the current player takes 2 matches at step i and is false if she takes only one. If we denote by Φ the conjunction of formulas representing the rules of Nim's game then the existence of a winning strategy for player 0 is simply written:

$$\begin{aligned} &\exists takes\_2(0) \forall takes\_2(1) \\ &\quad \exists takes\_2(2) \forall takes\_2(3) \\ &\exists takes\_2(4)\ .\ (\neg 0\_lost \wedge \Phi) \end{aligned}$$

In words, we seek to satisfy the fact that there is an action of player 0 on turn 0 such that whatever the action of player 1 on turn 1, there exists an action of player 0 in turn 2, such that for any action of player 1 on the turn 3 there is an action of player 0 (who will therefore be the last to play) such that player 0 does not lose and that the constraints inherent in the Nim's game be satisfied.

The implementation of this formula in `TouIST` indicates that it is true, which means the existence of a winning strategy for the player 0. The solver returns the value of existential variables (here only one) corresponding to the next move of player 0. At this stage, the opposing player must provide his move which fixes the value of the universal variables corresponding to her possible next moves. The modified program is then executed again as follows (so as to take into

account the calculation of the valuation of *takes_2*(0)):

$$\begin{aligned}&\exists takes\_2(0) \exists takes\_2(1)\\ &\quad \exists takes\_2(2) \forall takes\_2(3)\\ &\exists takes\_2(4)\ .\ \neg 0\_lost \wedge c_0 \wedge c_1 \wedge \Phi\end{aligned}$$

where c_0 is either *takes_2*(0) or $\neg$*takes_2*(0) depending on the move of player 0, and similarly for c_1 depending on the move chosen by the opponent. The situation after these two moves is the new initial situation for the solver and the search for the player's next move 0 ... until she wins! This process is repeated until all the variables have received a value.

5.2. TouIST for planning as satisfiability

5.2.1. Solving Classical Planning Tasks

A planning task can be transformed into a propositional formula whose models correspond to solution plans (i.e., sequences or steps of actions starting from an initial state and leading to a goal). These models can be found using a SAT solver [5]. Numerous improvements of this approach have been proposed via the development of more compact and efficient encodings. We here illustrate the expressive power of the TouIST language by encoding of explanatory frame-axioms. If a fact is false at step $i-1$ of a solution plan and becomes true at step i then the disjunction of actions that can establish the fact (i.e. it is a positive effect of such an action) at step i of the plan is true. Indeed, at least one of the actions that can establish the fact must have been applied.

$$\bigwedge_{i\in\{1..PlanLength\}} \bigwedge_{f\in Facts} \left((\neg f(i-1) \wedge f(i)) \Rightarrow \bigvee_{a\in Actions | f\in Effects^+(a)} a(i) \right)$$

```
bigand $i in [1..$PlanLength]:
  bigand $f in $Facts:
    not $f($i-1) and $f($i) =>
      bigor $a in $Actions when $f in $Effects_pos($a):
        $a($i)
      end
  end
end
```

Much more compact QBF encodings have also been developed.

5.2.2. Solving Conformant/Temporal Planning Tasks

Beyond classic planning, TouIST allows us to encode and solve *conformant* planning tasks with QBF [9]. It can also be used to solve *temporal* planning tasks involving durative actions, exogenous events and temporally extended goals with SMT encodings [10, 11]. We here focus on the SMT encoding rules proposed in [8]. Below we give an encoding of temporal mutual exclusion of actions. If two actions a_1 and a_2, respectively producing a fact f (i.e. f is a positive effect of a_1) and its negation $\neg f$ (i.e. f is a negative effect of a_2), are active in the plan, then the time interval $[\tau^+_{start}(a_1, f), \tau^+_{end}(a_1, f)]$ corresponding to the activation of f by a_1 and the time interval $[\tau^-_{start}(a_2, f), \tau^-_{end}(a_2, f)]$ corresponding to the activation of $\neg f$ by a_2 are disjoint.

$$\bigwedge_{a_1\in Actions} \bigwedge_{a_2\in Actions} \bigwedge_{f\in Facts | f\in Effects^+(a_1)\cap Effects^-(a_2)}$$

$$\left((a_1 \wedge a_2) \Rightarrow \begin{pmatrix} \left(\tau^-_{end}(a_2, f) < \tau^+_{start}(a_1, f)\right) \\ \vee \left(\tau^+_{end}(a_1, f) < \tau^-_{start}(a_2, f)\right) \end{pmatrix}\right)$$

```
bigand $a1,$a2,$f in $Actions,$Actions,$Facts
when $f in $Effects_pos($a1)and $f in $Effects_neg($a2):
    $a1 and $a2 =>
      (t_end_del($a2,$f) < t_start_add($a1,$f))
      or (t_end_add($a1,$f) < t_start_del($a2,$f))
end
```

5.2.3. TouISTPlan **Module**

In order to tune and compare different logical encodings of planning tasks we have implemented the `TouISTPlan` module which automatically solves planning tasks with `TouIST`. For example, thanks to this module we compared the performance of different QBF encodings for reference planning problems from different International Planning Competitions (IPC) [4]. We were able to show that our new encodings are two times more efficient in terms of resolution time.

6. CONCLUSION

We have developed `TouIST` to offer a friendly language together with a modular tool that makes it easier to use SAT, SMT and QBF solvers. Indeed, `TouIST` can be seen as a compiler from extended and high-level logical languages to efficient independent solvers. These two sides give it great ease of use, a wide application spectrum and good computational performance. As such, it constitutes a completely original and unique tool of its kind.

We use it as part of the introductory course to logic of bachelor of mathematics and computer science, but also for the master's degree, as part of practical work and projects. Students are thus called upon to go through the entire process, from formalization to problem solving that goes far beyond toy problems that can be solved by hand.

But even more, `TouIST` is already used by researchers in the context of work carried out in our laboratory and involving logical modeling (planning, epistemic reasoning via translation into QBF, ...), it fills a lack existing in formal calculation software such as Maple, SageMath, Mathematica or Maxima which only anecdotally integrate logical tools.

In fact, `TouIST` may be useful in all research domains where finite modeling is relevant.

Its flexible and open structure will permit to use it in frontline research projects concerning, for example, extended resolution and its applications [3].

References

1. *Barwise J.* Scenes and other situations. Journal of Philosophy, vol. 78(7), pp. 369–397, 1981.
2. *Biere A.* Resolve and Expand. In Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04), pp. 59–70, Vancouver, BC, Canada, 2005.
3. *Buss B., Nordström J.* Proof Complexity and SAT Solving. Handbook of Satisfiability, second edition, Biere A., Heule M., van Maaren H. and Walsh T. Eds, 2021.
4. *Gasquet O., Longin D., Maris F., Régnier P., Valais M.* Compact Tree Encodings for Planning as QBF. Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial, vol. 21(62), pp. 103–114, 2018.
5. *Kautz H.A., Selman B.* Planning as Satisfiability. In Proceedings of the 10th European Conference on Artificial Intelligence, ECAI 92, pp. 359–363, Vienna, Austria, August 3-7, 1992.

6. *Kroening D., Strichman O.* Decision Procedures - An Algorithmic Point of View, Second Edition. Texts in Theoretical Computer Science. An EATCS Series, Springer, 2016.
7. *Luo Z., Soloviev S.* Dependent Event Types. *WoLLIC20, LNCS 10388*, 2017.
8. *Maris F., Régnier P.* TLP-GP: New Results on Temporally-Expressive Planning Benchmarks. In Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2008), vol. 1, pp. 507–514, November 3-5, 2008, Dayton, Ohio, USA, 2008.
9. *Rintanen J.* Asymptotically Optimal Encodings of Conformant Planning in QBF. In Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, pp. 1045–1050, July 22-26, Vancouver, British Columbia, Canada, 2007.
10. *Rintanen J.* Discretization of Temporal Models with Application to Planning with SMT. In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, pp. 3349–3355, January 25-30, 2015, Austin, Texas, USA., 2015.
11. *Shin J., Davis E.* Processes and continuous change in a SAT-based planner. Artificial Intelligence, vol. 166(2), pp. 194–253, 2005.
12. *Tseitin G.S.* On the Complexity of Derivation in Propositional Calculus. Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970, pp. 466–483, 1983.

additional info, some grant information, if needed

Received June 30, 2021, The final version: XXXXX XX, 2021