Issue 16
2025

Problem-Specific Visual Feedback in Discrete Modelling

Maurice Herwig
maurice.herwig@uni-kassel.de

Norbert Hundeshagen
hundeshagen@uni-kassel.de

Martin Lange
martin.lange@uni-kassel.de

Theoretical Computer Science / Formal Methods
University of Kassel
Germany

Abstract

Discrete modelling as the basis of problem solving is an essential skill for computer scientists, but the correct use of formal languages like propositional logic for such purposes remains a big challenge for undergraduate students. The DIMO tool provides support for the acquisition of formal modelling competencies using propositional logic. We extend the tool by generic capabilities to generate problem-specific feedback to students. This allows them to visualise the result of their modelling attempts in terms of the modelled problem at hand, thus helping students to initiate corresponding learning cycles.

Keywords: e-learning; propositional logic; teaching modelling; feedback systems; visual feedback; error-driven learning.

1. Introduction

Formal modelling essentially happens in many places in courses and textbooks on programming, computational complexity, formal languages and mathematical logic where specific formalisms are being presented for specific or general modelling tasks, e.g. programming languages, abstract machines, or logical formulas, cf. [1].

Here we are concerned with formal modelling in propositional logic (PL), a simple language that is widely used in computer science, from Boolean circuits [2] to planning in A.I. [3], computer-aided verification [4], cryptanalysis [5] and many more, mainly due to nowadays' access to highly efficient SAT solvers.

The ability to use such a fundamental modelling language like PL as a problem-solving tool is a standard competency to be acquired by computer science students. PL is therefore typically part of compulsory modules on formal logic or discrete mathematics, cf. [6]. The ability to use PL as a modelling language to tackle (real-world) problems typically exceeds the level of writing and evaluating single formulas. Instead, students must learn to read and construct parametrised *formula schemes*, see for instance the standard proof of NP-hardness of SAT which constructs a formula scheme $\varphi_{\mathcal{A},p,w}$ depending on a Turing machine $\mathcal{A}$, a polynomial p and a word w. Likewise, the fact that satisfiability for PL is a standard problem in NP can be used to show that other problems belong to NP as well, for instance the 3-Colourability problem, by constructing, given an undirected graph G, a polynomially sized formula φ_G that is satisfiable iff G is 3-colourable. The ability to correctly read, write and evaluate such formula schemes introduces a whole new level of difficulty for students.

While there are numerous high-quality learning tools designed to develop basic PL competencies such as writing, evaluating, and normalising single formulas, as for instance discussed in an older overview [7], and demonstrated by recent tools [8], these often fall short when addressing application-oriented formal modelling as described above. The latter enjoys rudimentary support only, for instances through logical quizzes, with very few notable exceptions, cf. [9] [10].

We have developed DIMO [11] for that purpose. This learning tool essentially acts like an interpreter, allowing students to check their formula schemes by enumerating parameters and testing instances for satisfiability successively. DIMO, in its first version, provides generic and problem-*unspecific* output in the form of evaluations of propositional variables. To use this for checking correctness of a formula scheme, students must link these evaluations to instances of the real-world problem at hand, which is error-prone and in fact part of the modelling skill to be learnt in the first place.

This paper extends a preliminary version [12] and reports on extending DIMO's capabilities to provide *problem-specific* output that enhances the learning of formal modelling by enabling feedback in terms of the modelled problem rather than the formal language of PL to be learned. Problem-specific output is introduced into DIMO via a simple domain-specific language (DSL) whose programs turn propositional evaluations into arbitrary (graphical) output. Students do *not* need to learn this DSL; it allows teachers to equip their exercises in the background with a small imperative program that visualises the students' modelling effects.

2. Modelling in Propositional Logic – A Didactical Perspective

In general, a PL modelling task is typically an instance of the following scheme.

given:	*a (real-world) problem A whose instances $A(\overline{p})$ depend on parameters $\overline{p}$ and are either solvable or unsolvable*
task:	*construct a propositional formula (scheme) $\Phi(\overline{p})$ that is satisfiable if and only if $A(\overline{p})$ is solvable*

For instance, in the example above, A is the Queens Problem, the only parameter is n, and $A(2), A(3)$ are unsolvable whereas $A(1)$ and $A(n)$ for $n \geq 4$ are known to be solvable. Logical problems other than satisfiability (validity, equivalence, model counting, etc.) can of course also serve as target problems (and are supported by DIMO, too).

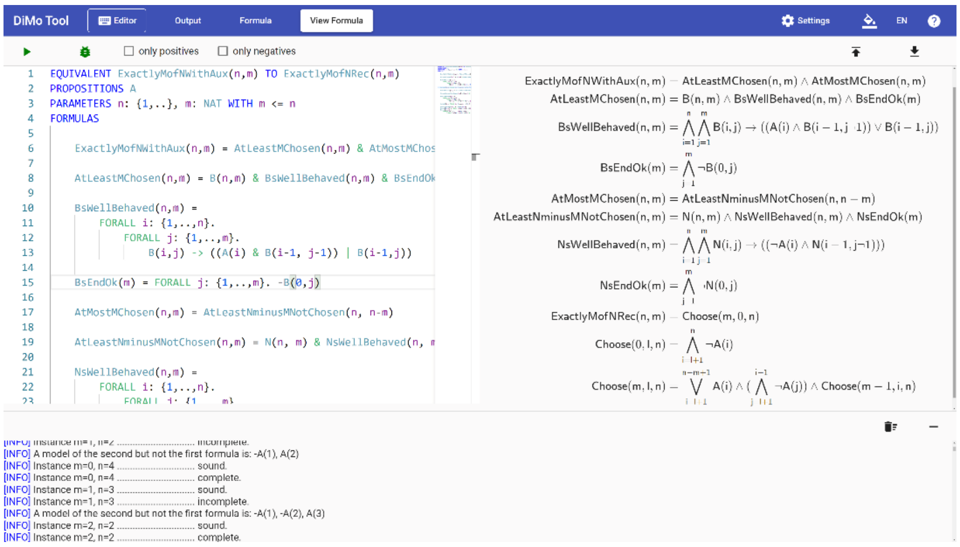
This generalisation offers two insights into difficulties that students have with solving such exercises. First of all, real-world problems typically depend on domain-specific parameters that influence the solution of such problems, and so do the constructed PL formulas. Students need to deal with an additional syntactical dimension on top of the language of PL. For example, a solution to the modelling task for the Queens Problem using propositions $D_{i,j}$ for "*a queen is placed on cell* (i,j)" typically is a formula scheme of the form:

$$\Phi(n) \ = \ \ldots \wedge \left(\bigwedge_{i=1}^{n-1} \bigwedge_{j=1}^{n} \bigwedge_{i'=i+1}^{n} D_{i,j} \rightarrow \neg D_{i',j} \right) \wedge \ldots$$

The ability to recognize that n is a parameter to this formula whilst i, j and i' are meta-variables used to describe the formula depending on n is not something that students master easily. The distinction between parameters and propositions is another source for confusion, and so is a sensible choice of a suitable set of atomic propositions depending on the parameters. These challenges comprise the syntactic level of learning formal modelling.

Secondly – and which makes up the semantical level of this learning challenge – students need to verify that their model is correct, i.e. Φ_n is satisfiable iff the instance of the Queens Problem for parameter n has a solution. This is typically done by letting a propositional evaluation encode a problem instance, here a particular placement of the queens. This requires a deep understanding of the role of different parts of the formula w.r.t. the problem at hand. The difficulties with the latter can best be described by the theory of Cognitive Dimensions of Notations, which discusses the accessibility of a (programming) language via the effect of syntactical changes on the semantics, cf. [13] [14]. Despite its syntactical simplicity, PL poses difficulties as a modelling formalism because of exactly this property: small syntactical changes in a formula can have large impacts on its semantics. The art of learning formal modelling is to start "seeing" how the semantics of a formula changes under variations in the parameters rather than to only know the meanings of the symbols in the formula (scheme) in isolation.

The learning tool DIMO addresses challenges on both the syntactical and the semantical level. Already the first version of DIMO featured a feedback system known from IDEs for pointing out errors in programming [11], using syntax highlighting and autocompletion to separate different syntactical elements in what may first appear to be a tangled mass of symbols to a learning beginner, as shown in Figure 1.



```
EQUIVALENT ExactlyMofNWithAux(n,m) TO ExactlyMofNRec(n,m)
PROPOSITIONS A
PARAMETERS n: {1,..}, m: NAT WITH m <= n
FORMULAS

    ExactlyMofNWithAux(n,m) = AtLeastMChosen(n,m) & AtMostMChos

    AtLeastMChosen(n,m) = B(n,m) & BsWellBehaved(n,m) & BsEndOk

    BsWellBehaved(n,m) =
        FORALL i: {1,..,n}.
            FORALL j: {1,..,m}.
                B(i,j) -> ((A(i) & B(i-1, j-1)) | B(i-1,j))

    BsEndOk(m) = FORALL j: {1,..,m}. -B(0,j)

    AtMostMChosen(n,m) = AtLeastNminusMNotChosen(n, n-m)

    AtLeastNminusMNotChosen(n,m) = N(n, m) & NsWellBehaved(n, m

    NsWellBehaved(n,m) =
        FORALL i: {1,..,n}.
            FORALL j: {1,..,m}
```

$$\mathrm{ExactlyMofNWithAux}(n,m) = \mathrm{AtLeastMChosen}(n,m) \wedge \mathrm{AtMostMChosen}(n,m)$$
$$\mathrm{AtLeastMChosen}(n,m) = B(n,m) \wedge \mathrm{BsWellBehaved}(n,m) \wedge \mathrm{BsEndOk}(m)$$
$$\mathrm{BsWellBehaved}(n,m) = \bigwedge_{i=1}^{n}\bigwedge_{j=1}^{m} B(i,j) \rightarrow ((A(i) \wedge B(i-1,j-1)) \vee B(i-1,j))$$
$$\mathrm{BsEndOk}(m) = \bigwedge_{j=1}^{m} \neg B(0,j)$$
$$\mathrm{AtMostMChosen}(n,m) = \mathrm{AtLeastNminusMNotChosen}(n,n-m)$$
$$\mathrm{AtLeastNminusMNotChosen}(n,m) = N(n,m) \wedge \mathrm{NsWellBehaved}(n,m) \wedge \mathrm{NsEndOk}(m)$$
$$\mathrm{NsWellBehaved}(n,m) = \bigwedge_{i=1}^{n}\bigwedge_{j=1}^{m} N(i,j) \rightarrow ((\neg A(i) \wedge N(i-1,j-1)))$$
$$\mathrm{NsEndOk}(m) = \bigwedge_{j=1}^{m} \neg N(0,j)$$
$$\mathrm{ExactlyMofNRec}(n,m) = \mathrm{Choose}(m,0,n)$$
$$\mathrm{Choose}(0,l,n) = \bigwedge_{i=l+1}^{n} \neg A(i)$$
$$\mathrm{Choose}(m,l,n) = \bigvee_{i=l+1}^{n-m+1} A(i) \wedge \Big(\bigwedge_{j=l+1}^{i-1} \neg A(j)\Big) \wedge \mathrm{Choose}(m-1,i,n)$$

```
[INFO] Instance m=1, n=2 .............................. incomplete.
[INFO] A model of the second but not the first formula is: -A(1), A(2)
[INFO] Instance m=0, n=4 .............................. sound.
[INFO] Instance m=0, n=4 .............................. complete.
[INFO] Instance m=1, n=3 .............................. sound.
[INFO] Instance m=1, n=3 .............................. incomplete.
[INFO] A model of the second but not the first formula is: -A(1), -A(2), A(3)
[INFO] Instance m=2, n=2 .............................. sound.
[INFO] Instance m=2, n=2 .............................. complete.
```

Figure 1. The interface of DiMo's previous version with syntax highlighting and human readable formula representation.

The provision of (tool-)assistance for the semantical level is the focus here. As this concerns the ability to validate the quality of a model (here: the formula scheme), which forms the core competency of modelling in general, we incorporate ideas from corresponding theories targeting related areas in order to enhance DIMO's learning support capabilities. For instance, issues and strategies in learning and teaching modelling are well studied in the field of STEM, cf. [15] [16]. The former describes the modelling process as a cycle that involves a repeated comparison of the model with the real-world problem; more precisely, the validation of a model-based solution in the real world is one of the major difficulties for students and, as pointed out above, one of the most important steps in modelling in PL. Admittedly, the data for the latter results is based on studies in school mathematics. We believe, based on own experience and the generality of the competencies needed for formal modelling, that these results carry over to the undergraduate setting in computer science. Thus, to support learning in this area the requirements for such a tool are "maximal students independence", cf. [15] p. 54, and implied "strategic interventions" aimed at the validation step, cf. [15], p. 52. Typical teachers' interventions for the latter are questions that encourage students to visualise their model w.r.t. the real-world problem. This plays a key role in understanding formal principles, cf. [17].

In PL the latter can be done generically by, for instance, a SAT solver that textually "visualises" the satisfiability of a formula in form of a satisfying variable assignment, as it is implemented in DIMO's first version. In the following we present a more involved visualisation concept by means of depicting the semantics of a formula in terms of the real-world problem, thus helping students to initiate the modelling cycle mentioned above.

3. Extending DiMo

3.1 DIMO’s Main Specs.

We only provide a very brief description of DIMO’s essential architectural structure und features. From a software architecture point of view, DIMO is a classical web-application with a backend implemented mainly in OCaml (Objective Caml) and a TypeScript/Angular frontend. Formula schemes are written in the so-called DIMO-language. A complete tool description is given in [11].

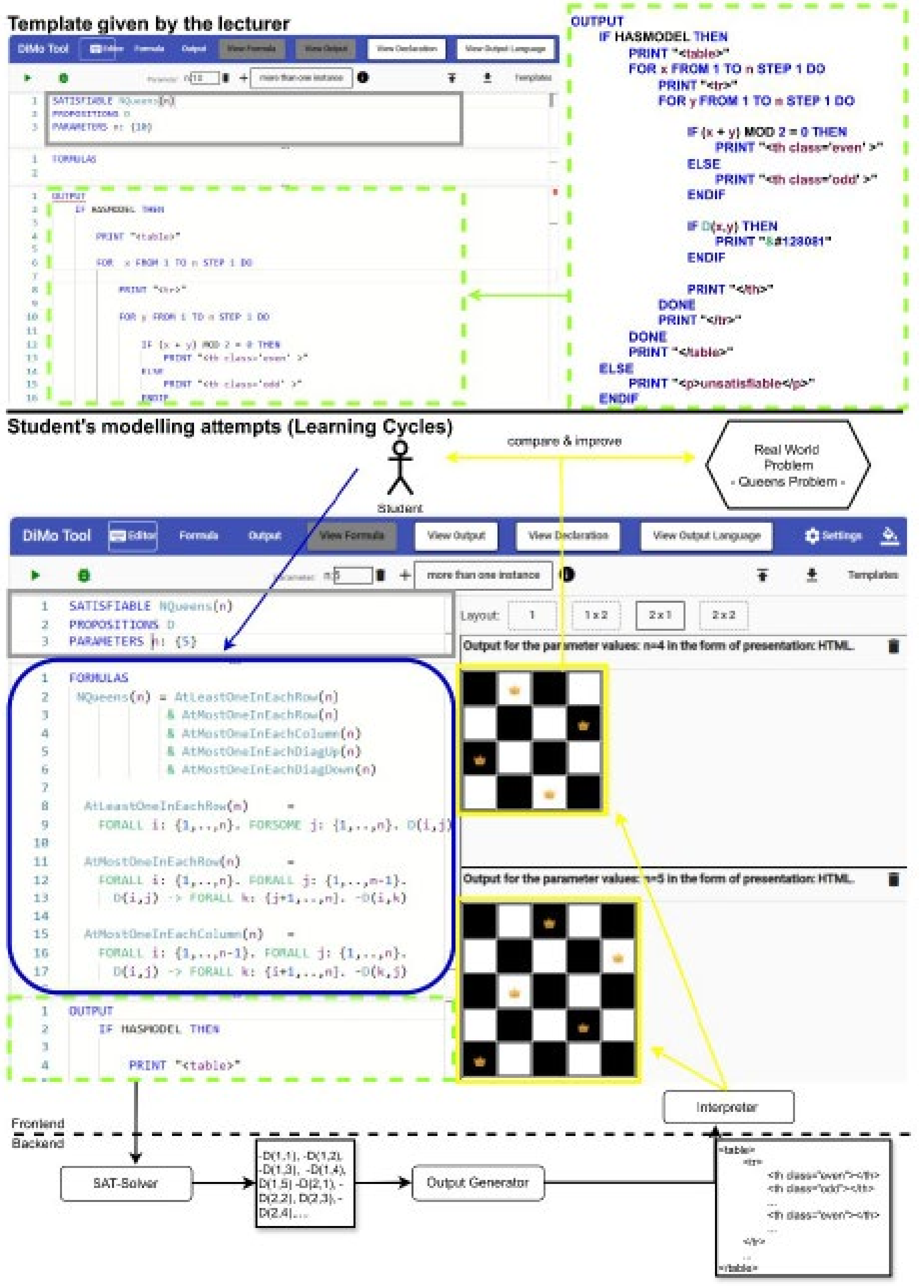
3.2 DIMO’s Novel Extension: Problem-Specific Visualisation of PL Models.

The main idea leading to the ability to generate problem-specific visualisations – instead of plain propositional models – is based on an extension of DIMO’s language by small imperative programs that can iterate over propositional models and print information in some interpretable output format. The green dotted box in Figure 2. marks an example program that is used to produce graphical output for the Queens Problem from above.

The output is generated using for-loops over the domain of parameters (cf. lines 4 and 6 of the green dotted box), conditional statements over Boolean expressions, in particular, arithmetic relations (line 7) and truth values of atomic PL variables (line 12). The evaluation of such programs is done in three-valued logic, as the output program may refer to PL variables that do not have truth values, for instance because they did not occur in the formula scheme created by a student. The core of the output language is a simple print-statement that allows arbitrary strings to be created (line 5 or 13).

This technically simple extension enables the use of different follow-on interpreters for the strings created by DIMO’s output program for various instances of the formula scheme. The lower part of Figure 2 shows how problem-specific visualisation can be given for the Queens Problem Q with parameter n: (I) the formula scheme $\Phi_Q(n)$ (blue box with rounded corners) is instantiated in the background for some parameter value like $n = 5$, (II) a SAT solver is used to compute a satisfying variable assignment ϑ for $Q(5)$ if there is some, (III) the output program (green dotted box) is run with the same parameter evaluation $n = 5$ and ϑ to create, e.g., HTML code which (IV) is used to display the model in the foreground as feedback about the correctness of the formula scheme (yellow solid box).

DIMO currently supports two more output types besides HTML: plain text and VisSat, an internal development aimed at easing the visualisation of matrix-like structures. Figure 3 shows examples of feedback given in different output types for different problems: HTML for the well-known River Crossing Puzzle, plain text for the Queens Problem, and VisSat-output for the problem of finding a knight’s tour on a chess board.


Template given by the lecturer
Student's modelling attempts (Learning Cycles)
Student
compare & improve
Real World Problem - Queens Problem -
DiMo Tool
Editor
Formula
Output
View Formula
View Output
View Declaration
View Output Language
Settings
more than one instance
Templates
Layout: 1 1 x 2 2 x 1 2 x 2
Output for the parameter values: n=4 in the form of presentation: HTML.
Output for the parameter values: n=5 in the form of presentation: HTML.
Interpreter
Frontend
Backend
SAT-Solver
Output Generator

```
OUTPUT
    IF HASMODEL THEN
        PRINT "<table>"
        FOR x FROM 1 TO n STEP 1 DO
            PRINT "<tr>"
            FOR y FROM 1 TO n STEP 1 DO

                IF (x + y) MOD 2 = 0 THEN
                    PRINT "<th class='even' >"
                ELSE
                    PRINT "<th class='odd' >"
                ENDIF

                IF D(x,y) THEN
                    PRINT "&#128081"
                ENDIF

                PRINT "</th>"
            DONE
            PRINT "</tr>"
        DONE
        PRINT "</table>"
    ELSE
        PRINT "<p>unsatisfiable</p>"
    ENDIF
```

```
SATISFIABLE NQueens(n)
PROPOSITIONS D
PARAMETERS n: {5}
```

```
FORMULAS
 NQueens(n) = AtLeastOneInEachRow(n)
              & AtMostOneInEachRow(n)
              & AtMostOneInEachColumn(n)
              & AtMostOneInEachDiagUp(n)
              & AtMostOneInEachDiagDown(n)

  AtLeastOneInEachRow(n)     =
    FORALL i: {1,..,n}. FORSOME j: {1,..,n}. D(i,j)

  AtMostOneInEachRow(n)      =
    FORALL i: {1,..,n}. FORALL j: {1,..,n-1}.
      D(i,j) -> FORALL k: {j+1,..,n}. -D(i,k)

  AtMostOneInEachColumn(n)   =
    FORALL i: {1,..,n-1}. FORALL j: {1,..,n}.
      D(i,j) -> FORALL k: {i+1,..,n}. -D(k,j)
```

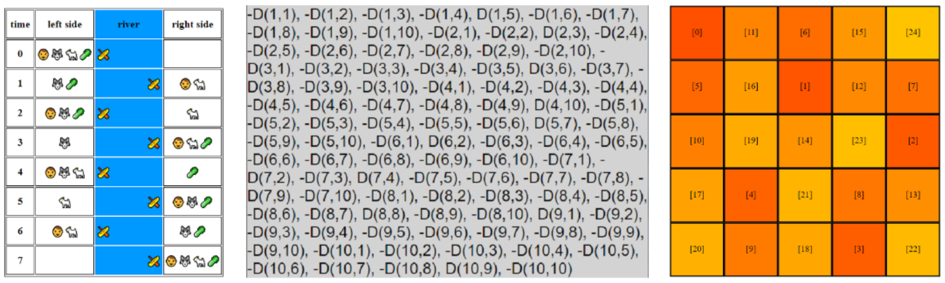
```
OUTPUT
    IF HASMODEL THEN

        PRINT "<table>"
```

```
-D(1,1), -D(1,2),
-D(1,3), -D(1,4),
D(1,5) -D(2,1), -
D(2,2), D(2,3), -
D(2,4),...
```

```
<table>
    <tr>
        <th class="even"></th>
        <th class="odd"></th>
        ...
        <th class="even"></th>
        ...
    </tr>
    ...
</table>
```

Figure 2. How to use DIMO: (top) lecturer’s creation of templates as parameter declarations etc. (grey box) and output programme (green dotted box). (Middle) Students’ attempts at modelling by writing formulas in the formula scheme editor (blue box with rounded corners) and receiving graphical feedback for different instances (yellow solid boxes), here n=4 and n=5. Both are easily seen to be valid solutions. (Bottom) DIMO’s technical process for creating graphical feedback.



time	left side	river	right side
0	👦🐺🐐🥒	🛶	
1	🐺🥒	🛶	👦🐐
2	👦🐺🥒	🛶	🐐
3	🐺	🛶	👦🐐🥒
4	👦🐺🐐	🛶	🥒
5	🐐	🛶	👦🐺🥒
6	👦🐐	🛶	🐺🥒
7		🛶	👦🐺🐐🥒

```
-D(1,1), -D(1,2), -D(1,3), -D(1,4), D(1,5), -D(1,6), -D(1,7),
-D(1,8), -D(1,9), -D(1,10), -D(2,1), -D(2,2), D(2,3), -D(2,4),
-D(2,5), -D(2,6), -D(2,7), -D(2,8), -D(2,9), -D(2,10), -
D(3,1), -D(3,2), -D(3,3), -D(3,4), -D(3,5), D(3,6), -D(3,7), -
D(3,8), -D(3,9), -D(3,10), -D(4,1), -D(4,2), -D(4,3), -D(4,4),
-D(4,5), -D(4,6), -D(4,7), -D(4,8), -D(4,9), D(4,10), -D(5,1),
-D(5,2), -D(5,3), -D(5,4), -D(5,5), -D(5,6), D(5,7), -D(5,8),
-D(5,9), -D(5,10), -D(6,1), D(6,2), -D(6,3), -D(6,4), -D(6,5),
-D(6,6), -D(6,7), -D(6,8), -D(6,9), -D(6,10), -D(7,1), -
D(7,2), -D(7,3), D(7,4), -D(7,5), -D(7,6), -D(7,7), -D(7,8), -
D(7,9), -D(7,10), -D(8,1), -D(8,2), -D(8,3), -D(8,4), -D(8,5),
-D(8,6), -D(8,7), D(8,8), -D(8,9), -D(8,10), D(9,1), -D(9,2),
-D(9,3), -D(9,4), -D(9,5), -D(9,6), -D(9,7), -D(9,8), -D(9,9),
-D(9,10), -D(10,1), -D(10,2), -D(10,3), -D(10,4), -D(10,5),
-D(10,6), -D(10,7), -D(10,8), D(10,9), -D(10,10)
```

[0]	[11]	[6]	[15]	[24]
[5]	[16]	[1]	[12]	[7]
[10]	[19]	[14]	[23]	[2]
[17]	[4]	[21]	[8]	[13]
[20]	[9]	[18]	[3]	[22]

Figure 3. Examples for displaying feedback using three different output types: HTML, plain text and VisSat.

HTML output is particularly useful for modelled problems that come with a straight-forward graphical view, like the Queens Problem. VisSat offers a simpler format than HTML output, whenever the view is based on a representation of matrices or graphs, in order to avoid hard-coding tables using tr- and td-tags. Plain text can be used to tweak the difficulty of the learning process, for instance by not providing a direct visualisation but requiring students to form this visualisation themselves. From a technical point of view, adding new interpreters for output languages to DIMO is fairly simple, only requiring their integration as an Angular component.

4. Usage – Examples and Benefits

DIMO can be used as a self-learning tool or integrated into a course teaching formal modelling with PL. It offers a concise language to provide the audience with reasonable complex examples of PL formula schemes and live demonstrations of the effect of different modelling attempts on the problem at hand.

Usage Examples

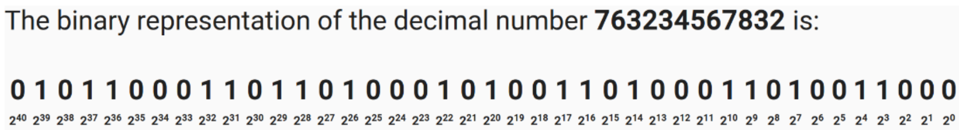
Even though implementing new visualisations in DIMO's output language is reasonably easy, we recommend the use of DIMO's template system, which allows instructors to provide a set of given output programs for modelling problems, such as the program shown at the top of Figure 2.

In a typical usage scenario, students would be given a modelling exercise and be required to use DIMO to create and verify their solutions using the provided output program. The following two examples demonstrate this approach.

Example 1

PL can be used to transform numbers in decimal format into binary format. This does not purely use the satisfiability problem for PL; a satisfying assignment is not just seen as a witness for satisfiability but instead as the solution to an exercise.

Exercise *(Building a Binary Encoder): Let n be a natural number. Construct a propositional formula $\Phi(n)$ over the variables $B_0, B_1, \ldots, B_{\lceil log(n) \rceil}$, such that the interpretation of B_i in every model of $\Phi(n)$ is the i-th bit of the binary encoding of n, where B_0 is the most significant and $B_{\lceil log(n) \rceil}$ the least significant bit.*



The binary representation of the decimal number **763234567832** is:

0 1 0 1 1 0 0 0 1 1 0 1 1 0 1 0 0 0 1 0 1 0 0 1 1 0 1 0 0 0 1 1 0 1 0 0 1 1 0 0 0

2^{40} 2^{39} 2^{38} 2^{37} 2^{36} 2^{35} 2^{34} 2^{33} 2^{32} 2^{31} 2^{30} 2^{29} 2^{28} 2^{27} 2^{26} 2^{25} 2^{24} 2^{23} 2^{22} 2^{21} 2^{20} 2^{19} 2^{18} 2^{17} 2^{16} 2^{15} 2^{14} 2^{13} 2^{12} 2^{11} 2^{10} 2^{9} 2^{8} 2^{7} 2^{6} 2^{5} 2^{4} 2^{3} 2^{2} 2^{1} 2^{0}

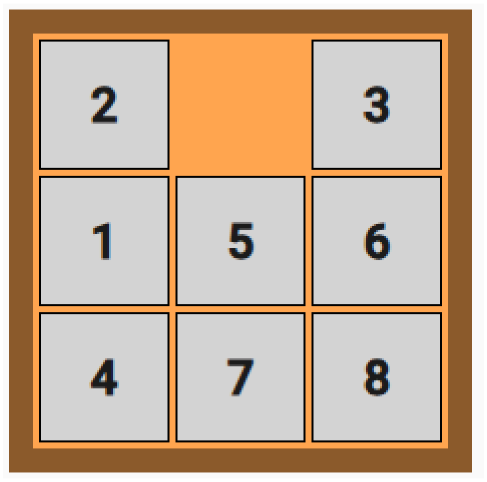
Figure 4. Graphical output for the binary encoder.

The above exercise contains the names of the propositional variables for the reason that the output program needs to access their values to correctly compute the visualisation. This can be seen in line 7 of the code in Figure 6, which is provided as a template with the above exercise. Note that Figure 6 is an example of an HTML output. A correct solution then leads to the graphical output shown in Figure 4.

Example 2

We consider the 8-Puzzle problem, a smaller variant of the perhaps better known 15-Puzzle problem.

Exercise *(Solving the Eight-Puzzle): Given some instance of the 8-Puzzle problem like the following one.*



Use propositional logic to determine the number of steps, denoted as n∈N, required to solve the instance above. For that, construct a propositional formula $\Phi(n)$ which is satisfiable if and only if a solution exists for this puzzle that takes exactly n steps. Use propositional variables $F(i,j,k,m)$ with the interpretation that at step m, the number k occupies cell (i,j) on the 8-Puzzle grid.

As a template for the latter exercise an output program using HTML can be given as shown in Figure 7. A correct solution then leads to the visualisation presented in Figure 5. Note that the examples above require the PL variables used to model the problem to be given in the exercise, as the output program makes use of them. This has pros and cons. Picking out suitable variables is essential for formal PL modelling, and this part of the task is taken out of the students' hands. On the other hand, when the propositional variables are known, DIMO can also be used to semi-automatically verify the quality of a solution by checking it for equivalence against a master solution.

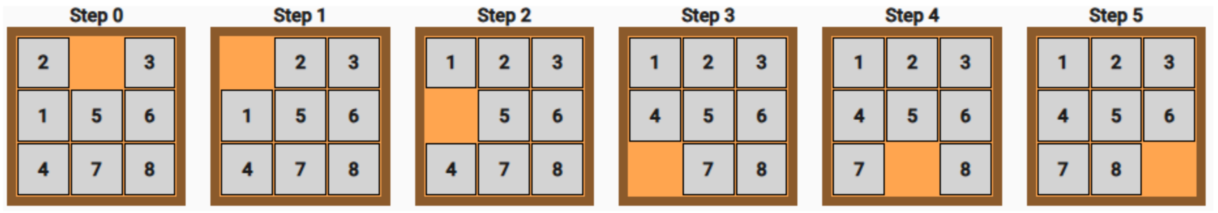


Figure 5. Graphical output for the 8-puzzle.

```
OUTPUT
    IF HASMODEL THEN
       PRINTF "<p>The binary representation of the"
               decimal number <b>%i </b> is: </p><br>" (n)
       PRINT "<table><tr>"
       FOR i FROM LOG  MAX {1,n} TO 0 STEP -1 DO
          IF B(i) THEN
             PRINT "<th>1</th>"
          ELSE
             PRINT "<th>0</th>"
          ENDIF
       DONE

       PRINT "</tr><tr>"

       FOR i FROM LOG  MAX {1,n} TO 0 STEP -1 DO
        PRINTF "<th class='binary'>2<sup>%i</sup></th>  (i)
       DONE

       PRINT "</tr></table>"
    ELSE
       PRINT "<p>unsatisfiable</p>"
    ENDIF

  AS-TYPE HTML FRONT
  *{
     font-size: 18px;
     margin:0;
  }
  .binary{
    font-size: 8px;
  }
  sup{
    font-size: 6px;
  }
```

Figure 6. Output program for the binary encoder.

```
  OUTPUT
  IF HASMODEL THEN
    PRINT "<table>"
    FOR k FROM 0 TO n DO
      PRINT "<th class='step'>"
      PRINT "<table class='frame'>"
      PRINTF "Step %i " (k)
      FOR i FROM 1 TO 3 DO
        PRINT "<tr>"
        FOR j FROM 1 TO 3 DO
          IF F(i, j, 0, k) THEN
            PRINT "<th> </th>"
          ELSE
            FOR x FROM 1 TO 8 DO
              IF F(i, j, x, k) THEN
                PRINTF "<th class='cell'> %i </th>" (x)
              ENDIF
            DONE
          ENDIF
        DONE
        PRINT "</tr>"
      DONE
      PRINT "</table>"
      PRINT "</th>"
    DONE
    PRINT "</table>"
  ELSE
    PRINT "unsatisfiable"
  UNDEF
    PRINT "undefined"
  ENDIF

AS-TYPE HTML FRONT
.frame{
 border: 8px solid #8B5A2B;
 background: #FFA54F;
}
.step{
 padding: 10pxL
}
.cell{
 border: 1px solid black;
 background: #D3D3D3;
 min-width: 40px;
 height: 40px;
}
```

Figure 7. Output program for the 8-puzzle.

Preliminary Evaluation

The extension of DIMO presented in this paper has not yet been implemented in a wider course setting. However, to evaluate the graphical-feedback system, we conducted a small trial involving nine undergraduate and graduate students. All participants had successfully completed the bachelor's course on (formal languages and) logic, which includes propositional modelling as a component.

The participants were provided with two erroneous formula schemes for two similar one-player-games: Solitaire and the 8-Puzzle, shown in Figure 10 and Figure 8. For the 8-Puzzle the formula scheme claimed to model the existence of a solution taking n steps. The Solitaire formula scheme also (erroneously) modelled a sequence of n steps according to the game's rules. For either game, DIMO's output was presented in textual form and in visualised form. Figure 9 exemplarily shows the textual output for the 8-puzzle modelling attempt and Figure 11 the visualisation for Solitaire. A two-level task was given with a 45 minutes time limit: first, find the mistake in the given propositional models; second, suggest how to fix it in the formula scheme. Finally, two questions were asked about the perceived difficulty of the given exercises and the helpfulness of the presentation form.

The main result taken from the latter question can be subsumed by the following quote from the answers: "*The error [in the propositional model] was obvious in the graphical representation. The more information is given in textual form, the more difficult it gets to find the error.*" Seven out of nine students answered the question accordingly, while one student found the textual representation more helpful. Moreover, some sparse data on the time spent to actually "see" the error in the representation suggests a huge impact on the time needed to validate a given model. The latter cannot be explained by a focus effect (mistakes found in the first task lead to the mistakes in the second task), as the study group was divided into two halves that received the games in different orders.

Only one student was able to correct the formula for Solitaire in the given time. Note that the erroneous formula schemes were given to them and it is commonly known that it is more difficult to repair someone else's code than one's own [18]. Furthermore, some observations made during the study also suggest that the actual form of visualisation has an effect on the time spent to pinpoint the mistake. The visualisation that four participants received for Solitaire was based on a picture that just contains the actual board of the game, whereas the other five were given a visualisation of all propositional variables present in the formula as shown in step 1 of Figure 11, i.e. including the outer edge. No student of the first group had the correct idea on where the mistake might be found, whereas three students from the second group made suggestions on how to correct a formula.

5. Conclusion

Based on the theory of modelling developed in the field of didactics of mathematics, we have extended DIMO, a learning tool aimed at supporting students in acquiring modelling competencies in the area of propositional logic, with the ability to graphically visualise solutions derived by propositional models for real-world problems.

A preliminary evaluation suggests that students perform better in finding errors in models, when provided with a suitable visualisation. Thus, this improvement suggests that such visual feedback is actually helpful in supporting the validation step in the modelling cycle according to Blum and Ferri [15].

Further extensions of DIMO are planned. Currently the DIMO-language only supports natural numbers as parameters which restricts possible modelling exercises. A future version will include more complex data types like strings or graphs. This will allow modelling tasks to be taken from other areas of application, such as string or routing problems (e.g. shortest common subsequence, Hamiltonian path in graphs, etc.).

$$\mathsf{EightPuzzle}(n) = \mathsf{Start} \wedge \mathsf{End}(n) \wedge \mathsf{Step}(n)$$

$$\begin{aligned}\mathsf{Start} = {} & F(1,1,2,0) \wedge F(1,2,0,0) \wedge F(1,3,3,0) \wedge F(2,1,1,0) \\ & \wedge F(2,2,5,0) \wedge F(2,3,6,0) \wedge F(3,1,4,0) \wedge F(3,2,7,0) \wedge F(3,3,8,0)\end{aligned}$$

$$\begin{aligned}\mathsf{End}(n) &= \bigvee_{k=0}^{n} F(1,1,1,k) \\ \mathsf{IsNull}(0) &= \mathsf{True} \\ \mathsf{IsNull}(z) &= \mathsf{False}\end{aligned}$$

$$\mathsf{Step}(n) = \bigwedge_{m=0}^{n-1} \mathsf{RightStep}(m) \vee \mathsf{LeftStep}(m) \vee \mathsf{UpStep}(m) \vee \mathsf{DownStep}(m)$$

$$\begin{aligned}\mathsf{RightStep}(m) = {} & \bigvee_{i=1}^{3} \bigvee_{j=1,}^{} \bigvee_{x=1}^{8} F(i,j,0,m) \wedge F(i,j+1,x,m) \wedge F(i,j,x,m+1) \wedge F(i,j+1,0,m+1) \\ & \wedge \bigwedge_{i'=1}^{3} \bigwedge_{j'=1}^{3} \bigwedge_{x'=\{0,\dots,8\}\setminus\{x\}} \mathsf{IsDifferent}(i',j',i,j) \wedge F(i',j',x',m) \rightarrow F(i',j',x',m+1)\end{aligned}$$

$$\begin{aligned}\mathsf{LeftStep}(m) = {} & \bigvee_{i=1}^{3} \bigvee_{j=2,}^{} \bigvee_{x=1}^{8} F(i,j,0,m) \wedge F(i,j-1,x,m) \wedge F(i,j,x,m+1) \wedge F(i,j-1,0,m+1) \\ & \wedge \bigwedge_{i'=1}^{3} \bigwedge_{j'=1}^{3} \bigwedge_{x'=\{0,\dots,8\}\setminus\{x\}} \mathsf{IsDifferent}(i',j',i,j) \wedge F(i',j',x',m) \rightarrow F(i',j',x',m+1)\end{aligned}$$

$$\begin{aligned}\mathsf{UpStep}(m) = {} & \bigvee_{i=2,}^{} \bigvee_{j=1}^{3} \bigvee_{x=1}^{8} F(i,j,0,m) \wedge F(i-1,j,x,m) \wedge F(i,j,x,m+1) \wedge F(i-1,j,0,m+1) \\ & \wedge \bigwedge_{i'=1}^{3} \bigwedge_{j'=1}^{3} \bigwedge_{x'=\{0,\dots,8\}\setminus\{x\}} \mathsf{IsDifferent}(i',j',i,j) \wedge F(i',j',x',m) \rightarrow F(i',j',x',m+1)\end{aligned}$$

$$\mathsf{DownStep}(m) = \bigvee_{i=1,}^{} \bigvee_{j=1}^{3} \bigvee_{x=1}^{8} F(i,j,0,m) \wedge F(i+1,j,x,m) \wedge F(i,j,x,m+1) \wedge F(i+1,j,0,m+1)$$

$3 \quad 3$

Figure 8. Erroneous modelling of the 8-puzzle problem

F(1,1,0,2), F(1,1,0,3), F(1,1,0,4), F(1,1,0,5), F(1,1,1,2), F(1,1,2,0), F(1,1,8,1), F(1,2,0,0), F(1,2,0,1), F(1,2,2,2), F(1,2,2,3), F(1,2,2,4), F(1,2,2,5), F(1,2,8,2), F(1,2,8,3), F(1,2,8,4), F(1,2,8,5), F(1,3,0,3), F(1,3,1,4), F(1,3,1,5), F(1,3,3,0), F(1,3,3,1), F(1,3,3,2), F(1,3,3,3), F(2,1,1,0), F(2,1,1,1), F(2,1,1,2), F(2,1,2,0), F(2,1,4,2), F(2,1,4,3), F(2,1,4,4), F(2,1,4,5), F(2,2,1,2), F(2,2,5,0), F(2,2,5,1), F(2,2,5,2), F(2,2,5,3), F(2,2,5,4), F(2,2,5,5), F(2,2,8,1), F(2,3,0,3), F(2,3,0,4), F(2,3,1,2), F(2,3,1,3), F(2,3,3,4), F(2,3,6,0), F(2,3,6,1), F(2,3,6,2), F(2,3,6,3), F(2,3,6,4), F(2,3,7,5), F(2,3,8,1), F(3,1,0,1), F(3,1,0,2), F(3,1,0,3), F(3,1,0,4), F(3,1,0,5), F(3,1,2,0), F(3,1,4,0), F(3,1,4,1), F(3,1,4,2), F(3,1,4,3), F(3,1,4,4), F(3,1,4,5), F(3,1,7,2), F(3,1,7,3), F(3,1,7,4), F(3,2,0,0), F(3,2,1,0), F(3,2,1,2), F(3,2,2,1), F(3,2,2,2), F(3,2,2,3), F(3,2,2,4), F(3,2,2,5), F(3,2,3,0), F(3,2,7,0), F(3,2,8,2), F(3,2,8,3), F(3,2,8,4), F(3,2,8,5), F(3,3,0,2), F(3,3,0,5), F(3,3,1,3), F(3,3,2,0), F(3,3,2,2), F(3,3,7,4), F(3,3,8,0), F(3,3,8,1)

Figure 9. Feedback in problem-unspecific form for the 8-puzzle problem.

$$\begin{aligned}
\mathrm{Solitaire}(n) &= \mathrm{Start} \wedge \mathrm{Step}(n) \wedge \mathrm{Positions}\\
\mathrm{Positions} &= \neg P(1,1) \wedge \neg P(1,2) \wedge P(1,3) \wedge P(1,4) \wedge P(1,5) \wedge \neg P(1,6) \wedge \neg P(1,7)\\
&\wedge \neg P(2,1) \wedge \neg P(2,2) \wedge P(2,3) \wedge P(2,4) \wedge P(2,5) \wedge \neg P(2,6) \wedge \neg P(2,7)\\
&\wedge P(3,1) \wedge P(3,2) \wedge P(3,3) \wedge P(3,4) \wedge P(3,5) \wedge P(3,6) \wedge P(3,7)\\
&\wedge P(4,1) \wedge P(4,2) \wedge P(4,3) \wedge P(4,4) \wedge P(4,5) \wedge P(4,6) \wedge P(4,7)\\
&\wedge P(5,1) \wedge P(5,2) \wedge P(5,3) \wedge P(5,4) \wedge P(5,5) \wedge P(5,6) \wedge P(5,7)\\
&\wedge \neg P(6,1) \wedge \neg P(6,2) \wedge P(6,3) \wedge P(6,4) \wedge P(6,5) \wedge \neg P(6,6) \wedge \neg P(6,7)\\
&\wedge \neg P(7,1) \wedge \neg P(7,2) \wedge P(7,3) \wedge P(7,4) \wedge P(7,5) \wedge \neg P(7,6) \wedge \neg P(7,7)
\end{aligned}$$

$$\begin{aligned}
\mathrm{IsDifferent}(x,y,x',y') &= \neg(\mathrm{IsNull}(x-x') \wedge \mathrm{IsNull}(y-y'))\\
\mathrm{IsNull}(0) &= \mathrm{True}\\
\mathrm{IsNull}(z) &= \mathrm{False}
\end{aligned}$$

$$\mathrm{Start} = \neg S(4,4,0) \wedge \bigwedge_{i=1}^{7}\bigwedge_{j=1}^{7} P(i,j) \wedge \mathrm{IsDifferent}(i,j,4,4) \rightarrow S(i,j,0)$$

$$\begin{aligned}
\mathrm{Step}(n) = \bigwedge_{m=0}^{n}\bigvee_{i=1}^{7}\bigvee_{j=1}^{7}\\
&P(i,j) \wedge P(i,j+1) \wedge P(i,j+2) \wedge S(i,j,m) \wedge S(i,j+1,m) \wedge \neg S(i,j+2,m) \wedge \neg S(i,j,m+1) \wedge \neg S(i,j+1,m+1) \wedge S(i,j+2,m+1)\\
&\wedge \bigwedge_{i'=1}^{7}\bigwedge_{j'=1}^{7} P(i',j') \wedge \mathrm{IsDifferent}(i,j,i',j') \wedge \mathrm{IsDifferent}(i,j+1,i',j') \wedge \mathrm{IsDifferent}(i,j+2,i',j') \wedge S(i',j',m) \rightarrow S(i',j',m+1)\\
&\vee P(i,j) \wedge P(i,j-1) \wedge P(i,j-2) \wedge S(i,j,m) \wedge S(i,j-1,m) \wedge \neg S(i,j-2,m) \wedge \neg S(i,j,m+1) \wedge \neg S(i,j-1,m+1) \wedge S(i,j-2,m+1)\\
&\wedge \bigwedge_{i'=1}^{7}\bigwedge_{j'=1}^{7} P(i',j') \wedge \mathrm{IsDifferent}(i,j,i',j') \wedge \mathrm{IsDifferent}(i,j-1,i',j') \wedge \mathrm{IsDifferent}(i,j-2,i',j') \wedge S(i',j',m) \rightarrow S(i',j',m+1)\\
&\vee P(i,j) \wedge P(i+1,j) \wedge P(i+2,j) \wedge S(i,j,m) \wedge S(i+1,j,m) \wedge \neg S(i+2,j,m) \wedge \neg S(i,j,m+1) \wedge \neg S(i+1,j,m+1) \wedge S(i+2,j,m+1)
\end{aligned}$$

Figure 10. Erroneous modelling of the Solitaire game.

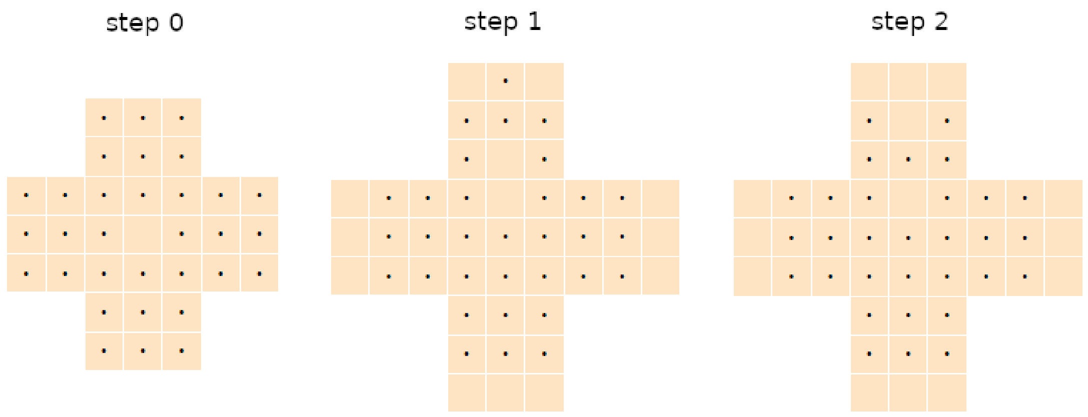


Figure 11. Problem-specific feedback as a graphical visualisation of the first 3 steps in Solitaire.

Acknowledgments

The work is partially funded by the “Stiftung Innovation in der Hochschullehre” within the project UKS_digi.

6. Bibliography

[1] Huth, M.; Ryan, M.: Logic in computer science: modelling and reasoning about systems. Cambridge University Press, Cambridge, 2004.

[2] Parkes, A. P.: Boolean Logic and Propositional Logic. In: Introduction to Languages, Machines and Logic: Computable Languages, Abstract Machines and Formal Logic, Springer, 2002, pp. 275–290. DOI: https://doi.org/10.1007/978-1-4471-0143-7_13 (last check 2025-09-23)

[3] Kautz, H. A.; Selman, B.: Planning as Satisfiability. In: Proc. of the 10th European Conf. on Artificial Intelligence, ECAI'92, 1992, pp. 359-363. https://dl.acm.org/doi/10.5555/145448.146725 (last check 2025-09-23)

[4] Clarke, E. M.; Biere, A.; Raimi, R.; Zhu, Y.: Bounded Model Checking Using Satisfiability Solving. In: Formal Methods in System Design, 2001, vol. 19, p. 7–34. https://link.springer.com/article/10.1023/A:1011276507260 (last check 2025-09-23)

[5] Massacci, F.; Marraro, L.: Logical Cryptanalysis as a SAT Problem. In: J. of Automated Reasoning, 2000, vol. 24, pp. 165–203. https://link.springer.com/article/10.1023/A:1006326723002 (last check 2025-09-23)

[6] Gesellschaft für Informatik (GI): Empfehlungen für Bachelor- und Masterprogramme im Studienfach Informatik an Hochschulen. Gesellschaft für Informatik e.V., Bonn, 2016. https://dl.gi.de/items/0986c100-a3b9-47c8-8173-54c16d16c24e (last check 2025-09-23)

[7] Huertas, M. A.: A classification of tools for learning logic. Universitat Oberta de Catalunya, 2011. https://openaccess.uoc.edu/items/8213d5fe-b877-4a3c-9e64-3ebce046b9b7#page=1 (last check 2025-09-23)

[8] Geck, G.; Ljulin, A.; Peter, S.; Schmidt, J.; Vehlken, F.; Zeume, T.: Introduction to Iltis: an interactive, web-based system for teaching logic. In: Proc. of the 23rd Annual ACM Conf. on Innovation and Technology in Computer Science Education, ITiCSE'18, 2018, pp. 141-146. https://doi.org/10.1145/3197091.319709 (last check 2025-09-23)

[9] Fernandez, J.; Gasquet, O.; Herzig, A.; Longin, D.; Lorini, E.; Maris, F.; Régnier, P.: TouIST: a Friendly Language for Propositional Logic and More. In: Proc. of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020, Yokohama, 2020, pp. 5240-5242. https://www.ijcai.org/proceedings/2020/756 (last check 2025-09-23)

[10] Gasquet, O.; Schwarzentruber, F.; Strecker, M.: Satoulouse: The Computational Power of Propositional Logic Shown to Beginners. In: Proc. of the Third International Congress on Tools for Teaching Logic, TICTTL 2011, 2011, pp. 77-84. https://link.springer.com/chapter/10.1007/978-3-642-21350-2_10 (last check 2025-09-23)

[11] Hundeshagen, N.; Lange, M.; Siebert, G.: DiMo - Discrete Modelling Using Propositional Logic. In: Proc. of the 24th Int. Conf. on Theory and Applications of Satisfiability Testing, SAT'21, 2021, pp. 242-250. DOI: 10.1007/978-3-030-80223-3_17 https://link.springer.com/chapter/10.1007/978-3-030-80223-3_17 (last check 2025-09-23)

[12] Herwig, M.; Hundeshagen, N.; Hundhausen, J.; Kablowski, S.; Lange, M.: Problem-Specific Visual Feedback in Discrete Modelling. In: Proc. of DELFI 2024 - Die 22. Fachtagung Bildungstechnologien der GI, Fulda, Germany, 2024.

https://dl.gi.de/server/api/core/bitstreams/15d05563-a785-4968-babc-111bc9c1b521/content (last check 2025-09-23)

[13] Green, T. R. G.: Cognitive dimensions of notations. In: People and Computers V. University Press, The Pennsylvania State University, 1989. https://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/papers/Green1989.pdf (last check 2025-09-23)

[14] Green, T. R. G.: Instructions and Descriptions: some cognitive aspects of programming and similar activities. In: Proc. of the Working Conf. on Advanced Visual Interfaces, AVI'00, 2000, pp. 21-28. https://dl.acm.org/doi/10.1145/345513.345233 (last check 2025-09-23)

[15] Blum, W.; Borromeo Ferri, R.: Mathematical Modelling: Can It Be Taught And Learnt? In: Journal of Mathematical Modelling and Application, vol. 1, December 2009. https://eclass.uoa.gr/modules/document/file.php/MATH601/3rd%20%26%204rth%20unit/3rd%20unit_Modelling%20cycle.pdf (last check 2025-09-23)

[16] Greca, I. M.; Moreira, M. A.: Mental, physical, and mathematical models in the teaching and learning of physics. In: Science Education, vol. 86, pp. 106-121, 2002. https://onlinelibrary.wiley.com/doi/10.1002/sce.10013 (last check 2025-09-23)

[17] Arcavi, A.: The role of visual representations in the learning of mathematics. In: Educational Studies in Mathematics, 2003, 52(3), pp. 215-241. https://link.springer.com/article/10.1023/A:1024312321077 (last check 2025-09-23)

[18] Lakhotia, A.: Understanding someone else's code: Analysis of experiences. In: Journal of Systems and Software, 1993, vol. 23, pp. 269-275. https://doi.org/10.1016/0164-1212(93)90101-3 (last check 2025-09-23)