Gradus Vol 5, No 2 (2018) 381-390
ISSN 2064-8014

Yggdrasil: a test question editor with symbolic expression support in high-throughput multi-language education

Péter Makra[1], László Égerházi[1] and Ferenc Bari[1]*

[1]Department of Medical Physics and Informatics, Faculty of Medicine, University of Szeged, Hungary

Keywords:
tests, question variants, symbolic expressions, C#

Article history:
Received 1 August 2018
Revised 4 September 2018
Accepted 1 Oktober 2018

Abstract
When preparing test questions for a large population of students, one usually needs to reconcile two conflicting factors: on one hand, the variability, and on the other hand, the consistent difficulty of questions. Random selection from a large pool of unrelated questions does ensure variability, but at the cost of comparable difficulty. One way to provide balanced difficulty is to create templates with variants that may differ in numeric values or relevant phrases, or may simply employ negation. Experience shows us, though, that manual variant management is as cumbersome as it is error-prone. In this paper, we shall introduce a program we wrote in C# to address this problem. The program, Yggdrasil*, supports the creation of templates with text-valued, logical or numeric symbolic expressions, manages their interdependence in an internal context and produces an output where the expressions are replaced by individual variants. The templates are easy to import into computer-assisted translation tools, allowing the translation of the templates themselves instead of the high number of variants. At the moment,* Yggdrasil *can provide xml output for the CooSpace learning management system used by the University of Szeged or LaTeX files for a printable pdf, but to increase availability, we plan to implement Word docx output.*

1 Introduction

The assessment of students is always a challenge regardless of the method of education. Though oral examination offers the greatest possible depth, it is inherently subjective, and above a certain student population it becomes unfeasible. To tackle the assessment of a large number of students, one needs to find a method that can be automated using computers. It is a science in its own right how to construct good automatable tests that lose as little of the probing depth in comparison with oral examinations as possible, and it is clearly outside the scope of this paper. What we focus on here is two problems associated with computerised tests: cheating and test difficulty.

A test question that can be automated is inherently easier for students to copy from their peers as it usually involves a choice between alternatives or a short numerical answer. To address this problem, one either ensures a greater physical distance between students during a test, thus sacrificing examination throughput, or gives different questions to each student. But if the questions are fully different, test difficulty inherently varies from student to student, it is hard to standardise performance and students may complain legitimately. A straightforward solution to avert this danger is to use questions that are not totally different but are variants of each other: multiple choice questions addressing the same concept, but with some variation (achieved using negation in the statements or

*Corresponding author. E-mail address: makra.peter@med.u-szeged.hu

a different choice out of a family of related or complementary concepts) in the introductory question or statement (*stem* [1]), in the true items (*keys* [1]) or the false items (*distractors* [1]), or numerical questions that use the same formula but with different numbers.

The more variants we have, the better we can balance variability with difficulty, but the harder it becomes to maintain the question bank. In creating variants of a multiple choice question, for instance, a negation in the stem may not be correctly reflected in the key or in the distractors, resulting in a faulty question having either a wrong key or multiple keys. Similarly, the greater the variation of the particular numerical values in a numerical question, the greater the probability of calculation errors. This problem is exacerbated in a multi-language environment, where question banks must be managed simultaneously in multiple languages. For consistent and feasible maintenance of question banks in such an environment, one must ensure the following:

1. a clear identification system that allows us to separate variants and trace them to their root template;

2. automation in variant generation to reduce the risk of human errors;

3. streamlined storage in which questions are reduced to templates and only these templates are stored and translated to other languages, whilst variants are created on demand from the translated templates.

In this paper, we shall introduce *Yggdrasil*, a test question editor program whose aim is to fulfil all these requirements. We have developed it in C#, using the .NET framework and the Windows Presentation Foundation (WPF) user interface architecture. It employs a symbolic notation we have devised, and this allows us to create templates from which variants of equal difficulty can be generated on demand in a consistent manner. It stores questions in its own XML format, which makes it compatible with computer-assisted translation tools, and it supports import from and export to CooSpace, the learning management system used by the University of Szeged. It can also produce LaTeX output that can be translated to printable pdf files for cases when computerised tests are not available.

2 The program

Yggdrasil is written in C# and relies on the classes provided by the .NET framework. The user interface is built using the Windows Presentation Foundation (WPF). We have written most of the code ourselves but used two third-party libraries: *mahApps* [2], which provides a more aesthetic user interface, a number of additional user controls and tools for easy application skinning; and *Math Parser .NET* [3], a library that allows the program to parse mathematical expressions in text format and translate them into programmatic representations of functions that can be evaluated at arbitrary values of the independent variables.

2.1 Questions

At the moment, *Yggdrasil* supports two types of questions: choice questions and numerical questions. Both of these derive from an abstract base class, which provides functionality shared by all question types and makes it possible for us to implement new question types in the future whilst staying compatible with the existing program logic.

All questions have a score, a culture and may have category labels. The score is the amount of points the learning management system – CooSpace in our case – awards for a correct answer. The culture is currently either English, German or Hungarian, and it determines the decimal separator in the textual representation of numbers. Categories serve as descriptors on the basis of which we plan to support searching or filtering by subject, topic or difficulty, but we have not yet implemented this feature.

Questions also have a unique identifier (ID), which can be an arbitrary piece of text, but we prefer structured identifiers that carry information on the subject, the topic, the index of the template and that of the particular version, each separated by a fixed character, such as a full stop. The identifier `physics.basics.18.2` in panel A of Figure 1, for example, tells us that it is a physics question, from a collection that deals with basics like SI, scalars v vectors, *&c,* and the third variant of the 19th disjunct question of this collection (indexing starts at 0). When creating variants of a question, *Yggdrasil* creates a variant identifier postfixing the identifier of the parent question with a full stop followed by the index of the variant. This system allows us to decide whether a particular question is a variant of a template or a disjunct question.

2.1.1 Choice questions

We apply the term 'choice question' to fuse English terminology, in which 'multiple choice' denotes a question with multiple items to choose from but with only one key, with Hungarian terminology, which distinguishes between multiple-item questions with one key and those with multiple keys. Where distinction is necessary, we shall call these 'single-key choice questions' and 'multiple-key choice questions', respectively. Panel A of Figure 1 illustrates a choice question in *Yggdrasil*.

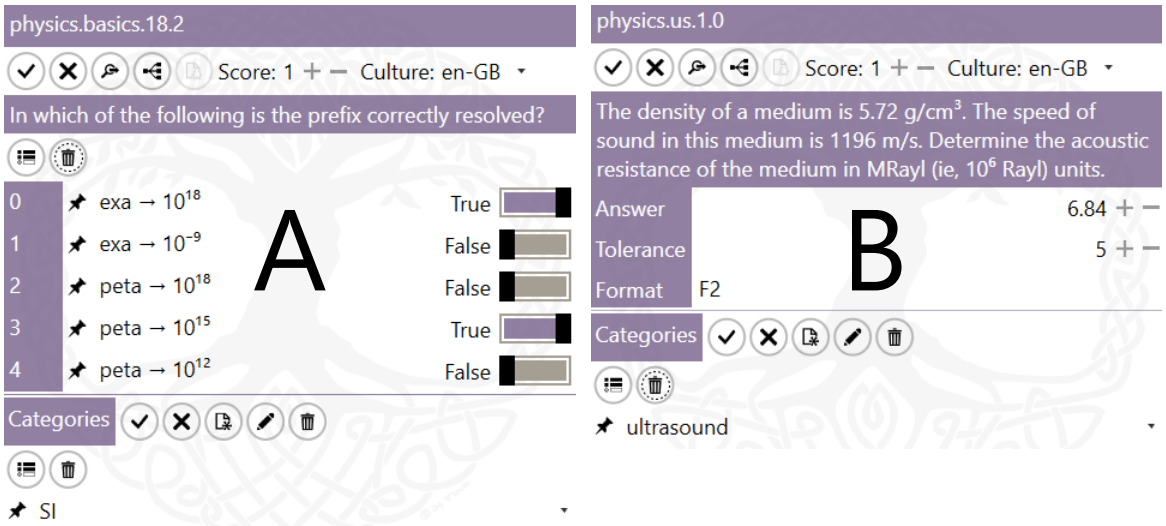


Figure 1. A multiple-key choice question (A) and a numerical question (B) in the question editor panel of Yggdrasil

The program itself does not distinguish between single-key and multiple-key choice questions: they belong to the same class, only the number of keys is different. In CooSpace, however, these represent two different question types. The only ambiguous case is when we export a choice question with a single key to CooSpace, since this can be understood either as a CooSpace question of single-key format or a question of multiple-key format which happens to have a single key. *Yggdrasil* offers an option to decide between single-key and multiple-key CooSpace export.

2.1.2 Numerical questions

Numerical questions are questions whose answer is a single real number. To allow for rounding errors, numerical questions have a tolerance value (5% by default), which determines the interval in which we consider an answer correct. CooSpace does not currently support entering a unit or a number in scientific format, so we prefer to specify the unit in the question text and choose an SI prefix for which the numeric value is convenient to enter. Panel B of Figure 1 shows an example of a numerical question in *Yggdrasil*.

2.2 Organisation and storage

Yggdrasil has a Windows installer, which, in addition to copying the required files on the computer, will associate two file types with *Yggdrasil* in the registry: catalogue files (extension: .ycm) and assortment files (extension: .yar). These will appear with the correct description and with their own icons in *File Explorer,* and double-clicking will open them in *Yggdrasil*.

2.2.1 Catalogues

A catalogue in *Yggdrasil* is a collection of questions similar in topic, type or difficulty. A catalogue is stored as a single xml file and can be exported to a single CooSpace quiz (see Subsection 2.4). We use catalogues to establish pools of questions of the same topic or group examination questions by topic, type and difficulty. A catalogue may contain simple questions or templates (see Section 3). Each question ID within a catalogue must be unique.

2.2.2 Assortments

Assortments represent larger units consisting of several catalogues. A typical use of an assortment is to represent a particular test, within which each catalogue is a pool of questions from the same topic and of similar difficulty. In the final realisation of the test, the learning management system can allot to each student a set number of questions randomly selected from each catalogue, ensuring that students receive questions that are similar in topic and difficulty, but are not identical. *Yggdrasil* stores assortments as zip files that contain the catalogues as individual xml files, and can export them to a zip file CooSpace will import as a folder structure with catalogues as subfolders.

2.3 User interface

The user interface of *Yggdrasil* is split into two panels of equal width. On startup, each panel is occupied by an assortment view. An assortment view contains a catalogue view showing the questions in the currently selected catalogue as a list of text blocks. The user can open assortments or catalogues in both panels and drag and drop questions between the panels with the mouse. Certain user actions, such as double-clicking a question in a catalogue to edit it, or generating variants of an assortment or a catalogue, will bring up a new view in the opposite panel. In such cases, the view previously occupying a panel will be saved on a stack to be restored when the new view closes (eg, when the user has finished editing the question). Figure 2 shows such an arrangement.

2.4 Output files

Our computerised tests are hosted by CooSpace, which is a learning management system that services a few Hungarian universities, including the University of Szeged. CooSpace imports and exports questions in a simple xml format, which was easy to implement in *Yggdrasil*. As mentioned above, the two possible import options are a simple xml file for stand-alone question collections that get translated into a single question folder in CooSpace, or a structured zip file, which will be imported into a folder structure of several question banks. Figure 3 shows how CooSpace renders the questions we have exported.

To facilitate offline tests, *Yggdrasil* can also generate LaTeX source files, which can be compiled into printable pdf files (as shown in Figure 4) if LaTeX is installed on the computer. These files contain both the student copy and the key, the latter also in a grid format, which makes correcting papers more convenient. Since LaTeX is only used by a relatively small dedicated community, we consider it our priority in future development to implement similar output in Microsoft Word to increase accessibility.

3 Templating and variant generation

Since the most important feature of *Yggdrasil* is the symbolic framework that makes it capable of handling templates, we shall devote a whole section to this. What we refer here as a template

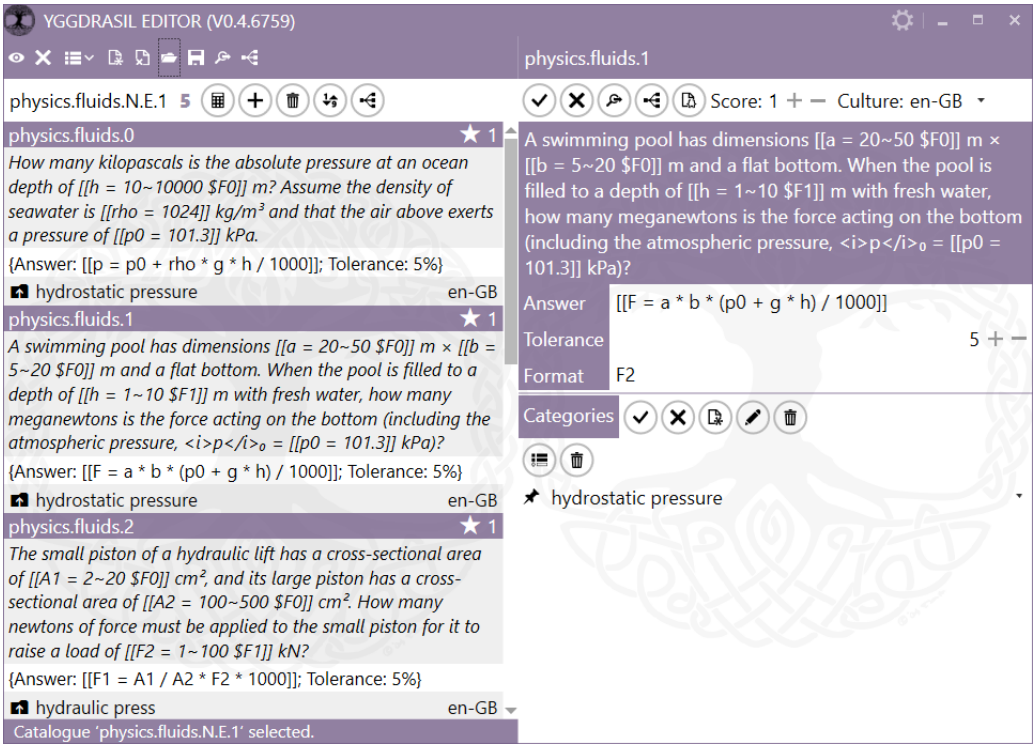


Figure 2. The user interface of Yggdrasil *with the active catalogue in the left panel and the question being edited in the right panel*

is basically a question of which we can generate variants. Templates can either be symbolic or non-symbolic.

3.1 Non-symbolic templates

Templates need not contain symbolic expressions to allow the creation of variants. Consider a choice question that contains a large pool of keys and distractors. A variant of such a template represents a random selection of a set number of keys and distractors out of the items of the template.

3.2 Symbolic templates

To achieve the required flexibility and consistency, we need to establish a syntax of symbolic expressions. In *Yggdrasil*, any component of a question may contain such expressions, which are denoted in the text as sections enclosed in double square brackets. Within the double square brackets, each symbolic expression must contain a symbol that must be unique within a single question (different questions may freely use the same symbol but no question may contain two expressions with the same symbol), followed by an equals sign and the expression body. The last section of a symbolic expression may be an optional format specifier (prefixed by a dollar sign), which regulates how the evaluated result of the symbolic expression will appear in the text of the variants. Most format specifiers are standard C# numeric format strings, but *Yggdrasil* also uses a custom format specifier H, which causes the value of expression to be hidden in the output – this is useful when defining expressions that are needed in the calculations but must not appear in the final text. The format specifier not only determines the format and the number of decimals of numerical expressions in the text, but also tells *Yggdrasil* to round the numerical expression internally to the number of decimals specified, in order to prevent rounding errors from appearing in the output. For example, the text [[**b** = 5~20 $F0]] on the right in Figure 2 defines a symbolic expression **b** whose value is an integer (F0 is the standard C# numeric format string representing a number with no fractional decimals) picked randomly from the interval between 5 and 20 (see 3.2.4).

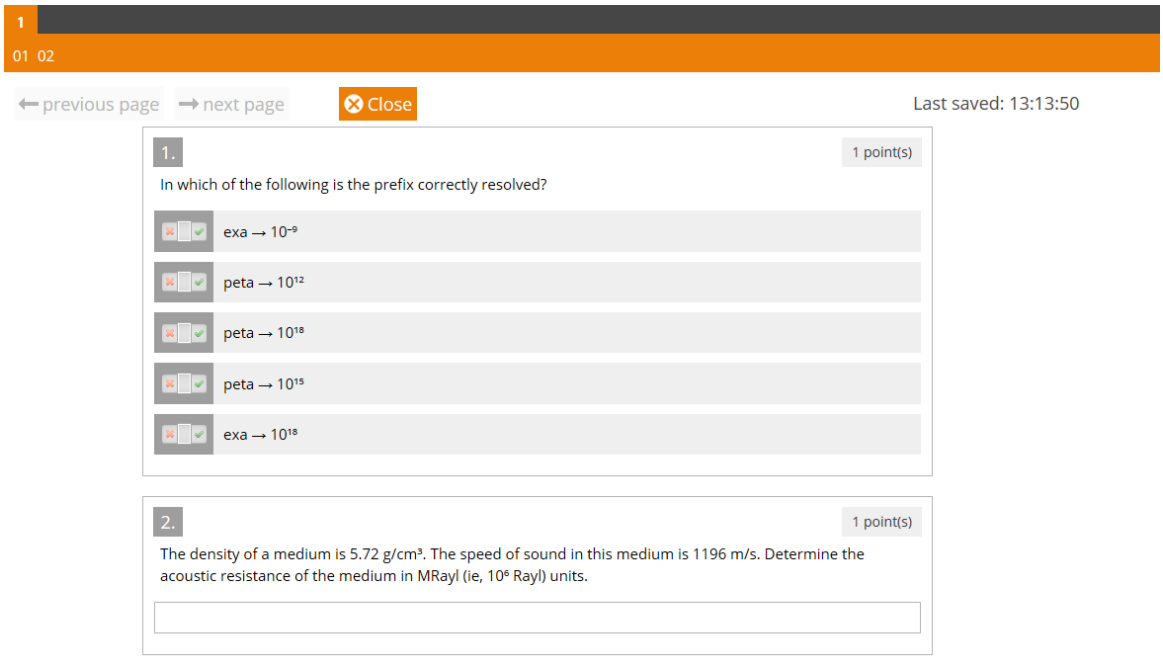


Figure 3. The questions from Figure 1 as they appear in CooSpace

Yggdrasil traces symbolic expressions as regular expression patterns. When generating variants, it looks for such patterns in the text and builds an internal database (the *context*) out of them. A context is essentially a list of expressions uniquely identified by their symbol, which allows cross-referencing between expressions. Internally, all symbolic expressions evaluate to either a number (rounded to the precision given by the format specifier), a piece of text or a logical value. In the output of variant generation, symbolic expressions are replaced by the text representation of their current values.

Expressions may rely upon the values of other expressions within the same context. Evaluation is always preceded by a parsing round, so expressions can refer to other expressions that are defined later in the text. To continue the previous example, [[**A** = **b**^2]] will calculate the square of the current value of **b**, and it will work correctly even when it precedes the definition of **b** in the text.

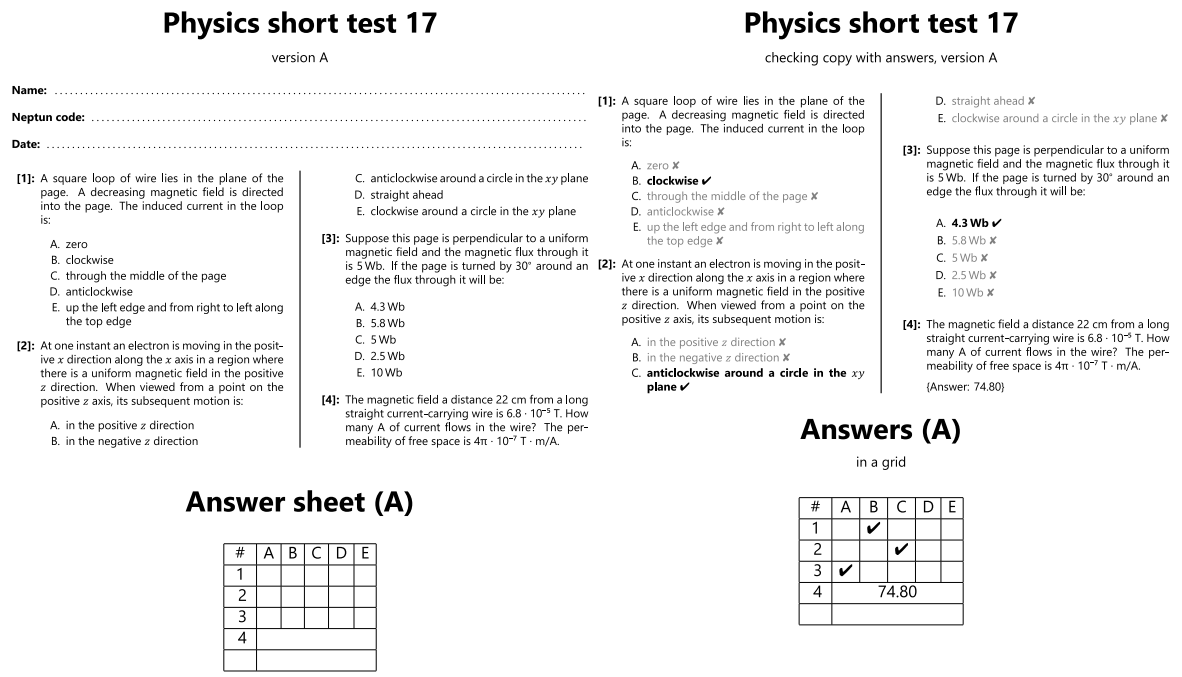
Symbolic expressions may be non-variable or variable. Non-variable expressions are either constants or are fully determined by the value of other expressions, whilst variable expressions have an internal state that may change from variant to variant. This internal state may simply be an index that determines which element of a table (see 3.2.2) we choose or the state of a random generator in a random pick (see 3.2.4). In the examples above, **b** is variable and **A** is non-variable. Variable expressions are required for a template to have several variants; if a template contains only non-variable symbolic expressions, it will only yield a single variant.

To illustrate the templating and variant generation process with an example, let us take the question being edited in in Figure 2. It reads

> A swimming pool has dimensions [[**a** = 20~50 $F0]] m × [[**b** = 5~20 $F0]] m and a flat bottom. When the pool is filled to a depth of [[**h** = 1~10 $F1]] m with fresh water, how many meganewtons is the force acting on the bottom (including the atmospheric pressure, p$_0$ = [[**p0** = 101.3]] kPa)?
> Answer: [[**F** = **a** * **b** * (**p0** + **g** * **h**)]]

When generating variants, *Yggdrasil* will first parse the whole text of the question and find the following symbolic expressions:

- **a**, **b**, **h**: random picks (see 3.2.4);
- **p0**: constant (see 3.2.1);
- **F**: formula (see 3.2.5), a function of all other expressions in the context;



Physics short test 17

version A

Name: .....

Neptun code: .....

Date: .....

[1]: A square loop of wire lies in the plane of the page. A decreasing magnetic field is directed into the page. The induced current in the loop is:

A. zero
B. clockwise
C. through the middle of the page
D. anticlockwise
E. up the left edge and from right to left along the top edge

[2]: At one instant an electron is moving in the positive x direction along the x axis in a region where there is a uniform magnetic field in the positive z direction. When viewed from a point on the positive z axis, its subsequent motion is:

A. in the positive z direction
B. in the negative z direction
C. anticlockwise around a circle in the xy plane
D. straight ahead
E. clockwise around a circle in the xy plane

[3]: Suppose this page is perpendicular to a uniform magnetic field and the magnetic flux through it is 5 Wb. If the page is turned by 30° around an edge the flux through it will be:

A. 4.3 Wb
B. 5.8 Wb
C. 5 Wb
D. 2.5 Wb
E. 10 Wb

[4]: The magnetic field a distance 22 cm from a long straight current-carrying wire is $6.8 \cdot 10^{-5}$ T. How many A of current flows in the wire? The permeability of free space is $4\pi \cdot 10^{-7}$ T · m/A.

Answer sheet (A)

#	A	B	C	D	E
1					
2					
3					
4					

Physics short test 17

checking copy with answers, version A

[1]: A square loop of wire lies in the plane of the page. A decreasing magnetic field is directed into the page. The induced current in the loop is:

A. zero ✘
B. **clockwise** ✔
C. through the middle of the page ✘
D. anticlockwise ✘
E. up the left edge and from right to left along the top edge ✘

[2]: At one instant an electron is moving in the positive x direction along the x axis in a region where there is a uniform magnetic field in the positive z direction. When viewed from a point on the positive z axis, its subsequent motion is:

A. in the positive z direction ✘
B. in the negative z direction ✘
C. **anticlockwise around a circle in the xy plane** ✔
D. straight ahead ✘
E. clockwise around a circle in the xy plane ✘

[3]: Suppose this page is perpendicular to a uniform magnetic field and the magnetic flux through it is 5 Wb. If the page is turned by 30° around an edge the flux through it will be:

A. **4.3 Wb** ✔
B. 5.8 Wb ✘
C. 5 Wb ✘
D. 2.5 Wb ✘
E. 10 Wb ✘

[4]: The magnetic field a distance 22 cm from a long straight current-carrying wire is $6.8 \cdot 10^{-5}$ T. How many A of current flows in the wire? The permeability of free space is $4\pi \cdot 10^{-7}$ T · m/A.

{Answer: 74.80}

Answers (A)

in a grid

#	A	B	C	D	E
1		✔			
2			✔		
3	✔				
4	74.80				

Figure 4. The printable output of Yggdrasil*, generated using LaTeX. The student page is on the left, whilst the key is on the right.*

- **g**: constant – not defined in the text but a built-in constant of *Yggdrasil*.

The random picks are variable expressions, whilst the formula and the constants are non-variable. To create a variant, *Yggdrasil* will change the state of the variable expressions – in this case, generate a new random number within the specified bounds for each random pick **a**, **b** and **h**. Each (**a**, **b**, **h**) combination of variable expression states defines a new variant. In the final step, *Yggdrasil* will evaluate **F** using the current value of **a**, **b** and **h** and that of the constants, and will replace each symbolic expression definition with the textual representation of the current value of the given expression. The final form of a variant will be like

> A swimming pool has dimensions 42 m × 16 m and a flat bottom. When the pool is filled to a depth of 7.5 m with fresh water, how many meganewtons is the force acting on the bottom (including the atmospheric pressure, p$_0$ = 101.3 kPa)?
>
> Answer: 117.52

Then *Yggdrasil* choses a new random value for **a**, **b** and **h** each, and repeats the process above. The decimal separator in the text is determined by the culture of the question: point for English and comma for German or Hungarian questions.

Yggdrasil ensures that each variant thus generated is unique: it maintains a table of unique integer keys (a hash table) for each variant using the built-in algorithm of .NET, which guarantees identical keys for identical variants, and if the table already contains the key of a 'new' variant, the variant is discarded. To avoid infinite loops in cases where no new variants are available, *Yggdrasil* defines a timeout: a maximum number of iterations above which it aborts variant generation even if it could not create enough variants.

In what follows, we shall provide an overview of the types of symbolic expressions *Yggdrasil* recognises.

3.2.1 Constants

Constants are numeric, text or logical literals that can be referenced by other symbolic expressions. Examples: [[**g** = 9.81 $H2]] for a numeric, [[**t** = "text" $H]] for a text and [[**b** = true $H]] for a logical constant. Constants are of course non-variable expressions.

3.2.2 Tables

Tables enumerate numeric, text or logical literals one after the other, separated by a semicolon. The list of literals in a table must be enclosed in braces, eg [[**t** = { "ultrasound"; "CT" }]]. Tables are variable expressions whose state is reflected by the index of the item currently chosen. When generating a new variant, *Yggdrasil* chooses a new index randomly between 0 and the number of items in the table minus 1.

3.2.3 Lookup tables

Lookup tables are tables whose value is determined by the current index of other tables. Their form is similar to that of tables, but the brace-enclosed list of literals is followed by a comma-separated list, in brackets, of the symbols of other tables whose current index determines the current value of the lookup table. Lookup tables are non-variable expressions which are useful when we have to select one out of a finite set of options depending on the state of other tables.

Example:

‘The [[**t** = { "lower threshold of the audible frequency domain for humans"; "upper threshold of the audible frequency domain for humans"; "lower threshold of the frequency domain for human speech"; "upper threshold of the frequency domain for human speech" }]] is [[**f** = { 0.02; 20; 0.3; 3 }(**t**)]] kHz. Assuming a speed of wave of [[**c** = 330]] m/s, how many metres is the wavelength of this tone?

Answer: [[**lambda** = 0.001 * **c** / **f**]]’

Here **t** is a table of four different choices. The lookup table **f** defines corresponding numeric values for each choice in **t**, in the same order as the choices appear in **t**. It is not a variable expression, which would determine a new variant, but a dependent expression whose current value is determined by the state of **t**. This template has four potential variants as the only variable expression in it, **t**, contains four distinct items. If the current value of **t** is "upper threshold of the audible frequency domain for humans", the second item in **t**, **f** will evaluate to its second item, 20.

Lookup tables may depend on multiple tables. In such cases, the enumerated body of the lookup table is to be understood as a two- (or higher) dimensional matrix whose rows are indexed by the first table in the argument list and whose column index is determined by the second table, *&c.* To make these lookup tables readable despite the fact that the elements are enumerated in the same row without line break, *Yggdrasil* also accepts the | character as a separator in addition to semicolons. Let us see an example.

‘Open the workbook CT.xlsx. Calculate the sinogram in worksheet [[**sheet** = { "A"; "B"; "C"; "D"; "E" }]]. What value does cell [[**cell** = { "F14"; "D15"; "C16" }]] of the sinogram contain? Read the values from left to right in the 0° projection, from the lower left corner to the upper right corner in the 45° projection and from bottom to top in the 90° projection, and enter them into the respective row in the sinogram from left to right.

Answer: [[**m** = { 3 ; 17 ; 6 | 5 ; 15 ; 3 | 5 ; 19 ; 8 | 2 ; 12 ; 2 | 6 ; 13 ; 4 }(**sheet**, **cell**)]]’

Here the lookup table **m** depends on two tables, **sheet** and **cell**. Sections delimited by | indicate enumeration by **sheet**, within which semicolons separate values that are indexed by **cell**. This means that when **sheet** assumes its second possible value, "B", and the current value of **cell** is the third one, "C16", **m** will evaluate to the third semicolon-separated item within the second vertical line delimited segment, that is, 3. The definition of **m** would be syntactically correct with semicolons instead of vertical lines and would produce the same output, but it would lose readability.

3.2.4 Random-pick expressions

The most crucial element of flexible variant creation is the ability to generate random numeric values. In *Yggdrasil*, random-pick expressions serve this purpose. They are variable expressions defined by two numeric literals (lower and upper bounds) separated by a tilde sign. To pick one example from the ones already mentioned: [[**h** = 1~10 $F1]]. This expression generates a random

number between 1 and 10 and rounds it to a single decimal place. It is important to note again that the format specifier (F1 in the example) does not only regulate the text representation of the result but it also instructs the program to round the random number to the specified precision (one digit in the example) and use this rounded value in all further tasks. This way we can avoid situations where the student using the rounded values that are displayed is bound to have a result different to the internal one based on more precise numbers.

3.2.5 Formulae

No quantitative question worth asking would be possible without the ability to incorporate calculations into the symbolic framework. To do this, we need to be able to recognise numbers, variables, mathematical operators and functions in the text, and translate them into executable program elements. This is called parsing. We did not want to reinvent the wheel and chose an existing open-source solution, *Math Parser .NET*, to accomplish this task. This library can parse text containing numbers, symbols, brackets, operators ($+, -, *, /, \%$ – the modulo operator, ^ – exponentiation) and basic mathematical functions (abs, sin, cos, tan, log, logn) and build executable algorithms out of them. In addition to built-in functions, it allows the user to register custom functions without having to edit the source code, which we do leverage in *Yggdrasil* to offer inverse sine, cosine and tangent; sine, cosine and tangent whose argument is provided in degrees (and not radians as in the built-in implementation) and functions for conversions between degrees and radians.

Math Parser .NET is an open-source library whose source we can and may (under Code Project Open Licence 1.02) modify, and this was important to us mainly because *Yggdrasil* needs to communicate with the parsing engine to know if a given piece of text is recognised as a mathematical expression and to register the list of symbols such a mathematical expression is dependent upon.

Using the functionality that *Math Parser .NET* offers, *Yggdrasil* defines formulae as non-variable symbolic expressions containing numbers, symbols of numerical expressions and the operators and functions discussed above which evaluate to a numeric value. For instance, the expression [[**F** = **a** * **b** * (**p0** + **g** * **h**)]] in the first example is a formula. When parsing, *Math Parser .NET* will recognise the expression body as a valid mathematical expression and identify the symbols **a**, **b**, **p0**, **g** and **h** as variables upon which the formula depends. *Yggdrasil* stores the list of these variables and the expression body, and when we evaluate the formula, it will retrieve the current value of each, register the values with *Math Parser .NET*, which in turn will evaluate the expression body using the current values.

3.2.6 Custom methods

Although *Math Parser .NET* does allow us to register custom functions, it still has a limitation: these functions can only have real numbers as their input or output. If we wish to process text or logical information, we have to implement our own custom methods. These are similar in appearance to the mathematical functions in formulae: the method name is followed by a pair of brackets which contain the comma-separated list of the symbols of the independent variables. The two main differences are that we are only allowed to list the symbols and no arithmetic is allowed within the brackets, and that the symbols can (and usually do) refer to non-numeric (that is, text or logical) variables or can return non-numeric values. The three custom methods at the moment are huart(**w**), which takes a text variable **w** and returns the appropriate Hungarian definite article (‘a’ or ‘az’); Huart(**w**), which does the same, only with a capital initial for the definite article (‘A’ or ‘Az’) and eq(**a**, **b**), which has two compulsory arguments and returns true when these are equal and false otherwise. Our future plan is to merge this category with formulae and rewrite our custom copy of *Math Parser .NET* so that it can handle text and logical values.

4 Conclusions and outlook

In this paper, we have introduced *Yggdrasil*, our test question editor with templating capability, devoting special attention to the symbolic expressions that make such templating possible. We are confident that such a framework as the one shown here fulfils the main requirements we outlined in the Introduction: it can support tests that make cheating less feasible whilst ensuring consistent difficulty. We are currently in the process of testing this statement with the statistical analysis of three years – six semesters – of data. We have been applying these templated tests for three years in our medical physics practicals. More than 500 students write these tests each year in three languages – English, German and Hungarian –, not all together but distributed into groups all over the week, which gives us a unique opportunity to check whether there is an efficient exchange of information between groups that wrote the test at the beginning of the week and those that wrote it later. Though it does not yield insight into the feasibility of cheating per se, but might give us a hint at how easy it is to pass information on the questions between the students, and these two are obviously related.

Currently *Yggdrasil* is geared towards our local needs with its CooSpace and LaTeX export capabilities, but it is easy to extend these if the desired export format is documented. One of our main priorities for the future is to implement Microsoft Word export, which has been made possible by the fact that docx is an open and documented format, however convoluted it may be. Even before that, our next step will be to incorporate searching and filtering functions, which will make use of the already supported categorisation capability we discussed in Subsection 2.1.

References

[1] Jerard Kehoe, 'Writing multiple-choice test items,' *Practical Assessment, Research & Evaluation,* Nov-1995 [Online]. Available: https://pareonline.net/getvn.asp?v=4&n=9. [Accessed: 18-Jul-2018].

[2] mahapps.metro [Online]. Available: https://mahapps.com/. [Accessed: 18-Jul-2018]

[3] Icemanind, 'Math Parser .NET,' *Code Project,* 31-Oct-2011 [Online]. Available: https://www.codeproject.com/articles/274093/math-parser-net. [Accessed: 18-Jul-2018]