Automatic Item Generation Integrated into the E-Assessment-System JACK

Michael Striewe[a]
Trier University of Applied Sciences, Department of Computer Science, Trier, Germany

Keywords: Automatic Item Generation, E-Assessment System, Educational Technology Engineering.

Abstract: Automatic item generation (AIG) can save time in the production of high-quality assessment items, but requires to create and maintain appropriate software tools that fit into a larger context in which the generated items are to be used. Hence, sustainable AIG solutions not only require sophisticated item generation capabilities, but also appropriate software design. This paper presents a concept for AIG that is integrated into an e-assessment system and promotes reusability and extensibility as its main software quality properties. The paper demonstrates the practicality of the concept and discusses the underlying software structure.

1 INTRODUCTION

Automatic item generation (AIG) is well-known for saving the time required to produce high-quality assessment items with defined properties, but it comes at the cost of time required for item modeling and tool development (Kosh et al., 2019). Item modeling is an integral part of a particular approach to automatic item generation (Gierl and Lai, 2012a) and thus cannot be eliminated if that approach is to be used. Instead, there is a chance to reduce the need for tool development by proper software engineering with a focus on reusability and integration. However, there are many single-purpose tools that are not specifically designed to be reusable or integrated. Nevertheless, using such a tool in practice sometimes "requires [...] digital ecosystem around the tool" (Kıyak and Kononowicz, 2024), which is a strong argument to not only focus on generation capabilities but also on software design quality in automatic item generation.

Reusability is improved with universal AIG tools like IGOR (Mortimer et al., 2012), which is still a standalone tool. Another universal AIG tool is CAFA (Choi and Zhang, 2019), which is not only reusable but also offers integration as it can serve as a platform that can be used by different client systems, such as item management systems or interactive workbooks (Choi et al., 2018).

In addition to reusability and integration, the extensibility of item generators is an additional concern. The current state-of-the-art in automatic item generation knows many different approaches, although it is dominated by "template or rule-based approaches as the primary method for creating item models" (Circi et al., 2023). These may be somewhat limited with respect to the complexity of the generated items (Baum et al., 2021). The reason may be that existing tools are not well designed to be extended by emerging approaches that can help to add complexity. This inevitable leads to situations in which new tools are designed (rather than existing ones are extended) as soon as an important delta between existing methods and the specific requirements in a particular domain is detected (Christ et al., 2024).

Extensibility is improved with AIG frameworks like SARAC (Liu, 2009), which uses individual components for the parameterizable elements that occur within an item model.

This paper presents an entirely different approach, in which item generation features have been added to an existing e-assessment system. In this way, automatic item generation is integrated directly into a tool that actually uses the items, reusable across all items created in that tool in any domain of study, and extensible with clearly defined interfaces to add more functionality in the future.

The remainder of this paper is organized as follows: Section 2 describes the concept of item generation used within the system. Section 3 provides two examples on how to use the concept in practice. Section 4 explains the software structure that is used to

[a] https://orcid.org/0000-0001-8866-6971

Striewe, M.
Automatic Item Generation Integrated into the E-Assessment-System JACK.
DOI: 10.5220/0013454600003932

In *Proceedings of the 17th International Conference on Computer Supported Education (CSEDU 2025) - Volume 1*, pages 747-753
ISBN: 978-989-758-746-7; ISSN: 2184-5026

implement the concept. The section also explains the software interfaces that allow to extend item generation. Section 5 discusses limitations of the approach, and Section 6 concludes the paper.

2 CONCEPT OF ITEM GENERATION

The e-assessment system JACK is a web-based, general-purpose assessment system that can be used in virtually any domain of study (Striewe, 2016). Thus, its item types are not limited to those that are typically used with automatic item generation, but range from universal types like multiple choice and fill-in-the-blanks to domain-specific types that ask students to write program code or draw molecules. The system does not provide a fixed set of items or item templates, but includes authoring features that allow teachers to define their own assessment items. In addition, actual assessment items can consist of several parts, where each part may be of a different type and where the sequence of parts is determined adaptively based on previous student input. For most item types, automatic grading and feedback generation are available. In fact, the ability to provide detailed and elaborated feedback for complex item types is one of JACK's most prominent features. Again, the system does not do so based on fixed algorithms in most cases but provides authoring features so that teachers can prepare detailed feedback definitions.

Consequently, the concept of item generation in JACK had to fulfill several requirements:

1. Item generation must happen online at runtime each time a student starts to interact with a new part of an assessment item.
2. Item generation must be based on functions that are independent of the actual type of the generated items so that it can be used with any type, including types that may be added in the future.
3. Item generation must cover both the item contents shown to students and the internal parts of an item that are used during automated grading and feedback generation.

Based on these requirements, JACK uses a *template-based approach* by introducing so-called *item variables*. For each assessment item (that may consist of several parts as mentioned above), authors can define an arbitrary number of such variables. Each variable definition must include a so-called *evaluator function* that determines its value. These functions may refer to other variables, so that constraint or dependent variables are possible. The actual contents of item variables are typically numbers or strings, but may also be complex mathematical formulas. In addition to item variables, authors may also define so-called *dynamic objects* that will render graphical elements in the final output such as mathematical graphs or chemical molecules. The method for defining dynamic objects is less generic than the evaluator functions for item variables, but may still refer to item variables, so that dynamic objects can be constraint and dependent as well. Finally, authors can use placeholders in virtually any part of their item definition (i. e. in all texts prepared for students and in all rules for grading and feedback generation) to turn an item into a templates from which different instances can be derived at runtime. In summary, the only pre-defined parts provided by the e-assessment system are general item types, types of dynamic objects, and evaluator functions for the generation of variable values, but no pre-defined item templates, item models or other kinds of structures. Authors are free to combine any of these provided parts in any way they like to define item templates with placeholders that are filled by variable values. Authors are also free to define arbitrary complex combinations of functions to define variable values.

Notably, it is left to the item authors to decide if they apply a strong theory approach (Gierl and Lai, 2012b) and derive the item variables from an item model they manage externally, or if they apply a weak theory approach and just turn a fixed item in JACK into a parent item by adding some variables. Since it is not mandatory to use each and every item variable for at least one placeholder, it is also possible to define a set of item variables that actually encode a complex item model directly in JACK. Whether this is actually meaningful depends on the complexity of the model and the availability of appropriate evaluator functions to express the relations between the model elements.

3 EXAMPLES

This section illustrates the concept of item generation by two examples. Both examples involve dynamic objects of different kind. The first example is a fill-in-the-blanks item that includes a dynamic object and involves automatic grading. The second example is a simple multiple choice item that also features a dynamic object and automatic grading, but additionally involves the use of external data sources during item generation.

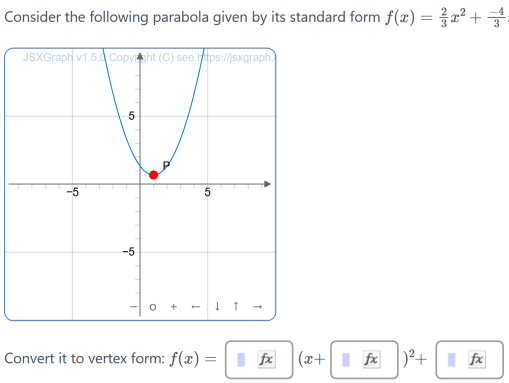

Figure 1: Student view on a sample instance of an item on the forms of parabola equations.

3.1 Example 1: Parabola Curves

The first example comes from mathematics education. The item is a fill-in-the-blanks item and asks students to convert a parabola equation from standard form to vertex form (see Figure 1). The item template is divided into two parts to cover both the item presentation and the automated grading. Both parts of the template refer to item variables. For item presentation, authors define the item stem which includes not only references to item variables, but also a placeholder for the curve plot as well as three input fields (see Figure 2). Authors can decide to use LaTeX formatting for the formula and can equip the input fields with a formula editor, so that students can enter rational numbers and alike in a convenient manner. For automated grading, authors specify rules that refer to the input fields and compare their values to expected values that may be fixed or stored in item variables (see Figure 3 for an example with one rule for the correct answer). Note that rules can refer to any input fields they like, so that there can be more than one correct solution per input field. Similarly, there can be multiple rules referring to the same input fields, resulting in elaborate feedback for different (wrong) inputs.

The complete list of item variable definitions is given in Figure 4. The variables are based on a mathematical model that makes sure that the equation will not get trivial but also not too hard to calculate. Consequently, the first four variables (“az”, “an”, “d”, and “e”) make random selections from carefully selected value ranges. Note that the value 0 is excluded in all cases, since it could cause invalid results. The next nine variables all compute derived values based on the previous variables. All of these variables are used internally either to prepare the curve plot (see Figure 5) or the rules for grading and feedback generation. In particular, some of the variables capture typical miscalculations students can make, so that student input can be compared to these values as well to generate specific feedback for these cases. The final four variables (“output1” to “output4”) are used within the item template to display information to students. They make use of a special evaluator function “rational” which makes sure that its parameters will be printed as rational number and not as decimal number.

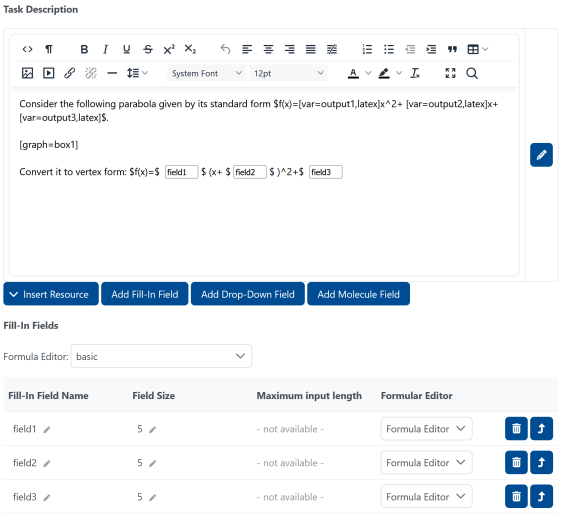

Figure 2: Authoring view on the item template for the item shown in Figure 1.

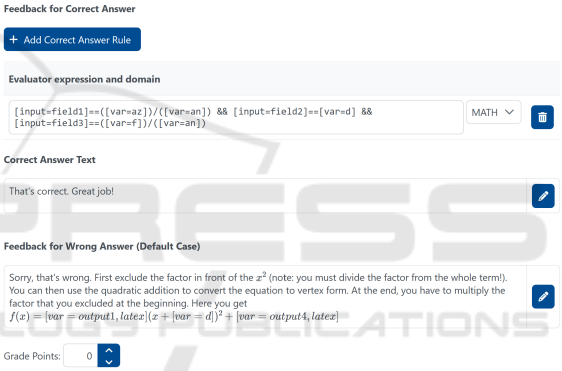

Figure 3: Authoring view on one of the feedback rules for the item shown in Figure 1.

3.2 Example 2: Molecules

The second examples comes from chemistry education. The item asks students to indicate the correct name for a given chemical compound (see Figure 6). The corresponding template is quite simple with an item stem and four answer options (see Figure 7). The item stem includes a placeholder referring to a dynamic object of type “molecule” and all four answer options refer to item variables.

Figure 4: Authoring view on the item variables used in conjunction with the item template in Figure 2 and Figure 3.

Figure 5: Authoring view on the dynamic object that renders the parabola in Figure 1.

The complete list of item variable definitions is given in Figure 8. It starts with drawing a random value for variable “size” that will roughly determine the complexity of the molecules that will appear in the item. Note that this is a simple heuristic that does not match academic standards in item writing, but is used here just for the sake of brevity. The main complex-

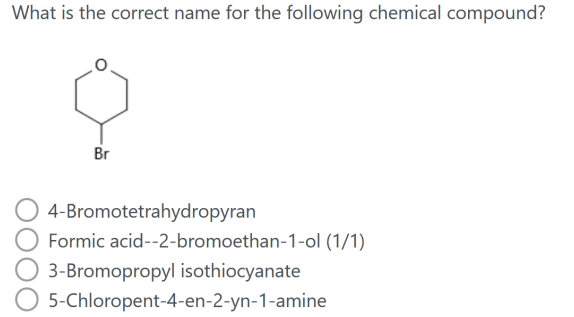

Figure 6: Student view on a sample instance of an item on chemical compounds.

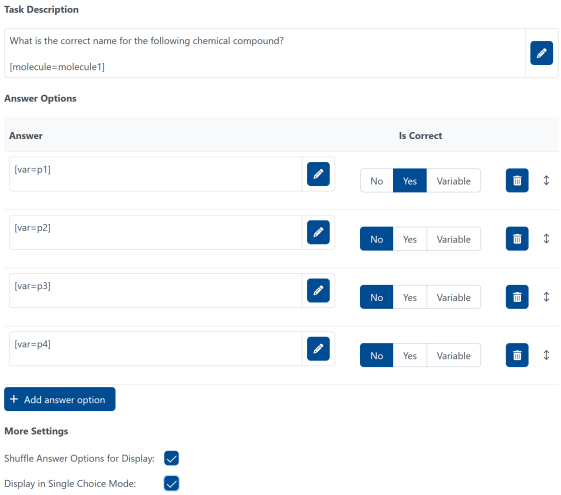

Figure 7: Authoring view on the item template for the item shown in Figure 6.

ity of the item generation is encoded in the second variable “tuples” that makes use of a call to an external data source. Precisely, a SPARQL query is issued against the Wikidata database to retrieve chemical compounds, their names, and their structural identifiers. Using SPARQL queries to generate items has been explored earlier (Foulonneau and Ras, 2013) and is thus not a unique feature of automatic item generation with JACK.

The remaining variables read four random compounds from the query result (“c1” to “c4”), extract their names (“p1” to “p4”) as well as the structural identifier for one of them in variable “n1”. The latter is used subsequently in the definition of a dynamic object that turns the identifier into its graphical representation (see Figure 9).

A similar visual result can be achieved by using image links that are available in Wikidata as well and that link to images of chemical compounds that are located at Wikimedia Commons. However, that has several major drawbacks: First, image links are ultimately resolved by the client browser and thus independent of the e-assessment system. Any failure in resolving the links may result in an incomplete item on the client side without any chance for the e-assessment system to notice. With dynamic objects, the whole output is prepared by the e-assessment system and any failure in rendering the output can be logged. Second, Wikimedia may block large numbers of requests for any reason, especially if they appear within a short period of time from a single IP location, as it would be the case in exam situations. With dynamic objects, no third-party server is involved that can block requests. Third, images in Wikimedia Commons may or may not show the molecule

Figure 8: Authoring view on the item variables used in conjunction with the item template in Figure 7.

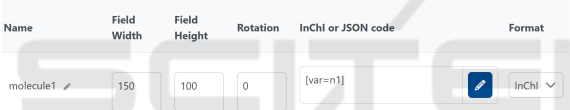

Figure 9: Authoring view on the dynamic object that renders the graphical representation of the molecule in Figure 6.

in the expected notation. With dynamic objects, item authors can be sure that all molecules will be shown in the same notation. Finally, files loaded from Wikimedia or any third-party server may have file names that reveal solutions to students how are able to inspect the file names with their browser tools. With dynamic objects, molecules and alike is rendered directly within the final output and no file names can be seen.

4 SOFTWARE STRUCTURE

Item variables and dynamic objects are handled differently within the e-assessment system for technical reasons, although both can ultimately be used to fill placeholders in an item template.

4.1 Data Types

There is only one generic data type for item variables that holds the variable name and the corresponding evaluator function. The latter is a string following some expression language that allows to make references to function names, variable names and constants. At runtime, these strings are passed to a dedicated evaluator component that parses the string, calls the required functions and returns the resulting value.

Dynamic objects are represented by individual data types due to their larger complexity. While they share a name, a width, and a height as common attributes, their individual content definition can be very different.

4.2 Template Instantiation Process

When a student starts to interact with an assessment item, a new instance of an object representing that item is created (even if it is a static item where no item generation takes place). Then, the evaluator functions for all variables are called, and the resulting values are stored within the object representing the item. The object is persisted, so all values are available for later manual inspection if necessary.

Then, the item contents that need to be presented to the student are looked up in the item definition. It depends on the item type which contents are relevant here. Usually, there will be some item stem or prompt. In multiple choice items, there will also be answer options that will not be present in other item types. In turn, some other item type may provide additional files for download to the student.

In a first pass, each content element is scanned for placeholders that refer to dynamic objects. Each occurrence of a placeholder will be replaced by a new instance of the respective dynamic object. Notably, the content definition for a dynamic object may include references to item variables, which need to be resolved first.

In a second pass, each content element is scanned for remaining placeholders that refer to item variables. These are replaced by the respective variable values, including graphical conversions like LaTeX formatting for mathematical formulas. Authors can add flags to each placeholder to indicate which kind of conversion they would like to use in each individual place.

4.3 Extensibility

The software structure presented above includes three dedicated interfaces for extensions. First, new evaluator functions can be added that define new ways to compute or derive variable values. Second, new types of dynamic objects can be added that render new types of graphical output. Third, new conversion flags can be added that allow to present variable val-

ues in different, potentially domain-specific formatting. Following the software structure, making use of these interfaces happens in three distinct places.

Evaluator functions are added within the evaluator component. Each new function is implemented in a separate code file and registered in a dictionary of available function names. Functions may be simple with just one line of code or may be complex, e. g. including calls to external software libraries. As demonstrated above, functions even can make calls to external databases and could make calls to AI-powered services and alike in a similar manner. Since they are implemented in the dedicated evaluator component, they can be added and updated independently of the actual e-assessment system. Thus, new functions for item generation can be added on-the-fly without the need to update the entire system.

Dynamic objects are added within the core of the e-assessment system, since they are more complex and provide individual settings that must be stored in the system's database. Adding a new type of dynamic objects thus requires to implement some technical interface, to define the object's properties and to define the output rendering for that object. In addition, input elements in the authoring interface must be defined so that teachers can actually make use of the new objects. Consequently, adding new dynamic objects is not as easy as adding new evaluator functions and requires to update and restart the entire e-assessment system.

Conversion flags are added within the converter component, which is a separate component similar to the evaluator component. However, flags are not independent of each other and thus there is a single conversion algorithm that handles all flags at once. That algorithm needs to be extended if new flags are required. Similar to the evaluator component, the converter component can be updated independent of the actual e-assessment system and thus provide new conversions at any time.

5 CURRENT USAGE AND LIMITATIONS

The approach is fully operational and in use. Among others, the mathematics department at University of Duisburg-Essen currently maintains a pool with about 300 item templates that are used for item generation for homework exercises. The economics department of the same university maintains smaller pools of item templates for various assignments in microeconomics as well as for homework exercises and exams in statistics. These pools have already been used with an older version of JACK (Massing et al., 2018) and helped to design the current software structure.

Although the approach is fully operational, it comes with three notable limitations.

On the technical level, item variables are somewhat limited in the amount of data they can handle efficiently. While it is technically possible to create an evaluator function that e. g. generates a large image and returns it in Base64 encoding, it is not efficient to do so, because the variable value would then be a very long string that must be stored in a single field in the system's database. Using compressible SVG may help here and add additional options for styling the image on the client-side. Still, that also does not solve the general problem that a single database field is not the optimal choice to store hundreds of kilobytes of data in a single string. That does not only apply to images, but also to evaluator functions that may e. g. return thousands of result entries for a single query to an external database.

The latter example may also have an impact on user experience. As defined in the requirements in Section 2, item generation happens online right before an item is displayed to a student. Hence, item generation is limited to cases in which the processing of evaluator functions can be completed in a reasonable short period of time. This applies both to the complexity of calculations and the delay created by connecting to external data sources or alike. The more item variables are defined in an item, the faster each variable value must be computed to achieve an overall acceptable user experience. Experience shows that waiting for about 5 seconds for an item to be loaded is acceptable for most students in homework exercises, but already causes distress for some students in exam situations.

On the conceptual level, the whole process of item generation is performed without manual supervision. There is no intermediate step in which generated items are reviewed by human experts before they are displayed to students. Hence, the applicability of the approach is limited to cases in which either a high item quality can be guaranteed by careful item template design and careful checking of all data sources, or in which a reduced item quality is acceptable as it may be the case in some specific formative assessment scenarios. In particular, students and teachers must be aware that low quality items can appear and students must have options to skip such items if they cannot be answered in a meaningful way. Notably, all generated items including all variable values are available in the e-assessment system for later manual inspection, so that quality checks can be made in retrospective. Based on these checks, item definitions can be amended in order to improve item generation.

6 CONCLUSIONS AND FUTURE WORK

This paper presented the automatic item generation concepts within the e-assessment system JACK. The focus was not on new capabilities in item generation, but on the software design. In particular, the concept can be considered more sustainable than common standalone, single-purpose solutions, as it is integrated, reusable and extensible:

- The concept is *integrated*, because item generation happens directly within an e-assessment system that is also responsible for delivering assessment items to students and for automatic grading and feedback generation. Thus, there is no need to create an ecosystem around the tool as it might be the case with other approaches.
- The concept supports *reusability*, because any of its features is realized in a distinct component that can be combined freely. The core concepts of item variables and dynamic objects are generic for assessment items and can be used regardless of the item types of their parts.
- The concept supports *extensibility*, because each component can be updated without interfering with the remaining system. There is no need to build a new system just for new types of dependencies between item variables. Instead, evaluator functions can be added as needed. There is also no need to build a new system just for new types of content elements. Instead, evaluator functions or dynamic objects can be added as needed. There is also no need for a new system just because of different representations. Instead, dynamic objects or output conversions can be added as needed.

Besides a general extension of JACK's item generation capabilities in terms of new evaluator functions, future work particularly includes the inclusion of a new type of dynamic objects for data structures in computer science education. While visualizing data structures in general is not a hard problem, it puts more emphasis on the internal structure of item variables (e. g. it might become important in what order a list contains elements).

A current research project tackles the automatic generation of questions on program code submitted by students in response to programming assignments. While the research project is concerned with more fundamental aspects of asking questions about program code, the creation of practical demonstrators may require to extend the current item generation capabilities of JACK with specific features for handling program code.

REFERENCES

Baum, H., Damnik, G., Gierl, M., and Braun, I. (2021). A shift in automatic item generation towards more complex tasks. In *INTED2021 Proceedings*, 15th International Technology, Education and Development Conference, pages 3235–3241. IATED.

Choi, J., Kim, H., and Pak, S. (2018). Evaluation of Automatic Item Generation Utilities in Formative Assessment Application for Korean High School Students. *Journal of Educational Issues*, 4(1).

Choi, J. and Zhang, X. (2019). Computerized Item Modeling Practices using Computer Adaptive Formative Assessment Automatic Item Generation System: A Tutorial. *The Quantitative Methods for Psychology*, 15(3):214–225.

Christ, P., Munkelt, T., and Haake, J. M. (2024). Generalized Automatic Item Generation for Graphical Conceptual Modeling Tasks. In *Proceedings of the 16th International Conference on Computer Supported Education, CSEDU 2024, Angers, France, May 2-4, 2024, Volume 1*, pages 807–818. SCITEPRESS.

Circi, R., Hicks, J., and Sikali, E. (2023). Automatic item generation: foundations and machine learning-based approaches for assessments. *Front. in Educ.*, 8.

Foulonneau, M. and Ras, E. (2013). Using Educational Domain Models for Automatic Item Generation Beyond Factual Knowledge Assessment. In *Proceedings of EC-TEL 2013: Scaling up Learning for Sustained Impact*, pages 442–447. Springer Berlin Heidelberg.

Gierl, M. J. and Lai, H. (2012a). The Role of Item Models in Automatic Item Generation. *International Journal of Testing*, 12(3):273–298.

Gierl, M. J. and Lai, H. (2012b). Using Weak and Strong Theory to Create Item Models for Automatic Item Generation: Some Practical Guidelines with Examples. In *Automatic Item Generation*. Routledge.

Kosh, A. E., Simpson, M. A., Bickel, L., Kellogg, M., and Sanford-Moore, E. (2019). A Cost–Benefit Analysis of Automatic Item Generation. *Educational Measurement: Issues and Practice*, 38(1):48–53.

Kıyak, Y. S. and Kononowicz, A. A. (2024). Case-based MCQ generator: A custom ChatGPT based on published prompts in the literature for automatic item generation. *Medical Teacher*, 46(8):1018–1020. PMID: 38340312.

Liu, B. (2009). SARAC: A Framework for Automatic Item Generation. In *2009 Ninth IEEE International Conference on Advanced Learning Technologies*, pages 556–558.

Massing, T., Schwinning, N., Striewe, M., Hanck, C., and Goedicke, M. (2018). E-Assessment Using Variable-Content Exercises in Mathematical Statistics. *Journal of Statistics Education*, 26(3):174–189.

Mortimer, T., Stroulia, E., and Yazdchi, M. V. (2012). IGOR: A Web-Based Automatic Item Generation Tool. In *Automatic Item Generation*. Routledge.

Striewe, M. (2016). An architecture for modular grading and feedback generation for complex exercises. *Science of Computer Programming*, 129:35–47.