Meclib: Dynamic and Interactive Figures in STACK Questions Made Easy

https://doi.org/10.3991/ijet.v17i23.36501

Martin Kraska(✉)
Brandenburg University of Applied Sciences, Brandenburg an der Havel, Germany
kraska@th-brandenburg.de

Abstract—Making and interpretation of drawings and schematics is an important skill to be developed in engineering education. The question type STACK in the Moodle learning management system combined with the interactive graphics library JSXGraph has a great potential for implementation of e-learning resources addressing these skills in formative and summative settings. The authoring process of such materials is complex due to multiple markup and programming languages like HTML, LaTeX, Maxima and JavaScript (JS) being involved. The Meclib concept mitigates this complexity and allows for efficient bulk production of material with consistent appearance and user experience. The core of Meclib is a set of pre-defined JSXGraph-based objects with an interface to Maxima, such that no problem specific JS code is required at all. The present paper outlines the basic ideas of the implementation and demonstrates the approach for some typical examples of different complexity. The focus is on applications in engineering mechanics, starting from static illustrations up to an editor for free body diagrams, with rich adaptive formative feedback.

Keywords—STACK, JSXGraph, engineering mechanics, automatic assessment, interactive graphics, free body diagram

1 Introduction

The increasing heterogeneity of skills and knowledge of engineering students challenges the classical concept of lectures and classroom exercises, even if augmented by weekly homework assignments. The use of learning management systems has brought options for asynchronous formative and summative assessment, which can support the learning process by providing a self-paced training ground and continuous self-assessment. An important aspect of such resources is the immediate automatic feedback on student input, which can go far beyond just indicating right or wrong.

Another aspect is the randomization of questions, which is important both in formative assessment (exercises for self-learning) and in summative assessment (auto-graded homework assignments or exams). Randomization of questions, including dynamic graphics is preferable over multiple different similar questions mainly from the perspective of maintenance. Improvements or bug fixes are easier to apply to a single

randomized questions than to a series of questions, in particular, if editing requires multiple interactions with a web interface.

The tool chain Moodle/STACK/JSXGraph provides a strong base for such e-learning resources [1], [2]. The focus of the present paper is on how to efficiently create auto-graded questions with interactive and non-interactive drawings of mechanical systems.

Section 2 describes the STACK question type as far as required for the understanding of the Meclib concept, which is outlined in section 3. Usage examples of different complexity levels are presented in section 4. In the discussion (section 5) the current state is summarized and possible further developments are outlined.

2 STACK

STACK (System for Teaching and Assessment of mathematics using a Computer algebra Kernel) is a question type plugin for the Moodle learning management system. Based on the use of the computer algebra system Maxima, the question type can establish mathematical properties of symbolic expressions to derive specific formative feedback [3], [4], [5]. STACK comes with an integrated interface to JSXGraph, a JS library for embedding interactive SVG-based graphics into web pages [6].

Even though STACK was developed with focus on mathematics, it's built-in support for physical units and for interactive graphics make it interesting for application to other fields in engineering education [7], [8].

Authoring STACK questions involves the following steps:

- Definition of problem variables ("question variables") with eventual randomization and calculation of the model answer. This requires knowledge of the Maxima computer algebra system.
- Writing the question text with injection of problem variables (given values) and definition of input fields. This requires knowledge of LATEX and, eventually, HTML.
- Configuration of the behavior of the input fields.
- Setup of potential response trees (involving answer tests and specifying feedback in the individual branches).
- Specification of answer test and deployment of randomized question variants.

Like many other Moodle activities, STACK comes with a web interface for authoring. For the purpose of efficient code editing, the integrated editors are far off the user experience provided by state of the art integrated development environments.

This is a clear disadvantage with respect to other learning tools like PrairieLearn [9], which uses XML-Files (supported by many editors), or STEMCStudio [10], which comes with a comfortable JavaScript editor with syntax highlighting and code introspection. Questions produced by these tools can be integrated in Moodle courses via an LTI (Learning Tool Interoperability)-interface [11]. They, however, lack the power of the elaborated computer algebra based feedback system of STACK.

Other approaches focus on efficient external editing and testing of JSXGraph applets using integrated development environments (IDE) like Visual Studio [12] or

the web based jsfiddle [13]. Some researchers even create code generators using sets of pre-defined parametrized high-level JSXGraph objects. The MUMIE platform [14] uses LaTeX style instructions to generate JSXGraph visualizations.

A series of courses in Engineering Mechanics can easily involve several hundreds of multipart (with multiple input fields) STACK questions. If a considerable part of these is to be equipped with static or interactive JSXGraph based graphics, then there is no reasonable alternative to one of the above-mentioned productivity approaches.

3 Meclib

The Meclib concept presented in this paper is novel in that it allows the question author to design dynamic, interactive JSXGraph applets with consistent appearance and user experience with a few lines of Maxima code from within a STACK question.

This mitigates the need for the question author to touch JavaScript at all. It does not add complexity because Maxima is used in STACK anyways. While the concept initially involved direct insertion of a problem-independent block of JavaScript code, recently it became possible to inject Maxima or HTML/JavaScript code from external web sources into STACK questions.

The core resource is the Meclib GitHub repository [15], containing JS code for object definitions and Maxima functions to support rich adaptive feedback. The wiki of the repository is a comprehensive reference for working with Meclib.

3.1 Basic concept

The basic idea of the Meclib concept is to entirely setup JSXGraph figures in STACK questions from within the question variables section using a list of expressions in Maxima format, see Figures 1 and 2. The actual figure still is generated by an embedded block of JavaScript code, yet this code is completely problem-independent.

Since STACK version 4.4 (2022), both the question variables and page content (HTML, LaTeX, JS) can receive dynamically injected content from external web sources. Previous Meclib usage involved static verbatim pasting the file contents in the STACK editor web interface. Dynamic injection only happens at compile time of a STACK question, when saving edits. Thus, at question usage time, there is no traffic from the external source.

Static embedding has the advantage to protect questions against regressions by changes of the external source. Yet this also prevents improvements in the external source from being applied.

Dynamic embedding includes the intrinsic danger of breaking existing questions when updating Meclib. Efficient bulk testing of large collections of questions is still an open question.

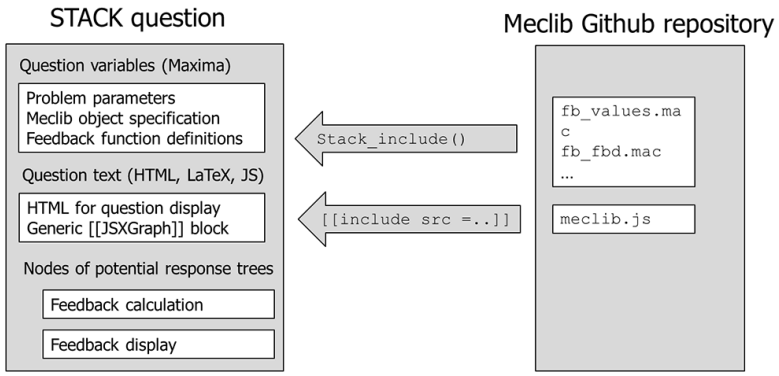


Fig. 1. The Meclib Github repository [15] provides generic, problem independent definitions to simplify writing STACK questions with embedded JSXGraph applets or illustrations

3.2 The Maxima-JSXGraph interface

The list of object specifications is stored in a Maxima variable, examples are shown in Figures 5–9. This variable is converted to a JSON string and conveyed to the JSXGraph block using standard STACK preprocessor tags like {#…#}. This essentially is a unidirectional interface for static applets and for definition of the initial state of interactive applets.

Bi-directional data exchange between the embedded JSX figure and the parent STACK question works via shared access to DOM elements (hidden input fields), which is another standard mechanism supported by STACK. The input fields are populated as soon as the attached JavaScript variable is modified by some interaction with the graphics. Two input fields are used: one text field to store the complete state information of the graphics and one for auxiliary algebraic expressions to support feedback, see Figure 2.

```
<p hidden>[[input:objects]] [[validation:objects]]</p>
<p hidden>[[input:names]] [[validation:names]] </p>
[[jsxgraph width='500px' height='400px' input-ref-
objects="stateRef" input-ref-names="fbd_names" ]]
  var mode  = "STACK";
  const initstring = {#init#};
[[include
src="https://raw.githubusercontent.com/mkraska/meclib/
main/meclib.js" /]]
[[/jsxgraph]]
```

Fig. 2. Question text for an interactive JSXGraph block in STACK, using text injection from the init variable (red) and a bi-directional link via reference to DOM contents (green). The Meclib source is inserted via external link (blue)

This split is required because the built-in Maxima doesn't execute injected code from string input by security reasons. Yet, for the application to free body diagrams as described in section 4.3, the extraction of variable names for further processing in the CAS kernel is mandatory.

The essential components of the generic Meclib code block are

- Configuration settings
- Board instantiation
- Class definitions for the available objects, see Figure 3
- Initialization function to instantiate the objects either from the contents of the hidden input field or from the injected question variable.
- Update function to populate the algebraic input field and modify the text input field, when the graphics has changed by user interaction.
- Helper functions to streamline the code.

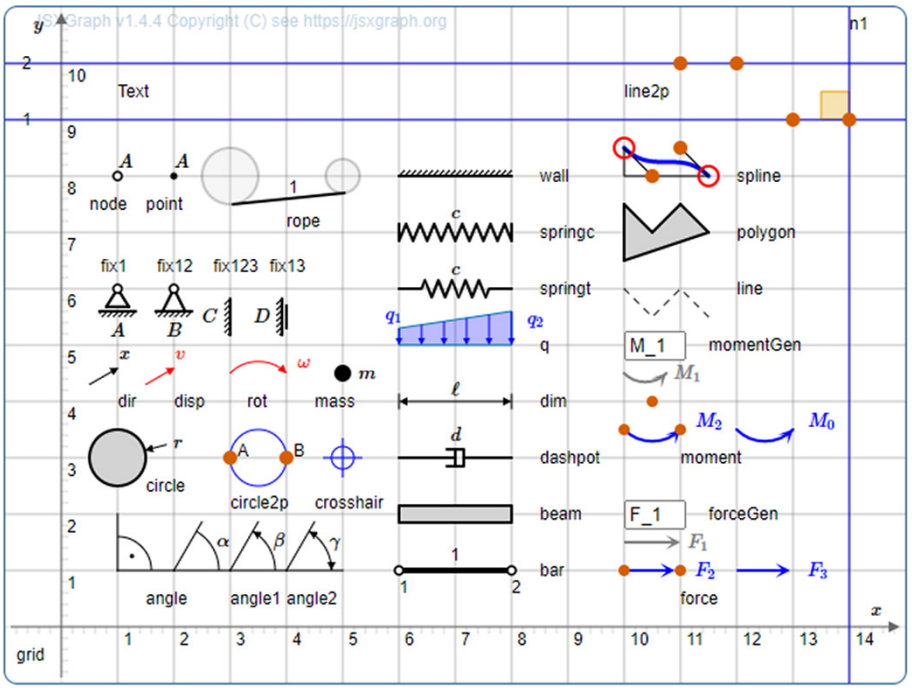


Fig. 3. Available objects and their names. Red dots indicate interactive control points

3.3 Rapid prototyping

Creating embedded graphics in STACK question using Meclib involves copying a short code snippet like in Figure 2 to the question variables, setting up the hidden input fields and specifying the contents via a Maxima variable. All this is done in the STACK editing web interface. Testing the image requires saving and previewing the question,

which can take several dozens of seconds, since the question text and Maxima code including the external links are checked and compiled for later security and better user experience (performance).

Jsfiddle is a web-based development environment, which is very popular in the JSXGraph community. Meclib has been designed such that the JS code works likewise in jsfiddle and in STACK. This includes the communication via shared HTML page elements. This allows to preview the actual contents of the hidden input fields used for data exchange with STACK [16]. Figure 4 shows the prototype version of the example in Figure 5.

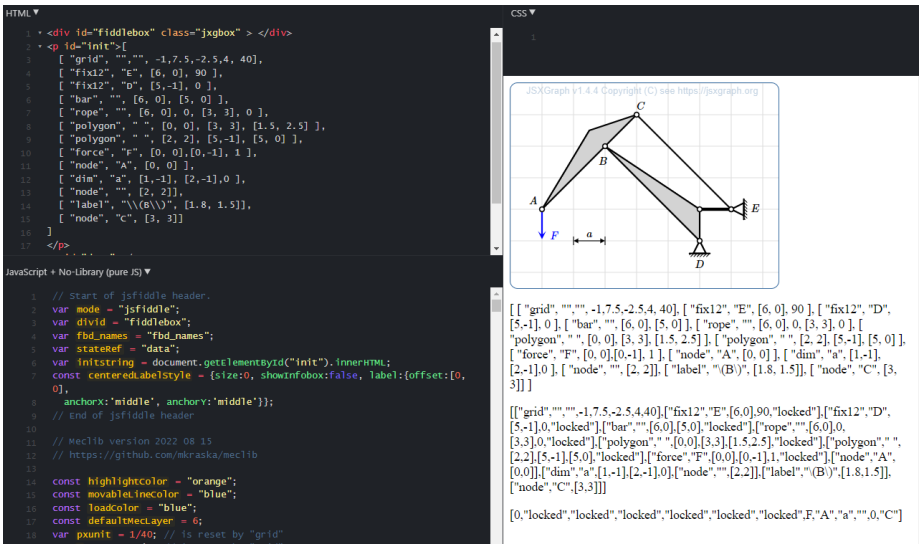


Fig. 4. Jsfiddle configured for rapid prototyping and debugging of Meclib graphics [16]. In the upper left pane, the list of objects can be edited. The right panes provide an interactive preview to the applet and to the data transferred to STACK

4 Application

The applications shown here illustrate the versatility of the Meclib concept. Focus is on the JSXGraph applets and the Maxima code defining them. The STACK questions around the applets aren't shown. The last example is an editor for free body diagrams, where also some capabilities for adaptive feedback are demonstrated.

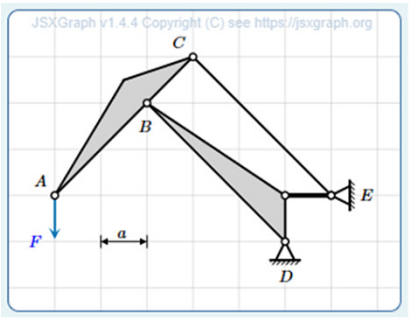
4.1 Static illustrations in STACK questions

Generation of non-interactive (static) but still randomizable illustrations in STACK questions is the most basic use case of Meclib. The unidirectional interface via text injection is sufficient, no hidden input fields are needed. Accordingly, the JSXGraph block in the question text is slightly simpler than shown in Figure 2, details can be found in [15].

Without Meclib, static illustrations usually are generated externally in some graphics program or taken from any other image source. Randomization can be obtained by saving multiple versions of a question and using the random picking mechanism in the Moodle test activity. Also, random picking of multiple embedded image files has been reported [7]. In the Meclib approach, the image is defined by a list of object specifications, see Figure 5. Note the human-readability and the compactness of the representation. Randomization of this Maxima expression is straightforward.

Meanwhile most of the several hundred existing STACK questions for the author's courses on engineering mechanics contain such Meclib images. Many of those have been created by student assistants. The wiki on the Meclib GitHub repository [15] provides comprehensive information and illustrated examples for the available objects.

Even though authors might miss the comfort of graphics programs, the Meclib approach ensures consistent appearance of the material. Also, the images can be modified in the STACK web editor without requiring external tools and there are no image-related copyright issues.




```
A: [0, 0]; B: [2, 2];  C: [3, 3]; E: [6, 0];
D: [5,-1]; P1: [5, 0]; P2: [1.5, 2.5];
th: 0.13;
initdata: [
  [ "grid", "","", -1,7.5,-2.5,4, 40],
  [ "fix12", "E", E, 90 ],
  [ "fix12", "D", D, 0 ],
  [ "bar", "", E, P1 ],
  [ "rope", "", E, 0, C, 0 ],
  [ "polygon", " ", A, C, P2 ],
  [ "polygon", " ", B, D, P1 ],
  [ "force", "F", A, A-[0,1], 1 ],
  [ "node", "A", A ],
  [ "dim", "a", [1,-1], [2,-1],0 ],
  [ "node", "", B],
  [ "label", "\\(B\\)", B+[-0.2,-0.5]],
  [ "node", "C", C]
];
```

Fig. 5. Non-interactive illustration for a STACK question along with the corresponding Maxima list

4.2 Simple interactive input

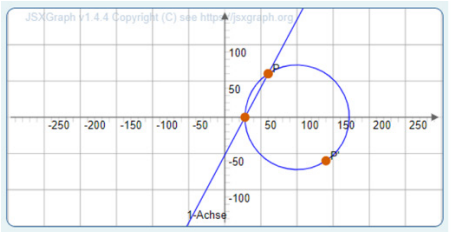
Interactive questions, were STACK evaluates the position of objects in the JSXGraph applet, are state of the art. The procedure is well documented on the STACK website [1]. For data exchange between applet and STACK, hidden *numeric* input fields are used (Meclib uses text and algebraic input). Integrating such applets into STACK question involves just a few (in very simple cases) up to several dozens of lines of JS code.

Yet even such applications benefit from using Meclib. This is illustrated with two examples.

Figure 6 shows an applet for construction of Mohr's circle. This is a visualization or graphical solution method for eigenvalue problems of two-dimensional real symmetric matrices. In engineering mechanics, this method can be used to establish or verify the principal values and the corresponding directions for plane stress and strain states as

well as for section moments of inertia from their Cartesian components with respect to some reference frame.

The problem-specific part of the graphics specification is just six lines of text.



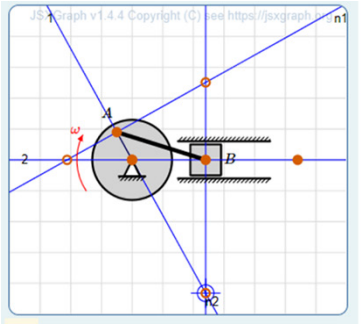

```
f:50;
initdata: [
  [ "grid", " ", " ", -6,6, -3, 3, 40, [f,f] ],
  [ "circle2p", "P","P'", [50,50],[100,100], f ],
  [ "line2p", "1-Achse", [200,100],[250,50], f ]
];
```

Fig. 6. Interactive applet for construction of Mohr's circle from given tensor components. The task is to correctly place the circle and the line for the first principal direction (eigenvector)

A more complex example is shown in Figure 7. Here, the student has to find the center of rotation of the connection rod in a crank and piston mechanism. The solution is obtained by recognizing the direction of movement at two points (here A and B) and construction of the perpendiculars to them. The center of rotation is at the intersection of these perpendicular lines.

Therefore the applet provides helper objects (lines with perpendiculars). Their control points snap to the grid and to key points of the drawing. The crosshair additionally snaps to the intersection points of the helper lines.

Currently, some additional lines of JS are required to mark the intersection points and add them to the snap targets of the crosshair. Alternatively, snapping could be omitted, perhaps with relaxed requirements to the precision of the answer.




```
initdata: [
  [ "grid", "","", minx, maxx, miny, maxy, 30, [1,1], [2,2] ],
  [ "line2p","1",pL, pL+[-1,0 ], "n1"],
  [ "line2p","2",pL+[1,1], pL+[0,1], "n2"],
  [ "crosshair", "", pL+[-1,-1] ],
  [ "circle", "", O, rr+0.3 ,0],
  [ "bar", "", A, B ],
  [ "beam", "", [xB-th, 0], [xB+th,0], th ],
  [ "fix12", "", O, 0 ],
  [ "wall", "", [ll-rr-th, th+0.1], [ll+rr+th,th+0.1] , 45 ],
  [ "wall", "", [ll+rr+th,-th-0.1] , [ll-rr-th, -th-0.1], 45 ],
  [ "label", "\\(A\\)", A+[-0.5,0.6] ],
  [ "label", "\\(B\\)", B+[0.6,0 ]  ],
  [ "line", "", [0, xA], [0, yA] ],
  [ "rot", "\\omega", O , [-1.5, -1 ], [-2, 1] ]
];
```

Fig. 7. Interactive applet for finding the center of rotation of the connection rod in the crank and piston mechanism. Interactive helper lines are provided as construction tools

4.3 Editor for free body diagrams

Right from the start, Meclib was developed with the automatic assessment of free body diagrams (FBD) in mind. These are idealized drawings of mechanical systems, where the system is isolated by removing the environmental components and replacing their action by forces and moments (constraint reactions). Only after this step, equilibrium

conditions can be established, which then are solved for the unknown reactions. Therefore, drawing correct FBDs is an important skill in engineering mechanics [17].

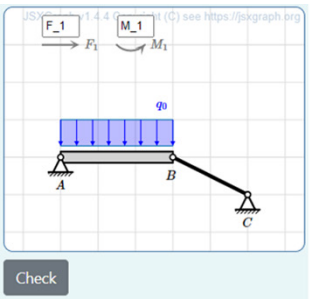
In [8] the use of STACK and JSXGraph for learning resources in engineering mechanics and electrical engineering has been reported. The questions partly had embedded JSXGraph applets, yet they admittedly did not include feedback on FBDs.

Based on observations of typical mistakes in written exams and classroom exercises, the following requirements and design decisions for an FBD training and assessment tool have been established by the author:

1. The system schematic should preferably be built of standard elements as shown in Figure 3.
2. System isolation is visualized by graying out elements of the environment (connectors, supports), thus not a new sketch is made but the FBD is created in place.
3. The user can create, delete and label force and moment symbols.
4. Proximity checks between deactivated parts of the environment and introduced force and moment symbols are established using standard features of JSXGraph.
5. Intelligent snap mechanisms enable precise positioning and orientation of reactions, even if that might give unwanted hints.
6. There should be rich formative feedback for the individual decisions made while creating an FBD.
7. Based on the force and moment symbols introduced by the user, the reference solutions for equilibrium conditions are auto-generated and used for feedback on student solution. This is why labelling and precise positioning are required.

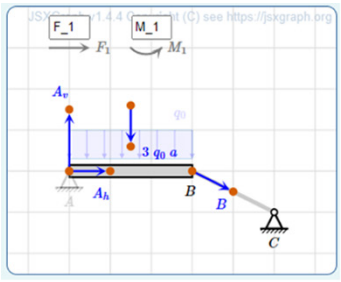
In particular, the last requirement initially seemed non-trivial, because the standard interface between STACK and JSXGraph doesn't allow for extraction of symbolic expressions from text input fields. The feasibility of introducing an additional algebraic input field as shown in Figure 2 has been demonstrated in [1].

Figures 8 and 9 show the idealized initial system and the completed FBD along with their respective representation on Maxima side. The elements *forceGen* and *momentGen* at the top of the canvas are generators for force and moment symbols. In the text fields, the label can be entered. The object is created by dragging the gray symbols away from their position. Elements of the system can be deactivated by double-clicking.




```
initdata: [
  [ "grid", "", "", -1.5, 6.5, -2.5, 4, 40],
  [ "forceGen", "F_1", [-0.5,3.5]],
  [ "momentGen", "M_1", [1.5,3.5]],
  [ "beam", "", pA, pB, 0.15],
  [ "bar", "", pB, pC, "show" ],
  [ "fix12", "A", pA, 0, "show" ],
  [ "fix12", "C", pC, 0, "show" ],
  [ "label", "\\(B\\)", pB+[-0.2,-0.5] ],
  [ "q", "", "q_0", pA+[0,0.3], pB+[0,0.3], 0.7,0.7,0, "show"]
];
```

Fig. 8. Initial system schematic and Maxima list of object specifications. The task is to replace the supports by appropriate reactions and the distributed load by a resultant force. Whenever the button Check is pressed, feedback is generated




```
tansdata: [
  ["grid","","",-1.5,6.5,-2.5,4,40],
  ["forceGen","F_1",[-0.5,3.5]],
  ["momentGen","M_1",[1.5,3.5]],
  ["beam","",[0,0],[3,0],0.15,"locked"],
  ["bar","",[3,0],[5,-1],"hide"],
  ["fix12","A",[0,0],0,"hide"],
  ["fix12","C",[5,-1],0,"show"],
  ["label","\\(B\\)",[2.8,-0.5]],
  ["q","","q_0",[0,0.3],[3,0.3],0.7,0.7,0,"hide"],
  ["force","3 q_0 a",[1.5,1.6],[1.5,0.6],10,"active"],
  ["force","A_v",[0,0],[0,1.5],10,"active"],
  ["force","A_h",[0,0],[1,0],10,"active"],
  ["force","B",[3,0],[4,-0.5],10,"active"]
];
```

Fig. 9. Reference solution and the corresponding Maxima list of objects. This list is specified as model solution just for display purposes (review by trainer, worked solution for the student). Note that there are many other correct solutions, such that the assessment must go far beyond a simple comparison

Figures 10 and 11 show intermediate states with the respective feedback. Meclib has specific feedback functions for most support objects. According to the above-mentioned design decision, proximity checks between supports and reactions (or distributed load and resultant) are performed on JSXGraph side using its powerful object methods. The feedback functions aren't internationalized yet and therefore in German. However, the figure captions give summaries.

As an example, the checks performed for a fixed support (as in point A) are:

1. Is the object really of type *fix12*? This is just for question developers.
2. Is the object deactivated?
3. Any reactions reported from the proximity check?
4. Are there exactly two reactions?
5. Are both reactions of type *force*?
6. Are the names of the reactions unique (not identical)?
7. Are the directions of the two reactions different?

Further checks for the names are performed in a separate function to allow for partial credit.

8. Is the base name a single character?
9. Does the base name match the name of the support point?
10. Is there a direction index?
11. Is the index a single character?
12. Is the index one of x, y, h(orizontal) or v(ertical)?
13. Does the index match the actual direction of the force?

The sequence of tests is designed such that whenever a test fails, the appropriate feedback is generated and further tests are skipped, because they would not make sense.

Using complex feedback functions allows for extremely compact potential response trees in the STACK question. The adaptive question mode in STACK allows the user to retry until all requirements are met.

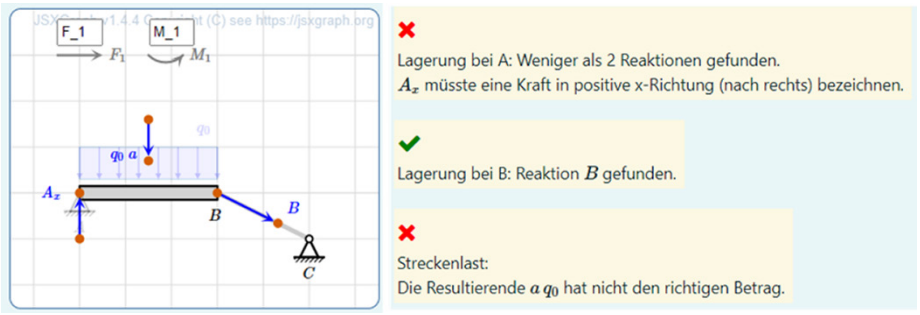


Fig. 10. Intermediate state: The distributed load and the supports have been deactivated, yet the created resultant doesn't have the correct value. Support B has been replaced by a correct reaction with an appropriate label. Support A needs more reactions and the subscript x isn't appropriate for vertical direction

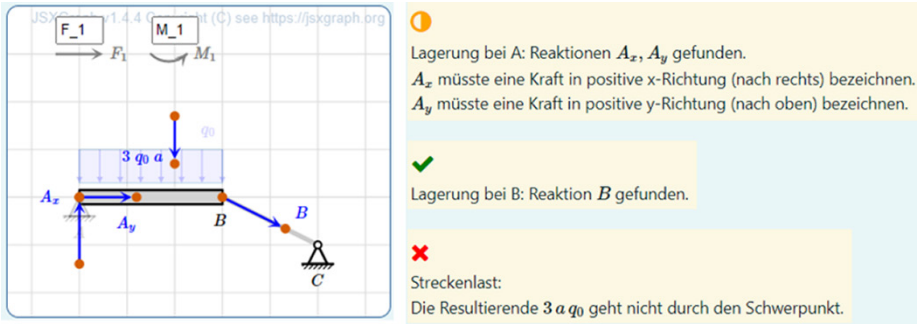


Fig. 11. Intermediate state: The resultant doesn't go through the centroid of the distributed load. Support A is now replaced by correct reactions, yet the labels are inconsistent

5 Discussion

Meclib has been designed to support the development of e-learning resources with interactive graphical input within the Moodle/STACK frame using the JSXGraph library. The focus was on applications for courses of engineering mechanics and the key application was an editor for free body diagrams with rich adaptive formative feedback. The STACK documentation recommended the concept of state storage and data exchange based on shared access to hidden text input fields. First feasibility studies indicated that this had to be extended by an algebraic input field due to security restrictions in the STACK-embedded Maxima kernel [1].

Meanwhile, a library of approximately 40 objects has been defined, which serves as a construction set for high quality, visually consistent learning materials. It soon became obvious, that this library was more than just a step towards the advanced interactive applications in mind [2]. In fact, currently the most extensive application are non-interactive static illustrations (several hundred) as described in section 4.1.

Rich adaptive formative feedback is a particular strength of the STACK platform, while efficient editing of the feedback behavior is not. Propagating elaborated solutions from one question to the other is very time consuming in the STACK web interface. Therefore, in parallel to dynamic and interactive graphics, the Meclib approach increasingly involves complex special purpose feedback functions. Figures 10 and 11 give an impression of what is possible with just a few function calls. This, however, is not the focus of the present paper and will be discussed elsewhere.

Another use case is presented in section 4.2. It involves questions, where the user interacts with existing movable objects without creating new ones. There are many other applications besides the two examples shown in Figures 6 and 7, e.g. drawing motion graphs in kinematics or shear-moment diagrams in statics.

The FBD editor as top end application has been shown in section 4.3. There are still some missing links. The feedback functions don't yet cover all types of support and contact situations. Another vector for further development is the integration into complex multipart questions where the FBD is just the initial step. Establishing the equations of equilibrium and solving for the support reactions are the subsequent ones. Currently, such tasks start with given FBD, such that the equations and solutions are unique and assessment is easy. Editor-created FBDs, aren't unique, even with rather strict naming conventions. Therefore, consistency between equations and FBD is an additional aspect of assessment.

Meclib has been actively presented to the JSXGraph and STACK communities [1], [2]. It has encouraged important extensions and performance improvements in the STACK kernel and has become an informal benchmark application. The time for feedback generation after pressing the Check button in a question with the interactive FBD editor has reduced from nearly 20 seconds to just 1–2 seconds.

The central platform for development and documentation of Meclib is the Meclib Github repository. The wiki has become a comfortable and comprehensive reference [15].

6 Acknowledgment

This work has been supported by the Department of Engineering at Brandenburg University of Applied Sciences by funding student assistants. These assistants, in particular Dennis Schulz and Niclas Kopp, contributed by testing and authoring questions, preparing prototypes of graphics and sometimes even by contributing new objects.

The STACK and JSXGraph development teams provide the essential software tools for this project. Chris Sangwin and Matti Harjula (STACK) and Alfred Wassermann contributed ideas and guidance.

7 References

[1] M. Kraska, "Concept for automatic assessment of free body diagrams", In 1. International JSXGraph Conference. University of Bayreuth, Germany, Eds. C. Miller and A. Wassermann, 2020.

[2] M. Kraska, D. Schulz, "Automatic assessment of free body diagrams using STACK", In Contributions to the International STACK conference 2021. [Online]. Available: http://doi.org/10.5281/zenodo.4916138. [Accessed October 8, 2022]

[3] J. C. Sangwin, *Computer Aided Assessment of Mathematics*. Oxford, UK, Oxford University Press, 2013. https://doi.org/10.1093/acprof:oso/9780199660353.001.0001
[4] STACK, System for teaching and assessment of mathematics using a computer algebra kernel, *stack-assessment.org*, 2022. [Online]. Available: https://stack-assessment.org. [Accessed October 8, 2022]
[5] Maxima. A computer algebra system, *maxima.sourceforge.io*, 2022. [Online]. Available: https://maxima.sourceforge.io. [Accessed October 8, 2022]
[6] JSXGraph, Dynamic mathematics with JavaScript, *jsxgraph.org*, 2022. [Online]. Available: https://jsxgraph.org. [Accessed October 8, 2022]
[7] M. T. T. Neitola, "Circuit theory e-assessment realized in an open-source learning environment", *Int. J. Eng. Ped.*, vol. 9, no. 1, pp. 4–18, Feb. 2019. https://doi.org/10.3991/ijep.v9i1.9072
[8] C. Klischat, P. Becker, and M. Vasko, "STACK is more than maths – development of online problems for mechanics and electrotechnics", In Contributions to the 1st International STACK conference 2018, Fürth, Germany. Zenodo, 2019.
[9] PrairieLearn learing management system, documentation, 2022. [Online]. Available: https://prairielearn.readthedocs.io/en/latest/. [Accessed October 9, 2022]
[10] STEMCstudio learning management system, *stemcstudio.com*, 2022. [Online]. Available: https://www.stemcstudio.com/. [Accessed October 9, 2022]
[11] 1EdTech Consortium. Learning tools interoperability, *imsglobal.com*, 2022. [Online]. Available: http://www.imsglobal.org/activity/learning-tools-interoperability. [Accessed October 9, 2022]
[12] Microsoft, Visual Studio, 2022. [Online]. Available: https://visualstudio.microsoft.com. [Accessed October 9, 2022]
[13] B. Gailer, "Tools and workflow for the development of interactive JSXGraph applets in a Moodle course", In 3. International JSXGraph Conference. University of Bayreuth, Germany, Eds. C. Miller and A. Wassermann, 2022.
[14] Mumie Project. *mumie.net*, 2022. [Online]. Available: https://www.mumie.net. [Accessed October 8, 2022]
[15] M. Kraska, Meclib Github repository, 2022. [Online]. Available: https://github.com/mkraska/meclib. [Accessed October 8, 2022]
[16] M. Kraska, Meclib tryout at jsfiddle with JSXGraph 1.4.4, 2022. [Online]. Available: https://jsfiddle.net/3tkasw0f/. [Accessed October 9, 2022]
[17] K.J. Shryock and J. Haglund. "Instrument for assessing skills related to free body diagrams in a sophomore engineering mechanics course", presented at 2017 ASEE Annual Conference & Exposition, Columbus, Ohio. [Online]. Available: https://peer.asee.org/28542. [Accessed October 8, 2022]

8 Author

Prof. Dr.-Ing. Martin Kraska teaches graduate and undergraduate courses of engineering mechanics, materials sciences and finite elements at Brandenburg University of Applied Sciences, Magdeburger Straße 50, 14770 Brandenburg an der Havel, Germany. He is in charge for the undergraduate program of mechanical engineering. He is supporting application of free mathematical and simulation software such as CalculiX and SMath Studio in education and industrial applications.

Article submitted 2022-09-03. Resubmitted 2022-10-28. Final acceptance 2022-10-01. Final version published as submitted by the authors.