PMark
A tool to support criteria-based marking

Paul Anderson <dcspaul@ed.ac.uk>
School of Informatics
University of Edinburgh

1 Introduction

PMark is a tool for computing assessment marks using a criteria-based marking scheme. Fig 2 shows a typical input file, and fig 7 shows the corresponding output generated by the marking scheme in fig 6. The software is freely available\(^1\) under the GPL license.

This document explains some of the rationale for the tool, and provides a guide to its usage. See Sadler [5] for a good overview of criteria-based assessment in general.

The document contains a lot of reference information. Sections §2-§5 are sufficient for a basic understanding and usage: section §2 provides some clarification of the terminology and explains the overall principles; §3 is an introduction to the application by walking through some basic examples; §4 describes the graph plotting functions; and §5 describes the feedback generation.

The remaining sections include more examples (§6 and §7); a complete reference to the language (§8); details of the algorithms and a guide to debugging (§9); and a full description of the command options (§10).

2 Background

PMark is designed to assist with criteria-based assessment. This involves being explicit about the criteria on which the quality of the work will be judged. The term is often interpreted in different ways [5], but we take a fairly strict interpretation. In particular:

- It is not norm-based: the results for one student do not depend on their past performance, or on the performance of the other students (“grading on the curve”).
- It is not compensatory: students can not succeed by excelling at some parts of the assessment and failing at others (unless this is an explicit intention for some part of the assessment). Schemes which involve summing numerical marks from separate sections or questions are compensatory.

Notice that schemes such as that shown in appendix B have clear criteria, but they do not meet the above requirements of being non-compensatory. The scheme shown in appendix A does meet these requirements.

2.1 Decision rules

PMark uses simple decision rules, such as those shown in appendix A to compute a final grade, based on a number of criteria. Automating this process has a number of advantages:

- The results can be easily and reliably calculated even with large numbers of rules. This provides an opportunity to experiment with larger numbers of small rules which may produce more consistent and reliable results, especially with multiple markers and many students.
- If there is some reason why the results do not appear to provide a true assessment of the objectives, the rules can be easily modified retrospectively. For example, we may decide that students who have just failed on one criterion, should pass overall if they have done exceptionally well on some others.
- The automation has the potential to automatically identify individual weaknesses and report those to the student as feedback.

As an example: one aspect of our assessment of student (computer) code, involves assessing how clear that code is for other (human) readers. Previously, this has been assessed with a single “readability” criteria. Especially with multiple markers, the results for this have been extremely variable, and the feedback to the students has depended on the variable comments from the markers. We now propose using a number of explicit questions, marked on a simple 4-point likert scale (which we think of as: no, not-really, sort-of and yes). For code readability, the questions might include:

- Is the code properly indented?
- Do the large-scope variables have meaningful names?
- Are there adequate comments?
- Are there too many comments?
- Is there redundant commented-out code?
- Are there methods which are too large?
- …

There is, of course, a danger that the overall result of this process may not conform with the marker’s intuitive impression of the readability. In this case, PMark...
gives us the opportunity to investigate why this is so, and potentially to adjust the rules to compensate.

2.2 Interpolation

An assessment based on decision rules such as these requires a separate set of rules to define the requirements for each grade. In many cases, there is a requirement to return a value on a finer-grained scale (such as a percentage). It is quite common for the rules to be mapped onto fixed ranges on the scale (e.g. a pass is 50-60%), and for the marker to decide manually on the intermediate value.

PMark can automatically compute an interpolated value in these cases, and the resulting ranking should correspond well with an intuitive reading of the results. However, the exact values will vary if the mark scheme is modified, and it would be a mistake to make a progression decision for example, based on the absolute values between the grade boundaries.

It is important to note that:

1. The interpolation has no effect on the final grades. These are awarded (or not) purely according to the corresponding rules.

2. The interpolated values are computed by heuristics which attempt to differentiate between students who have the same grade by considering how “well” the criteria for the grade have been met, and how “close” the student is to obtaining the next grade.

The first point represents an important decision. We would like to be able to justify and reason about the final grade. This is very difficult if more complex algorithms [4, 1] are used to make decisions at the grade boundaries.

3 Using PMark

3.1 Computing a grade

The PMark application computes an overall mark for an assessment, based on a number of attributes. An attribute is any property which can be assigned a value from a finite range. For example:

- numerical marks for a short answer question, code fragment or essay.
- an MCQ choice, such as A...E.
- likert-scale values (textual or numeric) for some attribute such as code readability or referencing in an essay.
- pass/fail results from an “auto-marked” program test.
- an attendance count.
- etc...

PMark requires a marking scheme to define the attributes and their expected types. For example, if we have an essay (marked out of 10) and two practical exercises (which have been completed or not), then the marking scheme might include:

```
[types]
out-of-10: [0..10]
is-completed: [no,yes]

[attributes]
essay: out-of-10
ex1: is-completed
ex2: is-completed
```

Notice that the values of the types must be written in ascending order: is-completed is defined as [no,yes] and not [yes,no], because yes is “better” than no. The names of the types and attributes can include letters and numbers, as well as hyphens and underscores, but they must start with a letter.

The marking scheme also needs to specify the various results and their types:

```
[results]
final-grade: pass-fail
```

We also need to define the result type (if it is not one of the types already defined for the attributes). Here, we are using a simple pass/fail:

```
[types]
...
pass-fail: [fail,pass]
```

PMark computes a final grade for the assessment using a criteria-based procedure. The marking scheme specifies a number of rules with associated criteria for each grade, and a particular grade will be awarded (only) if all of the corresponding criteria are met. For example, if a pass requires (only) a mark of 5 for the essay (ignoring of the practical exercises for now), we might have:

```
[rules]
pass: essay=5
```

Normally, there will be no rules for the bottom grade (fail in this case), and that will be awarded if none of the other criteria are met. If the criteria for several

---

2In this case there is only one result, but in general, there may be more.
grades are satisfied, then the highest one is used.

The complete marking scheme for this example is shown in fig 1.

```plaintext
[types]
out-of-10: [0..10]
is-completed: [no,yes]
pass-fail: [fail,pass]

[attributes]
essay: out-of-10
ex1: is-completed
ex2: is-completed

[rules]
pass: essay=5

[results]
final-grade: pass-fail
```

Figure 1: ex1.pmark

### 3.2 Running PMark (example 1)

Evaluating a marking scheme requires a CSV file with one row per student. The id column contains the student identifier, and the remaining columns are the values of the attributes. Fig 2 shows some typical data for the above scheme.

```plaintext
id,ex1,ex2,essay
Benjamin,no,yes,2
Madison,no,no,7
James,yes,yes,4
Emily,yes,no,9
Ryan,yes,no,8
Avery,yes,no,5
```

Figure 2: ex.csv

The data file and mark scheme for this example are included in the distribution, so it can be evaluated from the terminal with this command:

```bash
pmark eval ex1
```

Fig 3 shows the output.

As well as the column for the student identifier, this includes a column for each of the results listed in the marking scheme (only one in this case). Notice that Benjamin and James fail as expected (their essay mark is less than 5), and all of the others pass (their essay mark is at least 5).

New mark schemes can be created with a simple text editor: for example, the markscheme foo.pmark could be run with the data file foo.csv (both in the current directory) using the command:

```bash
pmark eval foo
```

### 3.3 Multiple rules (example 2)

If (in addition to the essay) we would like the students to complete (at least) one of the practical exercises in order to pass overall, then we can simply add the extra rule shown in fig 4.

```plaintext
[types]
out-of-10: [0..10]
is-completed: [no,yes]
pass-fail: [fail,pass]

[attributes]
essay: out-of-10
ex1: is-completed
ex2: is-completed

[rules]
pass: essay=5
pass: ex1=yes or ex2=yes

[results]
final-grade: pass-fail
```

Figure 4: ex2.pmark

Fig 5 shows the corresponding output (using the data from fig 2 again). Madison now fails, despite passing the essay, because she failed both of the practical exercises.

Notice that even this simple example is not possible to
replicate with an additive marking scheme: we do not want students to pass if they have not passed one of the practical exercises, regardless of how well they have performed in the essay (and visa-versa).

### 3.4 Multiple grades (example 3)

We can provide more granularity in the result by specifying an extra grade in the type: example 3 includes an extra distinction grade (fig 6A), and two extra rules (fig 6B).

```
[types]
out-of-10: [0..10]
is-completed: [no,yes]
grade: [fail,pass,distinction]  (A)

[attributes]
essay: out-of-10
ex1: is-completed
ex2: is-completed

[rules]
pass: essay=5
pass: ex1=yes or ex2=yes
distinction: pass
distinction: ex1=yes and ex2=yes  (B)

[results]
final-grade: grade
```

Figure 6: ex3.pmark

The final grade will now be pass, fail, or distinction. The pass and fail grades remain the same, but a distinction requires all of the conditions for a pass, as well as completion of both practical exercises (notice the and in the distinction as opposed to the or in the pass criteria).

```
id,final-grade
Benjamin,fail
Madison,fail
James,fail
Emily,pass
Ryan,distinction
Avery,pass
```

Figure 7: ex3 output

The results are shown in fig 7 (using the data from fig 2 again). Ryan now has the expected distinction.

### 3.5 Interpolation (example 4)

If we want to return a result with a finer granularity, such as a percentage, we could simply translate the fail/pass/distinction into (say) 0/50/70. But this does not capture the finer details: Avery and Emily both pass and would be awarded the same mark (50) by this process, although Emily performed much better than Avery on the essay (and the same on the exercises).

If we specify a finer-grained type for the results, PMark will interpolate the values between the rules (see §2.2). Example 4 (fig 8) uses the Edinburgh University common marking scheme\(^5\) which has a range of 0..100, with aliases for various grades.

```
[types]
out-of-10: [0..10]
is-completed: [no,yes]
cms: [0..100]
  pass = 40
  distinction = 70

[attributes]
essay: out-of-10
ex1: is-completed
ex2: is-completed

[rules]
pass: essay=5
pass: ex1=yes or ex2=yes
distinction: pass
distinction: ex1=yes and ex2=yes

[results]
final-mark: cms
```

Figure 8: ex4.pmark

```
id,final-mark
Benjamin,23
Madison,18
James,37
Emily,59
Ryan,97
Avery,51
```

Figure 9: ex4 output

The results are shown in fig 9 (using the data from fig 2 again). PMark uses heuristics to interpolate values between the grades, so that Emily now has a higher mark than Avery - although they both still have the same grade (pass). Benjamin, Madison and James who all failed are similarly differentiated.

3.6 Weighting (example 5)

Despite the inexactness of the interpolation, it is sometimes useful to have certain criteria treated as more important than others when calculating the interpolated values. For example, we might like the essay to carry more weight. Example 5 (fig 10) shows the weight of the essay increased by 500% relative to the exercises.

```
[types]
out-of-10: [0..10]
is-complete: [no,yes]
cms: [ 0..100
pass = 40
distinction = 70 ]

[attributes]
essay: out-of-10
ex1: is-complete #ex
ex2: is-complete #ex

[rules]
pass: 500% essay=5 and 1 #ex=yes
pass: ex1=yes or ex2=yes
distinction: pass
distinction: ex1=yes and ex2=yes

[results]
final-mark: cms
```

The results are shown in fig 11 (using the data from fig 2 again). Emily’s marks have now increased, and Ryan’s have decreased since we are emphasising the essay mark over the practical exercises. *The grades however have not changed*, since this affects only the interpolated values.

3.7 Shortcuts (example 6)

If a lot of attributes are being used, it can be useful to group them. PMark allows tags to be assigned to attributes (fig 12A) so that they can be referred to as a group (fig 12B).

The results are shown in fig 13 (still using the data from fig 2). This example is functionally equivalent to the previous one.

The `1` and `all` operators can be applied to any number of attributes, either using tags, or by grouping comma-separated conditions in `{braces}`. Section §8.4 describes the available rules.

3.8 A bigger example

Appendix C shows a larger example. This illustrates a few other features and techniques:

(A) Free text comments can be included in the markscheme using the // symbol. The comments above the rules in the example relate the markscheme directly to the rubric and the assignment objectives.

(B) This example uses sub-rules to define some concepts such as good-design. These do not represent a grade, but they can be referenced from other rules. This provides a useful way of defining higher-level concepts in terms of collections.
of criteria. PMark can also plot these rules (see §4) to provide an overview of the class performance on some specific aspect of the assignment.

(C) The attributes in this example include an opportunity for the marker to note any exceptional criteria which have not been explicitly covered elsewhere. The marker would explain these in the written feedback, and the markscheme takes these into account when considering whether to award a mark towards the top of the range.

(D) Type specifications can be prefixed with a fractional specifier (§8.1), such as ++/--, which allows the values to be followed by one or two plus or minus signs. This can be useful to provide a slightly finer granularity for the interpolation without obscuring the basic grade.

(E) The final results can be scaled by a specified factor.

4 Plotting graphs

PMark can generate histograms for any of the attributes, results or rules. This makes it easy to visualise the overall performance of the cohort on any specific aspect of the assessment. The gnuplot package (§10.5) is required to render the plots.

4.1 Plotting results

Histograms of the results can be plotted directly in a default style. The output can be written to a specific directory\textsuperscript{6} (the first command below), or the plot can simply be displayed (the second command):

\begin{verbatim}
  pmark plot lickert -p some-directory
  pmark plot lickert
\end{verbatim}

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{result_histogram.png}
\caption{A basic result histogram}
\end{figure}

The resulting graph is shown in fig 14. Any of the attributes can be plotted in a similar way using the \texttt{-g} option. For example:

\begin{verbatim}
  pmark plot lickert -g a1,a2
\end{verbatim}

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{rule_histogram.png}
\caption{A rule histogram}
\end{figure}

4.2 Plotting rules

The \texttt{-g} option can also be used to plot rules. The output format in this case is slightly different: the histogram shows how “well” the rule was satisfied (fig 15). Values are clearly separated into “pass” (green) and “fail” (orange), but the histogram shows the scores which are used in the interpolation: larger values indicate a “better” performance. Appendix C.1 shows the plots for each grade of a typical assignment.

4.3 Customising the graph

By default, the x-axis of a result or attribute plot is labelled with the values of the corresponding type\textsuperscript{7}. Explicit graph labels can be assigned, and the colours of the bars\textsuperscript{8} can be set by specifying properties on the type values (fig 16).

\begin{verbatim}
  result: [ 0..100
    F=0 ( colour=red, label="0" )
    P=40 ( colour=orange, label="pass" )
    D=70 ( colour=blue label="distinction" )
  ]
\end{verbatim}

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{custom_graph.png}
\caption{Customising the graph properties}
\end{figure}

Various properties of the graph, including the axis labels, the minimum height, and the width of the histogram bars can be specified explicitly by defining one or more graphs in a graph section (fig 17). See §8.5 for a description of the available properties.

Fig 18 shows a typical custom graph.

If a graph section is present, then the specified graphs will be plotted by default, instead of the results. Indi-

---

\textsuperscript{6}A directory specification is required because the output may consist of multiple files.

\textsuperscript{7}If a value has several aliases, then the lowest numerical one is preferred.

\textsuperscript{8}Colours are applied to the specified value and all higher values (until the next explicit colour).
individual graphs, as well as results, attributes and rules can then be plotted using the \texttt{-g} option.

4.4 Image formats & previewing

gnuplot can render the output in many different formats. The available formats and the exact appearance will depend on the installed version. The plot \texttt{-P} option can be used to specify the format. The default is pdf. Commonly supported alternatives include \texttt{png}, \texttt{svg}, \texttt{jpeg}, and \texttt{canvas (html)}. The plot \texttt{-F} option can be used to specify a different font which may be useful if the graphs are to be included at a small size (as in this document). For example: \texttt{-FHelvetica,24}.

The plot \texttt{-N} option produces the gnuplot source file as an output (rather than the image file). This can be manually edited to give complete control over the appearance and style.

PMark plot uses the \texttt{open} command on OSX to preview the graphs. This automatically selects an appropriate preview program for the format. On Linux, \texttt{evince} is used for the default PDF format. Previewers for other formats can be specified with environment variables \texttt{PMARK\_PREVIEW\_format}, or (the default) \texttt{PMARK\_PREVIEW}.

5 Generating feedback

Especially with a large number of attributes and rules, it is not always obvious why a particular grade has not been obtained. PMark can provide some feedback which shows what would have been necessary in order to achieve the next grade. The following command will display the feedback for example 5 (fig 10):

\texttt{pmark eval ex5 -f-}

Fig 19 shows the portion of the output for Madison and James.

This information can be returned directly to the students, or simply used as a guide to interpreting the results and tuning the marking scheme. However, it is usually helpful to provide more meaningful descriptions for the failing criteria. This can be achieved by adding annotations to the attributes and rules in the schemes. It is also possible to customise the format of the output so that it is more suitable for direct return to the students. Appendix D.1 shows an annotated example and the associated output:

5.1 Annotating attributes

If the failing grade requires an attribute to have a specific value, and that requirement has not been met, then PMark will display default feedback which simply shows the attribute name and the actual and required values (as shown in fig 19).

However, there will usually be a specific rubric for key values of the attribute, and this can be attached to the appropriate value as shown in the example (appendix D.1A). These annotations can be attached to any number of attribute values and the the highest annotation between the actual and required values will be the one displayed. Appendix D.2 shows the resulting output. Sometimes it is helpful to see the underlying attribute values as well as the annotation, and the debug option \texttt{-Dfbdetails} enables this.

5.2 Annotating rules

The reason why a particular grade has not been reached can always be described by a logical expression involving a set of attributes and their required values. It is possible to display this full expression using the debug option: \texttt{-Dfeedback}. However, this can be complex and confusing, so PMark normally shows only a simple list of attribute requirements which includes all of the failed requirements that would have contributed to the higher grade.

In some cases, this can generate feedback which is too detailed, in which case annotations can be attached to rules in order to summarise the more detailed feedback from the individual attributes.

For example: the report in fig 19 lists both \texttt{ex1} and \texttt{ex2} as requirements for Madison, even though only
Madison (25) did not meet the requirements for any of the grades. For a pass (40), we would like to have seen:
- a yes for the ex1 attribute instead of a no.
- a yes for the ex2 attribute instead of a no.

James (21) did not meet the requirements for any of the grades. For a pass (40), we would like to have seen:
- a 5 for the essay attribute instead of a 4.

Figure 19: (part of) the ex5 feedback output

one of the two is required for a pass. If the requirement had been for “1 out of 10”, then all 10 attributes would be listed. By adding the rule annotation shown in appendix D.1B, we get the simpler summary feedback shown in the corresponding output (appendix D.2).

By choosing and annotating appropriate sub-rules, a lot of control can also be exercised over the generated feedback. The default behaviour can also be changed so that only one necessary requirement is shown, rather than all possibilities (see §8.4).

If the debug option -Dfbdetails is enabled, the underlying feedback will be shown in addition to the summary.

5.3 Feedback destination

The -f option specifies a directory for the generated feedback files. By default, a separate file is generated for each student, in a sub-directory with the name of the corresponding result. If -f- is specified, all of the feedback is written to the standard output.

5.4 Formatting the feedback

The feedback can be formatted either as plain text, or HTML (see §8.4). However, both of these are generated by a templating process which allows complete control over the format of the output. As well as the feedback and computed results, this has access to all of the fields from the input CSV file.

The example generates plain text output (D.1C) but explicitly overrides one of the default templates (D.1D) to include the comment field from the input CSV file as part of the generated feedback. This can been seen in the output for John and Emily. Appendix E shows the available templates.

6 Further examples

6.1 Multiple choice

The previous examples could easily be used to process MCQ marks, provided that the questions have already been “marked” and the input file contains right/wrong values for each question. However, fig 21 shows how PMark can be used to perform the actual marking of the individual questions. In this case, the input CSV file (fig 20) contains the responses (A, B, C) to the questions, and the attribute types are used to determine which of these are correct for each question.

<table>
<thead>
<tr>
<th>id, Q1, Q2, Q3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Luna, B, C, A</td>
</tr>
<tr>
<td>Leo, B, A, B</td>
</tr>
<tr>
<td>David, A, A, B</td>
</tr>
<tr>
<td>Penelope, A, A, C</td>
</tr>
</tbody>
</table>

Figure 20: mcq input

Fig 22 show the corresponding output.

<table>
<thead>
<tr>
<th>[types]</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: [wrong=B=C, right=A]</td>
</tr>
<tr>
<td>B: [wrong=A=C, right=B]</td>
</tr>
<tr>
<td>C: [wrong=A=B, right=C]</td>
</tr>
<tr>
<td>mark: [F, G, VG, D]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[attributes]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1: A #q</td>
</tr>
<tr>
<td>Q2: A #q</td>
</tr>
<tr>
<td>Q3: C #q</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[rules]</th>
</tr>
</thead>
<tbody>
<tr>
<td>G: some #q=right</td>
</tr>
<tr>
<td>VG: most #q=right</td>
</tr>
<tr>
<td>D: all #q=right</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[results]</th>
</tr>
</thead>
<tbody>
<tr>
<td>total: mark</td>
</tr>
</tbody>
</table>

Figure 21: mcq.pmark

All of the marking information is now retained, and it would be straightforward to modify this to give partial marks for one of the “wrong” answers (for example). This could easily be done retrospectively, in the case
where a problem is only discovered after the assessment.

6.2 Overlapping topics

Assume that we have an assessment which requires a number of skills to be demonstrated. And assume that the separate attributes of the assessment cover different subsets of those skills: so question 1, for example, may cover skills A and B, etc. fig 24 shows how we can ensure that a combination of questions are passed which cover all of the required skills.

6.3 Multiple result formats

Sometimes it is useful to have multiple output formats for the same result. For example, the markscheme in fig 10 generates a percentage result based on the CMS. We might also like to have the result as a simple pass/fail/distinction.

This is straightforward to achieve by adding a new result type which shares some element names with an existing result type (fig 26). The same rules are then used for both results, so no new rules are required. The corresponding output is shown in fig 27.

6.4 Qualified names

If two result types have an element with the same name, then the same rule will be used to compute the result values. As in the previous example, this is usually what is required. Occasionally, different rules may be required for the different types. In this case, it is usually less confusing to simply change the names of the type elements.

However, the rule names can be qualified as shown in fig 28 so that different rules are used for different results.\(^9\)

\(^9\)In this case, the types of the results are the same, so all of the type elements are the same.
Other (ab)uses

PMark’s linear interpolation can be used to implement various other calculations which are not criteria-based. These are not particularly recommended, but they may occasionally be useful:

7.1 Range mapping

Assuming that the source file contains marks with a particular boundary (e.g. pass=40), PMark can remap the marks to move the boundary to some other mark (e.g. pass=50) - see fig 30 and the associated input (fig 29) and output (fig 31).

Additive marking

The linear interpolation can be used to perform additive marking. See the additive.pmark file in the distribution for an example. However, this is not particularly clear and is not recommended - a conventional spreadsheet usually provides a better solution.

Language Reference

Appendix G shows the formal grammar accepted by PMark.

Sections: A markscheme consists of a number of sections:

8.1 [types] defines the types used for the attributes and results.
8.2 [attributes] defines the attributes in the input file and their types.
8.3 [results] defines the results to be generated and their types.
8.4 [rules] defines the rules used to compute the results from the attributes.
8.5 [graphs] defines the properties for any custom graphs.
8.6 [options] defines any global options.

Sections may occur multiple times in any order, but the entities that they define (attributes, rules etc.) must be defined before they can be referenced.

Identifiers: All of the entities are named with identifiers. These:

- Start with a letter or underscore.
- End with a letter, underscore or number.
- Contain only letters, numbers, underscores and hyphens.

Separators: The entity definitions are separated by semi-colons. Other lists are separated by commas. Both of these can usually be omitted at a newline.
8.1 Types

The type section consists of one or more type specifications:

- A type specification starts with a unique identifier, followed by a colon.
- The elements of the type are an ordered list of identifiers or integers enclosed in []. The elements must be unique within the type.
- A range of numeric elements may be specified as: \( N..M \).
- Types have a maximum of 120 elements.
- Elements may be assigned alternative names (aliases): \( alias=alias=...=name \).
- Element names may be followed by a list of properties, enclosed in brackets (see below).
- The type list may be preceded by a fractional specifier (see below).

See fig 32 for an example.

```
[type]
pass-fail: [fail,pass]
lickert4: +/- [1,2,3,4]
percentage: [ 0..100
  pass=50 (colour=green
       label="pass")
]
```

Figure 32: Type specifications

Type properties: The following properties are supported on type elements:

- **colour=**colour
  The (optional) colour to be used for this element (and higher elements) when plotting the type on a graph. Supported colours are green, blue, orange, red, and gray.
  
- **label=**text
  The (optional) label to be used for this element when plotting the type on a graph.

Fractions: Types specifications with a fraction prefix support values and results with a suffix which gives a finer-grained value:

<table>
<thead>
<tr>
<th>Type prefix</th>
<th>Supported suffixes</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>. +</td>
</tr>
<tr>
<td>+/-</td>
<td>-</td>
</tr>
<tr>
<td>++/--</td>
<td>--</td>
</tr>
</tbody>
</table>

If a value has no explicit suffix, then the default is . for the actual value of an attribute, and the minimal suffix for a required value. This interpretation means that a criteria such as \( essay=5 \) will be true for a value of 5 with any suffix (for example 5--).

8.2 Attributes

The attribute section consists of one or more attribute specifications:

- An attribute specification starts with a unique attribute identifier, followed by a colon. This is the name which must appear as the header of the corresponding column in the input CSV file.
- The attribute name is followed by a type identifier. The elements of the corresponding type define the possible values for the attribute in the input file.
- The type may be followed by one or more tags and/or a list of annotations enclosed in brackets (see below).

See fig 33 for an example.

```
[attributes]
controller: lickert
model: lickert #critical
view: pass-fail (fail = "some text here", pass = "some text here") #critical #stuff
```

Figure 33: Attribute specifications

Tags: Tags have the form \#identifier. Assigning the same tag to multiple attributes allows the tag to be used as a shorthand for the group of attributes when writing the rules.

Annotations: The element annotations have the form \( element=\"text\" \). These are used when generating feedback (§5).

8.3 Results

The results section consists of one or more result specifications:

- A result specification starts with a unique result identifier. This is the name which will appear as the header of the corresponding column in the output CSV file.
- The result name is followed by a type identifier. The elements of the corresponding type will appear in the output CSV file in the result column.
- The type name may be prefixed with a qualifier (identifier). This is used in the rule names to disambiguate cases where two results with the same type require different rules (see example §6.4).
- The type may be followed by a list of properties, enclosed in brackets (see below).

See fig 34 for an example.
Result properties: The following properties are supported on result elements:

- **scale=**N\%  
  A percentage to scale numerical elements of the result when writing to the output file. The value is rounded.

Computing a result: To compute the value of a result, PMark looks for rules which correspond to each of the possible values - i.e. the values of the result type. The final value is interpolated between the the highest passing rule, and the next failing rule. For example, if the result type is:

```
[types]
percentage: [ 0..100
  pass=50, distinction=70 ]
```

then PMark will look for the rules `pass` and `distinction`. If the `pass` rule passes, and the `distinction` rule fails, then the value of the result will be interpolated (see below) somewhere between 50 and 70.

Note that the definition of higher refers to the order of the elements in the type. It is possible (although probably not wise) to create types where lower numerical values are considered higher. For example: `[10..20, 5]`.

Interpolation: The interpolated values depend on the scores of the passing and failing grades. Unlike the simple boolean algebra of the grades themselves, the score calculations are more complex. These are intended to give a heuristic measure of how well the requirements for the specific grade have been met. One consequence of this is that the score calculations are not associative, so the following rule specifications will return different intermediate values in some cases\(^{10}\):

- `r1: a=true and b=true and c=true`
- `r2: a=true and ( b=true and c=true )`
- `r3: ( a=true and b=true ) and c=true`

Section \(\S\) 9 gives a detailed explanation of the calculations.

---

8.4 Rules

The rule section consists of one or more rule specifications.

- A rule specification includes a rule identifier (followed by a colon); a criteria which is used to determine if the rule is satisfied or not; and an optional list of properties, enclosed in brackets (see below).

- The rule identifier may be prefixed with a qualifier (identifier). This is used in the result specifications to disambiguate cases where two results with the same type require different rules (see example \(\S\) 6.4).

- If the same rule identifier appears more than once, then all of the associated criteria must be met in order to satisfy the rule.

See fig 35 for an example.

```
[rules]
pass: essay=5
  distinction: 50% pass and most #ex=good
```

Figure 35: Rule specifications

Criteria: The following criteria are supported:

- **attribute=value**  
  The specified attribute has at least the specified value.

- **#tag=value**  
  All attributes with the specified tag have at least the specified value.

- **rule**  
  The named rule is satisfied.

- **{criteria}**  
  The specified criteria is satisfied.

- **criteria and criteria and ...**  
  All of the specified criteria are satisfied.

- **criteria or criteria or ...**  
  At least one of the specified criteria are satisfied.

- **one of {criteria, criteria, ...}**  
  At least one of the specified criteria are satisfied.

- **all of {criteria, criteria, ...}**  
  All of the specified criteria are satisfied.

- **some of {criteria, criteria, ...}**  
  At least one of the specified criteria are satisfied.

- **most of {criteria, criteria, ...}**  
  Most of the specified criteria are satisfied.

- **all but one of {criteria, criteria, ...}**  
  All but one of the specified criteria are satisfied.

- **all but N of {criteria, criteria, ...}**  
  All but N of the specified criteria are satisfied.

---

\(^{10}\)See the example scheme `assoc.pmark`
All but N of the specified criteria are satisfied. The of is optional in all cases and may be omitted.

**Precedence:** Braces (i.e. `{}`) can be used to specify the precedence when multiple criteria are combined - but note that this may affect the resulting score value (see §2.2). The grammar in appendix G shows the default precedence.

**Tag references:** `#tag=value` may be used in place of `{criteria, criteria, ...}` in any of the above to refer to all of the attributes tagged with the specified value. For example: `some #ex=good`

**Weighting:** Most of the terms may be prefixed with a percentage to adjust the relative weighting used in calculating the interpolated values. See example 5 (§3.6). The weighting does not affect the satisfaction of the rule.

**Fractions:** See §8.1 for an explanation of attribute criteria when applied to fractional types.

**Evaluation:** Section §9 discusses the evaluation of the rules in more detail.

**Rule properties:** The following properties are supported on result elements:

- `feedback=option/option/...`
  Feedback options (see below).
- `template/name=string`
  These template strings can be used to provide compete control over the output formatting. These are described in appendix E.

The following feedback options are supported:

- `all|best|worst|none`
  If this is `none`, then no feedback is generated for this result. Otherwise, it determines the type of feedback for rules which have no explicit annotation. If `best` (worst) is specified, then only the single highest (lowest) scoring alternative is listed.

- `text|html`
  The format of the generated feedback.

- `noattrs|attrs`
  Whether or not to display the list of attribute values.

**8.5 Graphs**

This section is optional. Graphs of attributes, rules and results can be plotted using the -g option without defining a graph section: see §4. However, this section can be used to specify explicit properties (such as labels and bar widths) for custom graphs, in which case the graph section consists of one or more graph specifications:

- An graph specification starts with a unique graph identifier, followed by a colon. This name can then be referenced using the -g option.
- The graph name is followed by an identifier for the attribute, rule or result to be plotted.
- This may be followed by a list of properties, enclosed in brackets (see below).

See fig 17 for an example.

**Graph properties:** The following graph properties are supported:

- `xlabel="text"`
  The X axis label.
- `ylabel="text"`
  The Y axis label.
- `barwidth=N`
  The width of the histogram bars. By default this will be 1 unit of the x axis, or 5 (%) for a rule graph.
- `min=N`
  The minimum height of the graph. It can be useful to set this when comparing multiple graphs to ensure that they all have the same (Y) scale.

**8.6 Options**

This section is optional. It can be used to specify global options. Currently, it is only used to set debugging flags (§9.3):

- `debug: flag, flag, ...`
  Enable the specified debug flags.

**9 Algorithms & debugging**

Evaluating a rule (§9.1) yields a score. Score values in the range [-1,0) represent a fail, and values in the range [0,1] represent a pass. The magnitude indicates how “well” the criteria have been met - or how “badly” they have been failed: a score value of -1 represents the “worst possible” fail, and a score value of +1 represents the “best possible” pass. A value of zero is the most marginal pass. The score value is plotted when a histogram of a rule is generated (§4.2).

To evaluate a result (§9.2), all of the rules matching an element of the corresponding type are evaluated.

- The **passing grade** is the highest\footnote{in terms of type index.} rule whose criteria have all been met.
- The **failing grade** is the lowest rule above the passing rule (whose criteria have not all been met).

The bottom (top) of the scale is used if there is no pass-
ing (failing) rule.

The final result is obtained by interpolating between the passing and failing grades based on their scores. This means that:

- A grade is only awarded if all of the corresponding criteria have been met.
- The interpolated result depends both on how well the passing grade has been met, and on how close the next (failing) grade is to being met.

### 9.1 Evaluating rules

Rule expressions may be quite complex, but there are only two fundamental forms:

- An attribute rule requires that an attribute has (at least) a specific value.
  - For example: essay = 5
- A compound rule requires (at least) \( n \) out of \( m \) sub-rules to be satisfied.
  - For example: essay = 5 and ex1 = true
  - Or: most of \( \{ \ldots \} \)

The status (pass/fail) of the resulting score is a straightforward logical evaluation of the boolean expressions. The value of the score is computed as follows:

#### 9.1.1 Attribute rules:

The required and actual values of the attribute are first normalised into the range \([0,1]\):

\[
v' = v/(n - 1 + f)
\]

(1)

Where \( n \) is the number of distinct elements in the type, and \( f \) is the maximum fractional value. For a basic type, \( f = 0 \). For a fractional type with \( n_{frac} \) equally spaced suffixes, \( f = 1 - 1/n_{frac} \).

Given the normalised actual value \((av)\) and the normalised required value \((rv)\), a rule passes iff \( av \geq rv \). The score value is computed by interpolating between the minimum and maximum possible values. These will be different depending on whether the rule has passed or not:

**For a passing rule:** The maximal pass is when \( av = 1 \). The minimal pass is when \( av = rv \). Interpolating between these to obtain the score gives:

\[
v = \begin{cases} 
1 & \text{if } rv = 1 \\
\frac{av-rv}{1-rv} & \text{otherwise} 
\end{cases}
\]

(2)

**For a failing rule:** The maximal fail (minimal score) is when \( av = 0 \). The upper bound on a failing score is the minimal pass \((av = rv)\). Interpolating between these to obtain the score gives:

\[
v = \frac{av}{rv} - 1
\]

(3)

Note that:

1. \( rv > 0 \) since \( rv > av \) and \( av \geq 0 \).
2. \( v \in [-1,0) \) since \( av < rv \).

#### 9.1.2 Compound rules:

Assume that the sub-rules are sorted in weight order:

\[
r = n \ out \ of \ \{ r_1, r_2 \ldots r_m \}
\]

where \( w_i \leq w_j \forall i \leq j \)

(4)

Where \( w_i \) is the weight associated with sub-rule \( r_i \).

The weighted sum of the sub-rules is:

\[
ws = \sum_{i=1}^{i=m} w_i r_i
\]

(5)

The rule passes iff a sufficient number (at least \( n \) out \( m \)) of the sub-rules have passed. The score value is computed by interpolating the weighted sum between the minimum and maximum possible values. These will be different depending on whether the rule has passed or not:

**For a passing rule:** If the rule \( r \) requires \( n \) out of \( m \) sub-rules to be satisfied, then the minimum requirement for a pass is that the \( n \) least-weighted sub-rules have a minimal pass \((0)\) and the remainder have a maximal fail \((-1)\):

\[
w_{s_{min}} = \sum_{i=1}^{i=n} 0 \times w_i + \sum_{i=n+1}^{i=m} -1 \times w_i
\]

(6)

The maximal pass is when all of the sub-rules have a maximal pass \((1)\):

\[
w_{s_{max}} = \sum_{i=1}^{i=m} 1 \times w_i
\]

(7)

The rule value is computed by interpolating the weighted sum between the minimum and maximum values, onto the range \([0,1]\):

\[
vr = \left\{ 
\begin{array}{ll}
0.5 & \text{if } w_{s_{min}} = w_{s_{max}} \\
\frac{ws-w_{s_{min}}}{w_{s_{max}}-w_{s_{min}}} & \text{otherwise} 
\end{array}
\right.
\]

(8)

The first case occurs in practice only if the weights of all the sub-rules are zero.

**For a failing rule:** The maximal fail (minimum score) is when all of the sub-rules have a minimal score value \((-1)\):

\[
w_{s_{min}} = \sum_{i=1}^{i=m} -1 \times w_i
\]

(9)
An upper bound on the maximum score occurs when the \( n - 1 \) most-weighted sub-rules have a maximal pass (1) and the remainder have a minimal pass (0):

\[
ws_{\text{max}} = \sum_{i=m-n+2}^{i=m} w_i + \sum_{i=1}^{i=m-n+1} 0 \times w_i 
\tag{10}
\]

The rule value is computed by interpolating the weighted sum between the minimum and maximum values, onto the range \([-1,0)\):

\[
v_p = \begin{cases} 
-0.5 & \text{if } ws_{\text{min}} = ws_{\text{max}} \\
\frac{w_s - ws_{\text{min}}}{ws_{\text{max}} - ws_{\text{min}}} - 1 & \text{otherwise}
\end{cases}
\tag{11}
\]

Note that: \( v \in [-1, 0) \) since the upper bound requires a pass for all of the sub-rules (and we are considering only failed rules here).

9.2 Evaluating results

There are four cases to consider when evaluating a result:

1. Some rules have passed, and there is a higher rule which has failed.
2. Some rules have passed but there are no higher failing rules.
3. No rules have passed but there are some failing rules.
4. No rules have passed or failed.

The final case (no rules) is not meaningful and should not be permitted by the language syntax. The other cases are evaluated as follows:

9.2.1 Two rules: Let \( i_p \) be the index of the type element corresponding to the passing grade, and \( i_f \) be the index of the next highest (failing) grade. The result index \( i \) is an interpolation between these grades, using a factor \( f \) which is based on the score values \( v_p \) and \( v_f \) for the two corresponding rules:

\[
i = \text{int}(f \times (i_f - i_p) + i_p) 
\tag{12}
\]

Where:

\[
f = \frac{v_p + v_f + 1}{2} 
\tag{13}
\]

Where: \( v_p \in [0, 1] \) and \( v_f \in [-1, 0) \), hence \( f \in [0, 1) \) and \( i \in [i_p, i_f) \).

9.2.2 Passing rule only: The score value of the passing rule is used to interpolate between the index of the rule and the maximum index for the type. This is a slightly awkward case, since we require a result in the (half-closed) interval \([i_p, i_{\text{max}} + 1)\). Values in the range \([i_{\text{max}}, i_{\text{max}} + 1)\) all represent the highest possible index, but if the type is fractional, they may have different fractional suffixes.

The following formula yields a result in the closed interval:

\[
i' = \text{int}(v_p \times (i_{\text{max}} + 1 - i_p) + i_p) 
\tag{14}
\]

Where \( v_p \in [0, 1] \), hence \( i' \in [i_p, i_{\text{max}} + 1] \).

We explicitly avoid the upper bound to ensure that \( \text{int}(i) \leq i_{\text{max}} \):

\[
i = \begin{cases} 
  i' & \text{if } i' < i_{\text{max}} - \epsilon \\
  i' - \epsilon & \text{otherwise}
\end{cases}
\tag{15}
\]

Where \( \epsilon \) is a small value\(^{12} \).

9.2.3 Failing rule only: The score value of the failing rule is used to interpolate between zero and the index of the rule:

\[
i = \text{int}((v_f + 1) \times i_f) 
\tag{16}
\]

Where \( v_f \in [-1, 0) \), hence \( i \in [0, i_f) \).

9.3 Debugging

The \( -D \) option to the \texttt{pmark} command can be used to enable a number of debugging flags. These can also be specified in the \texttt{options} section (§8.6) of the markscheme. The output from these contains a lot of information and can be rather cryptic. However, they can be very useful in tracing the derivation of the results when this is not clear:

\texttt{eval}

Evaluation of the scores for the criteria.

\texttt{parse}

Parsing of the markscheme.

\texttt{results}

Result computation for the various possible grades.

\texttt{files}

Location of the source, markscheme and output files.

\texttt{feedback}

The expression tree for the failed criteria used to generate the feedback.

\texttt{fbscore}

The numerical scores for the individual feedback points.

\texttt{fbdetails}

The default feedback which would be displayed in the absence of any explicit annotations.

\(^{12}10^{-8} \) in the current implementation.
Appendix F shows the output from -Deval and -D results (on different examples). These illustrate the rule (§9.1) and result (§9.2) evaluation algorithms.

### 10 Installation & usage

PMark is written in Perl and has been tested on MacOS (using Perl v5.18.4) and Linux. There is currently no port for Windows.

#### 10.1 Prerequisites

The basic functions require no dependencies other than the standard Perl distribution. However, the gnuplot package (§10.5) must be installed in order to generate graph plots. A suitable PDF previewer (e.g. evince on Linux, or Preview on OSX) is necessary to view the documentation and the plots. Rebuilding the documentation requires Latex.

#### 10.2 Unpacking & installation

Unpacking the distribution creates a directory pmark-version. This documentation uses the name topdir to refer to the full pathname of this directory. All of the tools are accessed via the single pmark command. This can be executed by using the full pathname:

```bash
topdir/Perl/pmark
```

or by including topdir/Perl in the path:

```bash
PATH=$PATH:
topdir/Perl
```

or by symlinking the pmark command to a directory in the path:

```bash
ln -s topdir/Perl/pmark ~/bin
```

Do not copy the pmark file itself - it uses its own location to locate the rest of the package.

The installation can be checked by running the basic unit tests:

```bash
pmark test all
```

This should report “all tests OK”.

#### 10.3 Help & documentation

For a full list of commands:

```bash
pmark help
```

For a full list of options to a command:

```bash
pmark eval help -v
pmark plot help -v
```

#### 10.4 Common options

The full list of options can be obtained using the above help commands. Some common evaluation options are:

- `-c name, name, ...`
  Include specified columns in the output.
- `-i id, id, ...`
  Process only the specified student IDs.
- `-m file.pmark`
  Specify the markscheme file.
- `-o file.csv`
  Specify the output CSV file.
- `-s name, name, ...`
  Sort the output on the specified fields.
- `-u symbol`
  Include output records for students with incomplete attribute data using the specified symbol for the result value.
- `-v`
  Verbose logging.

The plot command supports some further options, including:

- `-P format`
  Generate plots in the specified format.
- `-F font, size`
  Specify font for plot labels.
- `-g item, item, ...`
  Plot histograms for the specified attributes, rules, or results.
- `-p dir`
  Save plot results in the specified directory (rather than displaying to the screen).
- `-N`
  Save gnuplot source files without generating images.

#### 10.5 gnuplot

The gnuplot package must be installed in order to generate plots (this is not necessary for the other functions). PMark has been tested with gnuplot 5.2. This can be installed on OSX using HomeBrew.

Certain versions may need to be installed using additional options to correctly generate PDF. For example:

14[^14]: [https://brew.sh/](https://brew.sh/)
brew install gnuplot --with-cairo
Appendix A  A criteria-based scheme

The project is assessed entirely on the basis of your report. The report is read independently by the project supervisor and a second member of staff (and, if there is a significant difference of opinion between markers or if either of the two markers gives a failing mark, by a moderator). It must be self-contained and include all information relevant to the project since, in general, the readers will be unaware of the work undertaken, the difficulties encountered and the results obtained. They allocate a numerical mark after assessing the project work in terms of the following criteria:

Criteria

Basic criteria:

- Understanding of the problem
- Completion of the project (“Completion” covers achievement of the original objectives, achievement of modified objectives, or providing convincing evidence that the objectives are unachievable)
- Quality of the work
- Quality of the report

Additional criteria:

- Knowledge of the literature
- Critical evaluation of previous work
- Critical evaluation of own work
- Justification of the design decisions
- Solution of any conceptual problems
- Amount of work

Exceptional criteria:

- Evidence of originality
- Outstanding scholarship or engineering, and/or publishable research

Marks

Projects are marked according to the following classifications, which relate to the above criteria:

0-19: Bad Fail  The project is inadequate on all of the basic criteria.
20-29: Clear Fail  The project is inadequate on more than one of the basic criteria, but not all.
30-39: Marginal Fail  The project is inadequate on one of the basic criteria.
40-49: II  The project is adequate on all of the basic criteria.
50-59: II.1  The project is at least fair on all of the basic criteria and is fair and sometimes good or excellent on all of the additional criteria.
60-69: II.1  The project is at least good on all of the basic criteria and is at least fair and sometimes good or excellent on all of the additional criteria.
70-79: Low I  The project is good or excellent on all of the basic and additional criteria; or it almost achieves this by being fair on only one of the additional criteria, and also has elements of the exceptional criteria.
80-89: High I  The project is good or excellent on all of the basic and additional criteria and also has elements of the exceptional criteria.
90-100: Outstanding I  The project is excellent on all of the basic and additional criteria, and has strong elements of the exceptional criteria.
Appendix B  A compensatory scheme

Research report evaluation criteria

<table>
<thead>
<tr>
<th>Title &amp; affiliations (title effectively describes the research report in a concise, attention-grabbing way; author names professionally listed in alphabetical sequence)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract (100-200 words which reflects the final content of the research report as actually submitted – see FAQ in Course Handbook)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Organisation &amp; structure (overall, research report is organised in a sequence that is easy to follow, using appropriate headings and subheadings; an introduction is present which is not just the same as the abstract – see FAQ in Course Handbook; illustrations have figure numbers and these are referred to in the main text; word count is provided)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Overall visual appeal (layout and balance of text &amp; figures; use of illustrations, video, charts, etc.)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Style and consistency (a clear and coherent writing style is used throughout, with evidence of very good proof-reading for sense and consistency; no typos)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Themes and arguments (the main points are clearly presented and are adequately evidenced with sufficient reference to authoritative sources for academic work; good flow)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>References (house style followed accurately; all references in text appear in reference list, and vice versa)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Conclusion (the conclusion sums up the main points of the research report concisely and is justified by the discussion in the research report; links to practice, policy or research are made clear)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

TOTAL SCORE: /40

The research report makes up 70% of your total SRCf mark.
Appendix C  IPPO assignment

[types]

lickert3: ++/-- [0,1,2,3]
lickert4: ++/-- [0,1,2,3,4]
lickert5: ++/-- [0,1,2,3,4,5]

maybe: [NO,MAYBE,YES]

ippo-cms: [ 0..100
    P1 = diploma = 40
    P2 = 45
    G1 = good = 50
    G2 = 55
    VG1 = very-good = 60
    VG2 = 65
    D1 = distinction = 70
    D2 = 75
    E1 = exceptional = 80
    E2 = 85
    O1 = outstanding = 90
    O2 = 95
]

[attributes]

CLASSES2A: lickert4 // Is there a reasonable collection of classes?
MODEL2A: lickert4 #mvc // Is there a reasonable, explicit model?
CONTROLLER2A: lickert4 #mvc // Is there a reasonable, explicit controller?
VIEW2A: lickert4 #mvc // Is the JavaFx abstracted from the view?
JSON2A: lickert4 // Is there an attempt to implement the JSON?
INIT2A: lickert4 // Are the entities handled in an extensible way
DESIGN2A: lickert4 // Overall, is there a plausible design?
CODE2A: lickert4 // Is the code reasonably structured and readable?
EXCEPTIONAL2A: maybe #ex // Exceptional criteria?
EXCEPTIONAL2B: maybe #ex // Exceptional criteria?
EXCEPTIONAL2D: maybe #ex // Exceptional criteria?
EXCEPTIONAL2DES: maybe #ex // Exceptional criteria?
SUB2B: lickert3 // Does the submitted code run?
RUN2B: lickert3 // Does the application run?
BUGS2B: lickert3 // Is the implementation bug-free?
IF2B: lickert3 // Is the interface clear and well-designed?
DEMO2D: lickert5 // Does the demonstration run?
DRAFT2A: lickert5 // Is there a good draft design?
DESCR2A: lickert5 // Is the draft design clearly described?

[results]

a2: ippo-cms (scale=70%)

[rules]

// a reasonable collection of classes
// all of the MVC components must be reasonably explicit
// most of them must be very clear

good-design: all { CLASSES2A=4, all #mvc=3, most #mvc=4 }
either the submitted code, or the demo must allow movement between locations
no major bugs
working-app: some { RUN2B=2, DEMO2D=3 } and BUGS2B=2

some kind of plausible design is the most important criteria for a pass
P1: DESIGN2A=2

(a reasonable collection of classes & comprehensible code
(even if it doesn’t run)
P2: P1 and all { CLASSES2A=2, CODE2A=2 }

student must have submitted a draft design in order to pass
really, we require a basic working application to pass
but if the design is particularly good, we will accept a buggy implementation
we do require *some* sort of running implementation though
G1: P2 and all { DRAFT2A=1,
some { good-design, working-app },
some { RUN2B=1, DEMO2D=2 } }

a reasonable collection of classes, draft design & code readability
and a reasonable collection of classes, draft design & code readability
G2: G1 and all { some { RUN2B=2, DEMO2D=3 }, CLASSES2A=3, DRAFT2A=2, CODE2A=3 }

student must have a running app which allows movement between locations
and a submission which we were able to run without significant work
VG1: G2 and all { some { RUN2B=3, DEMO2D=4 }, SUB2B=2 }

clearer design & code
a good interface
VG2: VG1 and all { DESIGN2A=3, CODE2A=3, IF2B=2 }

the design must be good
student must submit portable code
initialisation must be handled in a good way
some attempt to implement the JSON
D1: VG2 and all { DESIGN2A=4, SUB2B=3, INIT2A=3, JSON2A=2 }

better collection of classes
bug free
reasonable JSON implementation
a good interface
D2: D1 and all { CLASSES2A=4, BUGS2B=3, JSON2A=3, IF2B=3 }

very good design
good JSON implementation
clear description of the draft design
E1: D2 and all { good-design, JSON2A=4, DESCR2A=3 }

some exceptional criteria
E2: E1 and some #ex=YES

most exceptional criteria
O1: E2 and most #ex=YES
<table>
<thead>
<tr>
<th>final-mark: a2 (barwidth=5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>good-design: good-design (barwidth=5)</td>
</tr>
<tr>
<td>p1: P1 (min=10)</td>
</tr>
<tr>
<td>p2: P2 (min=10)</td>
</tr>
<tr>
<td>g1: G1 (min=10)</td>
</tr>
<tr>
<td>g2: G2 (min=10)</td>
</tr>
<tr>
<td>vg1: VG1 (min=10)</td>
</tr>
<tr>
<td>vg2: VG2 (min=10)</td>
</tr>
<tr>
<td>d1: D1 (min=10)</td>
</tr>
<tr>
<td>d2: D2 (min=10)</td>
</tr>
</tbody>
</table>
C.1 Rule plots for the assignment grades
Appendix D  Feedback example

D.1 Example markscheme

[types]
out-of-10: +/- [0..10]
is-completed: [no,yes]
cms: [ 0..100
  pass = 40
  distinction = 70
]

[attributes]
essay: out-of-10 (
  5 = "An adequate essay."
)
ex1: is-completed
ex2: is-completed

[rules]
pass-either-ex: ex1=yes or ex2=yes
  ( feedback = "Completion of one of the of the practical exercises." )
pass-both-ex: ex1=yes and ex2=yes
  ( feedback = "Completion of both of the of the practical exercises." )
pass: essay=5
pass: pass-either-ex
distinction: pass
distinction: pass-both-ex

[results]
text-result: cms (
  feedback = all/text/attrs
  template/endresult = "<comment>

"
)

© Paul Anderson 2020-03-25 15:46:57
D.2 Example feedback

John (11) did not meet the requirements for any of the grades. For a pass (40), we would like to have seen:
- Completion of one of the practical exercises.
  - essay = 4 (out-of-10)
  - ex1 = no (is-completed)
  - ex2 = no (is-completed)
Note: late submission.

Benjamin (24) did not meet the requirements for any of the grades. For a pass (40), we would like to have seen:
- An adequate essay.
Attributes:
  - essay = 2 (out-of-10)
  - ex1 = no (is-completed)
  - ex2 = yes (is-completed)

Madison (18) did not meet the requirements for any of the grades. For a pass (40), we would like to have seen:
- Completion of one of the practical exercises.
Attributes:
  - essay = 7 (out-of-10)
  - ex1 = no (is-completed)
  - ex2 = no (is-completed)

James (38) did not meet the requirements for any of the grades. For a pass (40), we would like to have seen:
- An adequate essay.
Attributes:
  - essay = 4 (out-of-10)
  - ex1 = yes (is-completed)
  - ex2 = yes (is-completed)

Emily (59) achieved a pass (40) for the text-result. For a distinction (70), we would like to have seen:
- Completion of both of the practical exercises.
Attributes:
  - essay = 9 (out-of-10)
  - ex1 = yes (is-completed)
  - ex2 = no (is-completed)
Note: adjustments applied.

Ryan (97) achieved a distinction (70) for the text-result. This is the highest possible grade.
Attributes:
  - essay = 8 (out-of-10)
  - ex1 = yes (is-completed)
  - ex2 = yes (is-completed)

Avery (52) achieved a pass (40) for the text-result. For a distinction (70), we would like to have seen:
- Completion of both of the practical exercises.
Attributes:
  - essay = 5 (out-of-10)
  - ex1 = yes (is-completed)
  - ex2 = no (is-completed)
Appendix E  Feedback templates

E.1 Default templates

The following table shows the default template strings used when generating feedback text. These can be over-ridden to customise the output (see §5.4). Strings of the form `<:name:` are substituted with the value of the corresponding variable (see the following page). For HTML output, `<|name|>` can be used to escape HTML meta-characters.

<table>
<thead>
<tr>
<th>attr: attribute value</th>
</tr>
</thead>
</table>
| text: `<:*attr:*>: = `<:*av:*>: `{:*type:*}`
| html: `<li>` `<:*attr:*>` = `<:*av:*>` `{:*type:*}` '</li>' |

<table>
<thead>
<tr>
<th>attend: end of attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>text: <code>&lt;/ul&gt;</code></td>
</tr>
<tr>
<td>html: <code>&lt;/ul&gt;</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>attrstart: start of attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>text: Attributes: <code>\n</code></td>
</tr>
<tr>
<td>html: Attributes: <code>&lt;br/&gt;</code> <code>&lt;ul&gt;</code> <code>\n</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>details: one feedback point for an attribute requirement when debugging</th>
</tr>
</thead>
</table>
| text: i.e. a `<:rv:*>` for the `<:*attr:*>` attribute instead of a `<:*av:*>`.`
| html: `<br/>` i.e. a `<|rv|>` for the `<|attr|>` attribute instead of a `<|av|>`.` |

<table>
<thead>
<tr>
<th>endlist: end of list of points</th>
</tr>
</thead>
<tbody>
<tr>
<td>text: <code>&lt;/ol&gt;</code> <code>\n</code></td>
</tr>
<tr>
<td>html: <code>&lt;/ol&gt;</code> <code>\n</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>endpoint: end of one feedback point</th>
</tr>
</thead>
<tbody>
<tr>
<td>text: <code>&lt;/li&gt;</code> <code>\n</code></td>
</tr>
<tr>
<td>html: <code>&lt;/li&gt;</code> <code>\n</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>endresult: end of feedback for student</th>
</tr>
</thead>
<tbody>
<tr>
<td>text: <code>\n</code></td>
</tr>
<tr>
<td>html: <code>&lt;/p&gt;</code> <code>\n</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>fail: displayed if no grade passed</th>
</tr>
</thead>
<tbody>
<tr>
<td>text: did not meet the requirements for any of the grades. <code>\n</code></td>
</tr>
<tr>
<td>html: Did not meet the requirements for any of the grades. <code>\n</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>feedback: one feedback point for an explicit rule annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>text: <code>&lt;:*feedback:*&gt;: </code>&lt;:?score:*&gt;`</td>
</tr>
<tr>
<td>html: `&lt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>list: start of list of points</th>
</tr>
</thead>
<tbody>
<tr>
<td>text: For a <code>&lt;:failgrade:*&gt;</code>, we would like to have seen: <code>\n</code></td>
</tr>
<tr>
<td>html: For a `&lt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pass: displayed if pass and fail grades both exist</th>
</tr>
</thead>
<tbody>
<tr>
<td>text: achieved a <code>&lt;:passgrade:*&gt;</code> for the <code>&lt;:*resultname:*&gt;</code> . <code>\n</code></td>
</tr>
<tr>
<td>html: Achieved a `&lt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>point: start of one feedback point</th>
</tr>
</thead>
<tbody>
<tr>
<td>text: <code>\n</code></td>
</tr>
<tr>
<td>html: <code>&lt;li&gt;</code> <code>\n</code></td>
</tr>
<tr>
<td><strong>require</strong>: one feedback point for an attribute requirement</td>
</tr>
<tr>
<td>------------------------------------------------------------</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>result</strong>: start of feedback for student</th>
</tr>
</thead>
</table>
| **text**: `<:id:` (`:result:`)

<table>
<thead>
<tr>
<th><strong>top</strong>: displayed if no grade failed</th>
</tr>
</thead>
</table>
| **text**: This is the highest possible grade

| **html**: a `<|rv|>` for the `<|attr|>` attribute instead of a `<|av|>`<:?score:`>.

| **html**: `<h4>`<|id|>: `<|result|></h4>`<p>`

| **html**: `<br/>This is the highest possible grade.<br/>`
E.2 Template variables

The following variables are available for use in the templates:

*attr
   The attribute name.

?score
   The score for the current criteria (only displayed when debugging).

*score
   The score for the current criteria.

*av
   The actual value of the attribute.

*rv
   The required value of the attribute.

*feedback
   Feedback annotation.

*failgrade
   The lowest failing grade (if any).

*passgrade
   The highest passing grade (if any).

*resultname
   The name of the current result.

*result
   The mark/grade for the current result.

*type
   The attribute type.

1 only available in point, endpoint, details, feedback and require templates.
2 only available in feedback templates.
3 only available in details, require and attr templates.
4 only available in details and require templates.
5 only available in attr templates.

In addition, any of the fields in the input CSV file can be included as <name: > (or < |name| >).
Appendix F  Debug output

F.1 -Deval

```bash
$ pmark eval ex2 -Deval -iEmily

[eval] EXPR: <2>({essay=5},<1>({ex1=yes},{ex2=yes})) = ?
[eval] LIST: require 2/2 = ?
[eval]   EXPR: {essay=5} = ?
[eval]     ACTUAL: 9 (out-of-10) => (9+0)/10 = 0.9
[eval]     REQUIRED: 5 (out-of-10) => (5+0)/10 = 0.5
[eval]     ATTR: essay=5 => 0.9/[0.9..1] => 0.8
[eval]   EXPR: {essay=5} = 0.8
[eval] EXPR: <1>({ex1=yes},{ex2=yes}) = ?
[eval] LIST: require 1/2 = ?
[eval]   EXPR: {ex1=yes} = ?
[eval]     ACTUAL: yes (is-completed) => (1+0)/1 = 1
[eval]     REQUIRED: yes (is-completed) => (1+0)/1 = 1
[eval]     ATTR: ex1=yes => 1/[1..1] => 1
[eval]   EXPR: {ex1=yes} = 1
[eval]   EXPR: {ex2=yes} = ?
[eval]     ACTUAL: no (is-completed) => (0+0)/1 = 0
[eval]     REQUIRED: yes (is-completed) => (1+0)/1 = 1
[eval]     ATTR: ex2=yes => 0/[0..1] => -1
[eval]   EXPR: {ex2=yes} = -1
[eval] LIST: got 1/2 => 0/[-1..2] = 0.3333
[eval] EXPR: <1>({ex1=yes},{ex2=yes}) = 0.3333
[eval] LIST: got 2/2 => 1.333/[0..2] = 0.5667
[eval] EXPR: <2>({essay=5},<1>({ex1=yes},{ex2=yes})) = 0.5667
```

F.2 -Dresults

```bash
$ pmark eval ex3 -Dresults -iEmily

[results] evaluating result: final-grade {Emily}
[results] - #1: pass => 0.5667 (pass)
[results] - #2: distinction => -0.2556 (fail)
[results] = #1 (0.5667) << 0.66 >> #2 (-0.2556) => #1 (pass+0.66) => pass
```

Revision: 2.0  2020-03-25 15:46:57
Appendix G  PMark grammar

MARKSCHEME ::= SECTION *
SECTION ::= OPTIONS | TYPES | ATTRIBUTES | RULES | RESULTS | GRAPHS
NAME ::= See § (identifiers)
STRING ::= " CHAR *
INT ::= ( 0..9 ) +
QUALIFIEDNAME ::= NAME ( . NAME )?
SEP ::= , | END-OF-LINE

OPTIONS ::= [options] OPTDEF *
OPTDEF ::= debug : NAME ( , NAME ) * END-OF-LINE
ATTRIBUTES ::= [attributes] ATTRDEF *
ATTRDEF ::= NAME : TYPENAME ( DESCRS | TAG ) *
DESCRS ::= ( DESCR ( SEP DESCR ) * )
DESCR ::= ATTRVAL = STRING
ATTRVAL ::= ( NAME | INT ) (+|+||--|-. )?
TAG ::= #NAME

RESULTS ::= [results] RESULTDEF *
RESULTDEF ::= NAME : QUALIFIEDNAME PROPERTIES?

TYPES ::= [types] TYPEDEF *
TYPEDEF ::= NAME : TYPESPEC
TYPESPEC ::= +/- ? [ VALUES ]
VALUES ::= TYPEVAL ( SEP TYPEVAL ) *
TYPEVAL ::= INT .. INT
| ATTRVAL (= ATTRVAL ) * PROPERTIES?

RULES ::= [rules] RULEDEF *
RULEDEF ::= RULENAME : CRITERIA PROPERTIES?
RULENAME ::= QUALIFIEDNAME
CRITERIA ::= ORTERM
ORTERM ::= ANDTERM ( or ANDTERM ) *
ANDTERM ::= WEIGHTEDTERM ( and WEIGHTEDTERM ) *
WEIGHTEDTERM ::= WSPEC * TERM
WSPEC ::= WEIGHT | SPREAD
WEIGHT ::= INT %
SPREAD ::= < INT % >
TERM ::= SUBSET
| TAGREF
| RULERREF
| ATTR
| BLOCK

SUBSET ::= SELECTOR of ? SET
SELECTOR ::= all | some | most | INT | one | allbut INT | allbut one
BLOCK ::= ( CRITERIA )
SET ::= ( CRITERIA ( SEP CRITERIA ) * )
| TAGREF
TAGREF ::= #NAME = VALUE
RULERREF ::= QUALIFIEDNAME
ATTR ::= NAME = VALUE
GRAPHS := [graphs] GRAPHDEFDEF *
GRAPHDEF := NAME : NAME PROPERTIES ?

PROPERTIES := ( PROP ( SEP PROP ) * )
PROP := NAME = PROPVAL
PROPVAL := STRING | INT | NAME
Appendix H References


