PatternRefiner Tutorial

Bootstrapping a pattern. PatternRefiner presents examples that have been indicated to be positive (indicated with a green header, e.g. Example 1024) or negative (red header, e.g. Example 1001). Our goal is to find more examples resembling the positive examples as we interactively provide feedback. The details of the API (e.g., its semantics, why is a given usage is misuse, and the consequences of their misuse) **can be ignored**.

Scroll to the bottom of the page and **click** on "Infer Pattern" to find the most general pattern that matches the positive examples. Next, you can **inspect** the most general pattern that matches the positive examples in the left pane, ①, and provide feedback on the features in the right panel, ②. To **provide feedback** to specialize the pattern, **check the suggest checkboxes**. A checkmark means that PatternRefiner should consider including this feature after you have clicked on "Reinfer Pattern".

Candidate features for inclusion in the pattern

R: Representativeness
I: Informativeness

Suggest	Row #	R	I	Feature
☐	0		0.00	`new DigestInputStream(...)`
☐	1		0.00	`new DataInfo(...)`
				`try {`
☐	2		0.00	`    MessageDigest.getInstance(..."MD5"...)`
☐	3		1.00	`    MessageDigest.digest()`
☐	4		1.00	`    MessageDigest.update()`
☐	5		0.00	`    StringBuilder.toString()`
				`} catch (`
☐	6		1.00	`  NoSuchAlgorithmException`
☐	7		0.00	`  IOException`
☐	8		0.00	`  CertificateEncodingException`
				`){`
☐	9		1.00	`    new RuntimeException(...)`
☐	10		0.00	`    AbstractEndpoint.getLogCertificate()`
☐	11		0.00	`    new IllegalStateException(...)`
				`}`

A **B**

Optimal
Redundant
May be non-optimal

[Reinfer pattern]

In the feature feedback pane, the candidate features ((A)) for inclusion in the pattern are organized as a skeleton, grouped by their relationship (e.g., a guard, exception handling) to the API under analysis.

The features are collected from the entire dataset of examples, and do not necessarily appear in the same examples as one another.
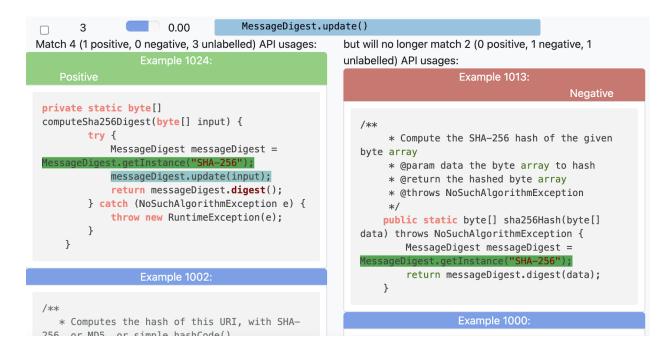
Each feature is displayed with a score indicating their representativeness (R) and informativeness (I):

Representativeness measures how common this feature is in the entire population.

Informativeness measures how well a feature separates the positive examples from negative examples. Higher is better.

There are two additional analyses: 1) **Impact Analysis** and 2)**What-If Analysis**.

Impact analysis. To understand the impact of including a feature, click its name to expand a panel displaying which examples will match, and which examples will no longer match.

What if analysis. When multiple features are selected, some **may** be redundant. Redundant components are colored yellow or red. To inspect the differences between two components, click on the "Why?" link that may appear under the suggest checkbox.

This brings up a new panel which shows a side-by-side comparison of the examples matched by one feature, but not the other.

Warm up task: Given some positive and negative examples (examples 1024, 1001, 1009, 1013, 1017; Example 1024 is positive, while the rest are negative), **go to PatternRefiner** through the provided URL. Once the examples have loaded, **scroll down** to the bottom of the page and **click "Infer pattern"**. Now **wait** as PatternRefiner infers the most general pattern that matches the positive examples.

On the left pane, you can view the pattern inferred by PatternRefiner. Initially, this pattern is "`MessageDigest.getInstance(...`"SHA-256")". Examples matching this pattern instantiates MessageDigest through the `getInstance()` method with **"SHA-256"** as an argument.

This pattern is a starting point for analyzing the matching examples. The right panel now shows possible features for inclusion in the pattern. To answer questions that require you list examples, identify them by their IDs, e.g. 1024.

1. On the right panel, **click** on the suggestion checkbox for `MessageDigest.update()`, and `MessageDigest.digest()`. (typically, they are found between **Rows 4--6**) After PatternRefiner has performed its computations in the background, click on "Why?" under the checkbox in **Row 6** to bring up the What-If Analysis. **Which negative examples** call `MessageDigest.digest()` but do not call `MessageDigest.update()`?

2. **Close** the What-If Analysis from Question 1. **Uncheck** the rows from Step 1.

   Click on the text "`new RuntimeException(...)`" to expand the Impact Analysis panel, which displays the examples matched and not matched by the feature. **Which** positive example **constructs** a `RuntimeException`? (i.e., has "`new RuntimeException(...)`")

3. **Close** the Impact Analysis Panel from Question 2 by clicking on the text "`new RuntimeException(...)`" in **Row 11.**
   You can see the exceptions handled among the examples matching the pattern (usually in **Rows 8-10**, under the "`catch (`"). Judging by the **informativeness** of the exception-handling features, **can at least one pair of** positive and negative examples be distinguished from one another if the pattern included `NoSuchAlgorithmException` (i.e., is its informativeness greater than 0)?

4. **Click** on the text NoSuchAlgorithmException, in **Row 8**. This expands the impact analysis panel, which displays the examples matched and not matched. **Which** unlabelled examples match NoSuchAlgorithmException?

Finally, to **reinfer** the pattern with your feedback on which feature to include,

5. **Close** the Impact Analysis Panel from Question 4.
   Given that we wish to focus on examples that catch the **same exceptions** as the given positive examples, **click** the suggest checkbox of the exception with the **highest informativeness**, and **click** on "Reinfer Pattern". After the pattern has been updated, **click** on "End Task" (on the left panel).

ExampleRefiner Tutorial

Inspecting examples. ExampleRefiner displays every example. Some examples have been labelled as positive or negative. In the following image, Example 1024 is positive, while Example 1001 is negative. Our aim is to discover more examples similar to the positive examples while not similar to the negative examples.

Inspect these examples

Here are the API usage examples in the dataset. Please inspect them to answer the program comprehension questions given to you.

```
Example 1024:
                Positive

private static byte[] computeSha256Digest(byte[] input) {
      try {
          MessageDigest messageDigest = MessageDigest.getInstance("SHA-256");
          messageDigest.update(input);
          return messageDigest.digest();
      } catch (NoSuchAlgorithmException e) {
          throw new RuntimeException(e);
      }
   }
```

```
                        Example 1001:
                                          Negative

public final static String getMessageDigest(byte[] buffer) {
      char[] hexDigits = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c',
'd', 'e', 'f'};
      try {
          MessageDigest mdTemp = MessageDigest.getInstance("MD5");
          mdTemp.update(buffer);
          byte[] md = mdTemp.digest();
          int j = md.length;
          char[] str = new char[j * 2];
          int k = 0;
          for (int i = 0; i < j; i++) {
              byte byte0 = md[i];
              str[k++] = hexDigits[byte0 >>> 4 & 0xf];
```

Click on the "Infer Pattern" button at the bottom of the page.

Infer
pattern

Providing feedback. After you have bootstrapped the pattern, click on "View and label matching examples".

View and label examples

To provide more information that ExampleRefiner can use to construct the pattern, label the other examples to indicate if they are "Positive" or "Negative". You should label examples as "Positive" if they resemble the given *positive* examples in their use of the targeted API, and do not resemble the given *negative* examples.

```java
public static String md5(String input) {
            try {
                    // Create MD5 Hash
                    MessageDigest digest = MessageDigest.getInstance("MD5");
                    digest.update(input.getBytes());
                    byte[] messageDigest = digest.digest();

                    // Create Hex String
                    StringBuilder hexString = new StringBuilder();
                    for (byte b : messageDigest)
                            hexString.append(Integer.toHexString(0xFF & b));
                    return hexString.toString();

            } catch (Exception e) {
                    e.printStackTrace();
            }
            return "";
    }
```

Once you are done providing feedback, click on "Infer Pattern" at the bottom of the page. This shows the inferred pattern. Next, click on "End task".