Пояснения к процедуре валидации датасета

В соответствии с критериями приемки нужно подготовить два набора данных по исходному коду: проверочный набор данных и обучающий набор данных.

В проверочный набор данных должно входить не менее 100 сниппетов кода, среди которых 50% сниппетов должны содержать дефект и 50% сниппетов кода должны не содержать никаких дефектов, где под сниппетом понимается код функции или метода. Корректность разметки на дефектный/недефектный должна быть 95%.

Проверочный набор данных должен содержать сниппеты, позволяющие оценить обучающую способность модели на коде из ранее не встречаемых проектов. Проверочный набор данных должен содержать примеры, позволяющие оценить обучающую способность модели на коде проектов из разных доменных областей.

В обучающий набор данных должно входить не менее 1000 сниппетов кода без ограничений на сбалансированность классов. Единственным требованием является корректная на 95% разметка фрагментов кода.

Планируется, что корректные сниппеты кода без дефектов будут взяты среди сниппетов, содержащих стабильный код. Сниппеты кода с дефектами предлагается отобрать вручную среди сниппетов, которые являются bug-сниппетами пар bugfix, ассоциированных с обработкой инцидентов с меткой “bug”, содержащих полный traceback ошибки. При этом для проверки полученной разметки предлагается сообщать определенную информацию о каждом сниппете с дефектом. Именно, предлагается указывать индекс сниппета в нашей таблице собранных сниппетов с ошибками, url инцидента, полное описание ошибки (full traceback), описание ошибки в коде, описание того, как можно ее устранить, локализация ошибки (файл и название функции/метода), а также тип ошибки. Тип ошибки является важной с исследовательской точки зрения информацией об ошибке, и будет обсуждаться ниже. Предполагается определенное обсуждение структуры предоставляемых наборов с Huawei. К данному руководству также прикладывается файл bugs.xlsx с примерами найденных ошибок и небольшим справочником типов багов.

Описание сниппетов без дефектов

На данный момент предполагается, что корректный сниппет должен содержать название репозитория, исходный код, название файла, где он содержится, название функции, и, быть может, еще количество коммитов до последнего свежего состояния репозитория. Данная информация (за исключением быть может количества коммитов до последней версии репо) уже собрана.

Описание сниппетов с дефектами

Для сниппетов с дефектами предполагается собирать следующую информацию. Во-первых, это индекс bugfix-пары в таблице собранных пар сниппетов по коду с ошибками. Следует иметь в виду, что индексы могут поменяться при перезапуске наших скриптов сбора данных из репозиториев, например, после выявления и устранения ошибок в коде этих скриптов.

Во-вторых, предполагается собирать исходный код сниппетов с ошибками. Здесь тоже кроется определенная проблема. Как будет показано ниже, мы заменим задачу поиска реальной ошибки некоторой синтетической задачей поиска формальной ошибки или дефекта. Таким образом, сниппет кода, содержащий синтетическую ошибку, может отличаться от сниппета кода, который правился разработчиками для исправления реальной ошибки.

Это обстоятельство приходится учитывать при внесении информации по исходному тексту сниппета. Так, например, возникает вопрос: надо ли и каким образом найти сниппет кода нужной версии, поскольку наши данные содержат только фактические исправления и не содержат версий сниппетов кода, которые не менялись.

Сниппет кода, в котором, собственно, и зарождается ошибка, необязательно должен быть расположен в тех функциях, которые сообщены в traceback.

Кроме того, предполагается вносить либо последнюю строку, либо вообще весь traceback ошибки в отдельную графу. Если вносить последнее, то тогда можно данный процесс автоматизировать, поскольку по url инцидента можно найти информацию по traceback ошибки в нашей таблице данных по инцидентам.

Далее, предлагается максимально подробно (в меру понимания) описать ошибку (графа real bug description), сосредотачиваясь больше на причине ошибки, а не на формальном описании, содержащемся в full traceback. Подробность позволит потом легко (в том числе самому человеку) проверить собранные данные.

Анализируя сообщение об ошибке в описании инцидента, описании пулл-реквеста или коммита, собственно, само исправление bugfix (состояние до и после), предлагается сформулировать соответствующее описание так называемой синтетической ошибки (см. ниже), т.е. некоторого аналога реальной ошибки, позволяющего отыскивать похожие реальные ошибки, делая при этом возможно меньше false positives.

Также предлагается описывать как можно исправить найденную синтетическую ошибку (how to fix) как можно подробнее, чтобы читателю было понятно, где, по нашему мнению, здесь синтетическая ошибка. Этот пункт очень важен. Нужно тщательно описать ошибку, чтобы потом было легко проверить чужую работу.

Также предполагается сообщать локализацию синтетической ошибки: название файла и функции (bug file location и bug snippet location).

Наконец, последнее, что нужно внести о сниппете кода с ошибкой, это описание самой синтетической ошибки. В рамках определения типа возникающей синтетической ошибки нужно либо выбрать тип из списка существующих типов (см. вкладку bug types), либо описать новый тип ошибки (synthetic bug pattern).

Синтетические дефекты

Далее мы будем предполагать, что реальные ошибки чаще всего связаны с недопроверенностью кода, а также отсутствием обработки различных крайних значений и исключительных ситуаций. Наш подход состоит в том, чтобы, анализируя хороший код, предложить формальный подход на основе статического анализа кода, который бы позволил описывать ошибки как несоответствие лучшим практикам написания кода.

Рассмотрим серию из нескольких примеров.

ПРИМЕР 1. Ошибка деления на 0

У нас есть две функции:

Первая

def rescale(image, scale, order=None, mode='reflect', cval=0, clip=True,

preserve\_range=False, multichannel=False,

anti\_aliasing=None, anti\_aliasing\_sigma=None):

scale = np.atleast\_1d(scale)

if len(scale) > 1:

if ((not multichannel and len(scale) != image.ndim) or

(multichannel and len(scale) != image.ndim - 1)):

raise ValueError("Supply a single scale, or one value per spatial "

"axis")

if multichannel:

scale = np.concatenate((scale, [1]))

orig\_shape = np.asarray(image.shape)

output\_shape = np.round(scale \* orig\_shape)

if multichannel: # don't scale channel dimension

output\_shape[-1] = orig\_shape[-1]

return resize(image, output\_shape, order=order, mode=mode, cval=cval,

clip=clip, preserve\_range=preserve\_range,

anti\_aliasing=anti\_aliasing,

anti\_aliasing\_sigma=anti\_aliasing\_sigma)

Вторая

def resize(image, output\_shape, order=None, mode='reflect', cval=0, clip=True,

preserve\_range=False, anti\_aliasing=None, anti\_aliasing\_sigma=None):

output\_shape = tuple(output\_shape)

output\_ndim = len(output\_shape)

input\_shape = image.shape

if output\_ndim > image.ndim:

# append dimensions to input\_shape

input\_shape = input\_shape + (1, ) \* (output\_ndim - image.ndim)

image = np.reshape(image, input\_shape)

elif output\_ndim == image.ndim - 1:

# multichannel case: append shape of last axis

output\_shape = output\_shape + (image.shape[-1], )

elif output\_ndim < image.ndim - 1:

raise ValueError("len(output\_shape) cannot be smaller than the image "

"dimensions")

if anti\_aliasing is None:

anti\_aliasing = not image.dtype == bool

if image.dtype == bool and anti\_aliasing:

warn("Input image dtype is bool. Gaussian convolution is not defined "

"with bool data type. Please set anti\_aliasing to False or "

"explicitely cast input image to another data type. Starting "

"from version 0.19 a ValueError will be raised instead of this "

"warning.", FutureWarning, stacklevel=2)

factors = (np.asarray(input\_shape, dtype=float) /

np.asarray(output\_shape, dtype=float))

if anti\_aliasing:

if anti\_aliasing\_sigma is None:

anti\_aliasing\_sigma = np.maximum(0, (factors - 1) / 2)

else:

anti\_aliasing\_sigma = \

np.atleast\_1d(anti\_aliasing\_sigma) \* np.ones\_like(factors)

if np.any(anti\_aliasing\_sigma < 0):

raise ValueError("Anti-aliasing standard deviation must be "

"greater than or equal to zero")

elif np.any((anti\_aliasing\_sigma > 0) & (factors <= 1)):

warn("Anti-aliasing standard deviation greater than zero but "

"not down-sampling along all axes")

# Translate modes used by np.pad to those used by ndi.gaussian\_filter

np\_pad\_to\_ndimage = {

'constant': 'constant',

'edge': 'nearest',

'symmetric': 'reflect',

'reflect': 'mirror',

'wrap': 'wrap'

}

try:

ndi\_mode = np\_pad\_to\_ndimage[mode]

except KeyError:

raise ValueError("Unknown mode, or cannot translate mode. The "

"mode should be one of 'constant', 'edge', "

"'symmetric', 'reflect', or 'wrap'. See the "

"documentation of numpy.pad for more info.")

image = ndi.gaussian\_filter(image, anti\_aliasing\_sigma,

cval=cval, mode=ndi\_mode)

# 2-dimensional interpolation

if len(output\_shape) == 2 or (len(output\_shape) == 3 and

output\_shape[2] == input\_shape[2]):

rows = output\_shape[0]

cols = output\_shape[1]

input\_rows = input\_shape[0]

input\_cols = input\_shape[1]

if rows == 1 and cols == 1:

tform = AffineTransform(translation=(input\_cols / 2.0 - 0.5,

input\_rows / 2.0 - 0.5))

else:

# 3 control points necessary to estimate exact AffineTransform

src\_corners = np.array([[1, 1], [1, rows], [cols, rows]]) - 1

dst\_corners = np.zeros(src\_corners.shape, dtype=np.double)

# take into account that 0th pixel is at position (0.5, 0.5)

dst\_corners[:, 0] = factors[1] \* (src\_corners[:, 0] + 0.5) - 0.5

dst\_corners[:, 1] = factors[0] \* (src\_corners[:, 1] + 0.5) - 0.5

tform = AffineTransform()

tform.estimate(src\_corners, dst\_corners)

# Make sure the transform is exactly metric, to ensure fast warping.

tform.params[2] = (0, 0, 1)

tform.params[0, 1] = 0

tform.params[1, 0] = 0

out = warp(image, tform, output\_shape=output\_shape, order=order,

mode=mode, cval=cval, clip=clip,

preserve\_range=preserve\_range)

else: # n-dimensional interpolation

order = \_validate\_interpolation\_order(image.dtype, order)

coord\_arrays = [factors[i] \* (np.arange(d) + 0.5) - 0.5

for i, d in enumerate(output\_shape)]

coord\_map = np.array(np.meshgrid(\*coord\_arrays,

sparse=False,

indexing='ij'))

image = convert\_to\_float(image, preserve\_range)

ndi\_mode = \_to\_ndimage\_mode(mode)

out = ndi.map\_coordinates(image, coord\_map, order=order,

mode=ndi\_mode, cval=cval)

\_clip\_warp\_output(image, out, order, mode, cval, clip)

return out

Во второй функции resize возникает ошибка деления на 0, если в первой функции параметр scale=0.1, а orig\_shape=(3,10). А именно, после округления в первой функции (показано красным) получаем output\_shape = (0,1). Далее в rescale вызывается функция resize с параметром output\_shape, который в функции resize возникает в знаменателе (показано красным). Подытожим, что мы имеем:

1. выражение для factors в функции resize является отношением двух операндов, один из которых вычисляется в другой функции
2. нет никакой проверки на деление на 0 в этом выражении

Эта ошибка исправляется путем определения немного другой формулы для output\_shape в функции rescale. А именно output\_shape = np.maximum(np.round(scale \* orig\_shape), 1)

Иными словами, исправление ошибки приводит к тому, что output\_shape не содержит нулевых координат, и, как следствие, ошибки деления на 0 не возникает. Таким образом, реальная ошибка в данном примере состоит в том, что параметр output\_shape вычисляется неправильно.

Поскольку правильность выражения для output\_shape проверить довольно трудно, приходится прибегать к тому, чтобы проверять код не на предмет наличия ошибок, а на предмет наличия проверок или обработки разного рода исключительных ситуаций. Мы будем считать синтетической ошибкой отсутствие проверки на наличие нулевых координат в массиве output\_shape в выражении для factors. Таким образом, несмотря на то, что реальная ошибка содержится в функции rescale, соответствующая синтетическая ошибка содержится в функции resize или в перегруженной функции \_\_truediv\_\_ , где нет соответствующей проверки на деление на 0. Сам факт наличия такой проверки говорит о том, что разработчик знает об этой возможной крайней ситуации и умеет ее обрабатывать.

Безусловно, в коде полно таких ситуаций, где знаменатель не равен 0 ни при каких условиях. Наличие подобного рода проверок везде и всюду затруднило бы читабельность кода. В связи с этим, возможно, что стоит считать синтетической ошибкой только те ситуации, где операнд, фигурирующий в знаменателе, вычисляется полностью в другой функции.

Приведем также пример кода, содержащего реальную ошибку, но не содержащего синтетических ошибок.

ПРИМЕР 2. Ошибка IndexError

Если массив, в котором индексируются элементы, и собственно, его индекс не согласованы, возникают ошибки типа IndexError. Имеются две функции:

def \_build\_laplacian(data, spacing, mask, beta, multichannel):

l\_x, l\_y, l\_z = data.shape[:3]

edges = \_make\_graph\_edges\_3d(l\_x, l\_y, l\_z)

weights = \_compute\_weights\_3d(data, spacing, beta=beta, eps=1.e-10,

multichannel=multichannel)

if mask is not None:

# Remove edges of the graph connected to masked nodes, as well

# as corresponding weights of the edges.

mask0 = np.hstack([mask[..., :-1].ravel(), mask[:, :-1].ravel(),

mask[:-1].ravel()])

mask1 = np.hstack([mask[..., 1:].ravel(), mask[:, 1:].ravel(),

mask[1:].ravel()])

ind\_mask = np.logical\_and(mask0, mask1)

edges, weights = edges[:, ind\_mask], weights[ind\_mask]

# Reassign edges labels to 0, 1, ... edges\_number - 1

\_, inv\_idx = np.unique(edges, return\_inverse=True)

edges = inv\_idx.reshape(edges.shape)

# Build the sparse linear system

pixel\_nb = edges.shape[1]

i\_indices = edges.ravel()

j\_indices = edges[::-1].ravel()

data = np.hstack((weights, weights))

lap = sparse.coo\_matrix((data, (i\_indices, j\_indices)),

shape=(pixel\_nb, pixel\_nb))

lap.setdiag(-np.ravel(lap.sum(axis=0)))

return lap.tocsr()

и функция:

def \_build\_linear\_system(data, spacing, labels, nlabels, mask,

beta, multichannel):

if mask is None:

labels = labels.ravel()

else:

labels = labels[mask]

indices = np.arange(labels.size)

seeds\_mask = labels > 0

unlabeled\_indices = indices[~seeds\_mask]

seeds\_indices = indices[seeds\_mask]

lap\_sparse = \_build\_laplacian(data, spacing, mask=mask,

beta=beta, multichannel=multichannel)

rows = lap\_sparse[unlabeled\_indices, :]

lap\_sparse = rows[:, unlabeled\_indices]

B = -rows[:, seeds\_indices]

seeds = labels[seeds\_mask]

seeds\_mask = sparse.csc\_matrix(np.hstack(

[np.atleast\_2d(seeds == lab).T for lab in range(1, nlabels + 1)]))

rhs = B.dot(seeds\_mask)

return lap\_sparse, rhs

Переменная pixel\_nb (помечено красным) определяет размер матрицы лапласиана, выдаваемой на выходе функции \_build\_laplacian. Далее в функции \_build\_linear\_system в строке, помеченной красным мы производим операцию выборки из массива lap\_sparse по индексу unlabeled\_indices, где из-за несоответствия (при определенных условиях) размеров индекса unlabeled\_indices и массива lap\_sparse получается ошибка IndexError, которая затем обрабатывается в \_\_getitem\_\_ массивов scipy.

Таким образом, реальная ошибка состоит в неправильном вычислении значения pixel\_nb в функции \_build\_laplacian, при этом код не содержит никаких синтетических ошибок.

ПРИМЕР 3. Рассмотрим пример ошибки, связанной с обработкой файлов. У нас есть две функции:

def send(self, content=None, \*, wait=False, username=None, avatar\_url=None, tts=False,

file=None, files=None, embed=None, embeds=None):

"""|maybecoro|

Sends a message using the webhook.

If the webhook is constructed with a :class:`RequestsWebhookAdapter` then this is

not a coroutine.

The content must be a type that can convert to a string through ``str(content)``.

To upload a single file, the ``file`` parameter should be used with a

single :class:`File` object.

If the ``embed`` parameter is provided, it must be of type :class:`Embed` and

it must be a rich embed type. You cannot mix the ``embed`` parameter with the

``embeds`` parameter, which must be a :class:`list` of :class:`Embed` objects to send.

Parameters

------------

content

The content of the message to send.

wait: bool

Whether the server should wait before sending a response. This essentially

means that the return type of this function changes from ``None`` to

a :class:`Message` if set to ``True``.

username: str

The username to send with this message. If no username is provided

then the default username for the webhook is used.

avatar\_url: str

The avatar URL to send with this message. If no avatar URL is provided

then the default avatar for the webhook is used.

tts: bool

Indicates if the message should be sent using text-to-speech.

file: :class:`File`

The file to upload. This cannot be mixed with ``files`` parameter.

files: List[:class:`File`]

A list of files to send with the content. This cannot be mixed with the

``file`` parameter.

embed: :class:`Embed`

The rich embed for the content to send. This cannot be mixed with

``embeds`` parameter.

embeds: List[:class:`Embed`]

A list of embeds to send with the content. Maximum of 10. This cannot

be mixed with the ``embed`` parameter.

Raises

--------

HTTPException

Sending the message failed.

NotFound

This webhook was not found.

Forbidden

The authorization token for the webhook is incorrect.

InvalidArgument

You specified both ``embed`` and ``embeds`` or the length of

``embeds`` was invalid.

Returns

---------

Optional[:class:`Message`]

The message that was sent.

"""

payload = {}

if files is not None and file is not None:

raise InvalidArgument('Cannot mix file and files keyword arguments.')

if embeds is not None and embed is not None:

raise InvalidArgument('Cannot mix embed and embeds keyword arguments.')

if embeds is not None:

if len(embeds) > 10:

raise InvalidArgument('embeds has a maximum of 10 elements.')

payload['embeds'] = [e.to\_dict() for e in embeds]

if embed is not None:

payload['embeds'] = [embed.to\_dict()]

if content is not None:

payload['content'] = str(content)

payload['tts'] = tts

if avatar\_url:

payload['avatar\_url'] = avatar\_url

if username:

payload['username'] = username

if file is not None:

try:

to\_pass = (file.filename, file.open\_file(), 'application/octet-stream')

return self.\_adapter.execute\_webhook(wait=wait, file=to\_pass, payload=payload)

finally:

file.close()

elif files is not None:

try:

to\_pass = [(file.filename, file.open\_file(), 'application/octet-stream')

for file in files]

return self.\_adapter.execute\_webhook(wait=wait, files=to\_pass, payload=payload)

finally:

for file in files:

file.close()

else:

return self.\_adapter.execute\_webhook(wait=wait, payload=payload)

и вторая

def execute\_webhook(self, \*, payload, wait=False, file=None, files=None):

if file is not None:

multipart = {

'file': file,

'payload\_json': utils.to\_json(payload)

}

data = None

elif files is not None:

multipart = {

'payload\_json': utils.to\_json(payload)

}

for i, file in enumerate(files, start=1):

multipart['file%i' % i] = file

data = None

else:

data = payload

multipart = None

url = '%s?wait=%d' % (self.\_request\_url, wait)

maybe\_coro = self.request('POST', url, multipart=multipart, payload=data)

return self.handle\_execution\_response(maybe\_coro, wait=wait)

При асинхронной отправке файлов с локального диска с помощью веб-хуков (специальных функций, которые активируют определенные HTTP-запросы по заданному адресу в ответ на какое-то событие на веб-странице, скажем, обрабатывая событие отправки файла)

await webhook.send(file=discord.File("file.txt"))

выдается ошибка о том, что файл закрыт:

ValueError: I/O operation on closed file

В секции try (помечена красным) происходит собственно отправка файла и ожидается ответ от сервера, при этом в разделе finally перед этим происходит закрытие отправляемого файла, что приводит к ошибке.

Таким образом, реальная ошибка состоит в том, что не синхронизированы завершение отправки файла и закрытие этого файла в рамках асинхронного запуска. Синтетическая ошибка состоит в том, что обработка события отправки файла и закрытие этого файла расположены в разных функциях.

(см. <https://github.com/Rapptz/discord.py/commit/311788c7cbac19ee62c6e38a1214ec32259d9d29>)

Подытожим. Мы различаем реальные и синтетические ошибки. Реальные ошибки могут быть самые разные. Мы пытаемся сами придумать для каждой реальной ошибки ее синтетический аналог, который можно обнаружить статически, т.е. не запуская код на выполнение.

Типы синтетических ошибок

Прежде чем пытаться собирать информацию об ошибках, предлагается посмотреть на страницу bug types в файле bugs.xlsx. В ней изложены основные типы синтетических ошибок. Первое, что стоит сделать, это посмотреть эту вкладку и убедиться, что все понятно и обоснованно покритиковать имеющуюся типизацию.

У многих ошибок есть дополнительные метки. Эти метки обозначают тот факт, что операнды выражений и операторов, вызывающих, или более общо, являющихся основной причиной ошибки (появление traceback), вычисляются в той же самой функции, в которой и возникает ошибка ((s)ame function) или в другой функции ((a)nother function). То есть, скажем, если все операнды в выражении, где происходит ошибка, вычисляются внутри функции, содержащей это выражение, то пишем s, иначе пишем a. При этом под термином внешний операнд подразумевается, что нет явного присваивания вида var = внутри этой функции вплоть до того момента, где по вашему мнению содержится ошибка в этой функции, при этом var это либо глобальная переменная, аргумент функции или атрибут объекта self или сам self.

По мере просмотра ошибок могут также возникать какие-то новые типы ошибок. Нужно стараться как можно тщательнее описывать суть синтетической ошибки, чтобы проверка на наличие такой ошибки собирала бы как можно меньше false positives, т.е. примеров корректного кода, удовлетворяющих условиям проверки. Условие с внешними и внутренними операндами является одним из таких условий по моему мнению. Когда в одной функции сводятся воедино несколько операндов, вычисляемых в разных функциях, больше вероятности того, что будет какая-то нестыковка между ними, в особенности, когда функции писались разными людьми. Также, как мне кажется, сложные названия атрибутов подвержены опечаткам, что может привести к ошибкам типа AttributeError.

По моему мнению лучше вначале собрать небольшой черновой вариант датасета с багами, а потом пройтись по нему на второй раз и:

1. проверить новые добавленные категории ошибок на предмет отсутствия пересечений с существующими категориями ошибок
2. простоты и удобства категоризации

При сборе нас не интересуют следующие типы ошибок:

1. ошибки из-за несовместимости (dependency compatibility) с другими библиотеками: например, новая версия используемой проектом библиотеки теперь не поддерживает какой-то атрибут, скажем, он либо переименован, либо удален и т.п. Ошибки в нашем проекте, возникающие в результате этого, нас не интересуют
2. ошибки обратной совместимости библиотеки (backward compatibility)
3. сложные ошибки, которые трудно описать и отыскивать (т.е. требуются специальные знания о библиотеке)

Зачем собирать данные?

1. Чтобы удовлетворить требованиям Хуавей по качеству разметки кода с ошибками
2. Чтобы найти такую задачу поиска дефектов, которую можно решить
3. Чтобы отыскать представление кода, которое поможет нам в решении задачи

Как собирать данные по ошибкам

В прилагаемом файле bugs.xlsx приведено много примеров заполнения данных. Структура таблицы еще неокончательная, может добавиться графа “сниппет после правки” (если сниппет правился).

Каждому из вас переслан набор из 1000 сниппетов, которые правились в рамках обработки инцидентов с меткой ошибка. Первым делом нужно открыть этот набор данных в виде таблицы. В колонке commit\_message иногда содержится краткое описание сделанных исправлений в коде. По этому описанию можно понять подходит ли нам ошибка, исправленная в сниппете кода (см выше про ошибки, которые нам не нужны).

Далее нужно заглянуть на страницу инцидента, к которому относится данное исправление (issue\_url), и попробовать понять суть ошибки из текста описания ошибки и сообщения traceback. Наконец, если суть ошибки не очень ясна, походить по страницам пулл-реквестов (pr\_url) и коммитов (доступны со страницы инцидентов), которые тоже могут содержать информацию по ошибке и тому как она исправлялась. Наконец, нужно посмотреть что исправлялось в сниппете (графы before\_merge и after\_merge). Далее, исходя из этой информации нужно понять в чем состоит реальная ошибка, описать соответствующую синтетическую ошибку, отнести последнюю к одному из типов (см. bugs.xlsx вкладку bug types) или придумать свой тип ошибки.

В таблице нужно заполнить информацию об индексе, issue\_url, о сообщении об ошибке. Эту информацию можно взять со страницы инцидента. Далее нужно дать описание реальной ошибки (real bug description) и описание соответствующей синтаксической ошибки и как ее исправить (how to fix).

Локализацию ошибки для исправлявшегося сниппета можно взять в высланной вам таблице (графа filename), а название сниппета – в графе before\_merge.

Далее нужно отнести синтетическую ошибку к одному из существующих типов (см типы ошибок во вкладке bug types файла bugs.xlsx). При этом надо указать букву a или c в соответствии с тем, являются ли все параметры выражения внутренними или нет. Если тип синтетической ошибки не указан в справочнике, то внести описание новой синтетической ошибки. При этом надо максимально детально описать ошибку и как ее искать с тем, чтобы при такой проверке было как можно меньше ошибок (false positives), т.е. примеров коректного кода, которые удовлетворяют условию проверки.