Lint for Apertium

General

- Apertium lint is a python module that lints out irregular yet acceptable constructs that may creep into files involved in the transfer process. The lint is designed specifically to handle 4 file types dictionaries, transfer, modes and tagger.
- The lint has minimal external dependency and only uses a single external library lxml for it's functioning.
- The lint support only python versions > 3.4
- The the main lint file apertium-lint and it's other associated files are places in the apertium-tools repository in the trunk
- Every language pair has a config file associated with it which governs the issues that are linted out.

Installation

To install apertium-lint :

    Pip install apertium-lint

Usage

To launch apertium-lint:

    Python3 apertium-lint <<filename>>

Features

- Notify developers of irregular yet acceptable constructs that may creep into files involved in the transfer process.
- Specify line number for warnings wherever possible.
- Be awesome.

Contribute

- Issues : http://wiki.apertium.org/wiki/lint
- Code : https://gitlab.com/jpsinghgoud/apertium-lint

Support

Lint is a project solely for the developers, so if you think that there's something missing in the lint or something that you could improve in it, feel free to ping us on the irc or drop us an email on the mailing list.

License

We're yet to get one for the lint.

Config File

Every language pair has a config file that is associated with it. This config file is responsible for guiding the lint while it parses any file. The config file, is essentially a JSON file.

```
{
    "filetype1": {
        "issueName":{
            "message": "Warning message in the form of a string when the given error is detected.",
            "enable" : true
        },
        "issueName":{
            "message": "Warning message in the form of a string when the given error is detected.",
            "enable" : true
        }
    },

    "filetype2": {
        "issueName":{
            "message": "Warning message in the form of a string when the given error is detected.",
            "enable" : true
        }
    }
}
```

The config file is rather interesting. As it's visible above, the config files containing issues associated with different file types. Each issue in turn has an error message and an enable option associated with it. Further description regarding the parts of the config file :

- **fileType** : The lint is designed to handle 4 files types in specific as of now, tagger, modes, transfer and dictionaries (monodixes and bidixes individually).

- **issueName** : The name of the issue serves multiple purposes. First of it's a minimalistic idea about what the issue may be about. Secondly, the issue name corresponds to the name of the method/function present in the lint code that is responsible for detecting this issue. It is extremely vital that the name of the issue is ***not changed*** in the config file. If one needs to change the name of an issue, one also needs to make sure that the name for the corresponding function is also altered.

- **message** : Each issue has a warning message associated with it. This is printed out on the console when the issue is encountered. This helps in making the code more modular. One does not need to change anything in the core code if one wishes to alter the error message associated with any issue.

- **enable** : As the name suggests, **enable** holds a boolean value and assists the lint in selecting or discarding individual issues.

Monodix Lint

The monodix lint, currently detects 12 issues as listed out on the wiki. Here is a list of issues and their detailed descriptions :

1. redundantPardef : This issue is responsible for detecting the reporting issues such as:

   ```
   <pardef n="di/e__vblex"><e><par n="liv/e__vblex"/></e></pardef>
   ```

   Such an entry is considered to be redundant because there is no reason for the paradigm "`di/e__vblex`" to be present unless it is adding any extra rules to generate new lexical units using the root word. It is just linked to "`liv/e__vblex`" and hence can be done away with. Every entry for the prior paradigm can be replaced with the later.

2. rTagData : This issue is responsible for detecting any inconsistency in the data present in the <r> tag in the pardef entries.

   ```
   <e><p><l>as</l><r>o<s n="prn"/><s n="tn"/><s n="f"/><s n="pl"/></r></p></e>
   <e><p><l>a</l><r>abz<s n="prn"/><s n="tn"/><s n="f"/><s n="sg"/></r></p></e>
   ```

The snippet shown above is that of an irregular construct. The text in the <r> tag is chopped off the from the root at the time of definition and is replaced with the text present in the <l> tag. It is important to maintain consistency in the data present in the <r> tag and that is exactly what this issue does.

3. paradigmNames : This issue is responsible for enforcing certain rules associated with naming paradigms.

```
<pardef n="outr/o__prn">
```

Above mentioned is a correct paradigm name entry. Currently the core, accepts almost all possible names for paradigms. There are certain subtle rules that are to be followed like having two “_” before the part of speech, the part that is to be trimmed out is present after the “/” and before the __, etc. This issue handles such names and makes sure that paradigms are named in the proper manner. There is a lot of scope for improvement in this particular issue.

4. repeatedAttributesPardef : This issue is responsible for checking any repeated entries in the attributes associated with an entry in the pardef section.

```
<e><p><l>y</l><r>y<s n="adj"/><s n="adj"/></r></p></e>
```

As seen above, the attribute “adj” is repeated twice. This issue detects such entries and warns the developer/user about their presence.

5. repeatedEntriesPardef : This issue is responsible for detecting repeated entries in the paradef entries for a given paradigm.

```
<pardef n="br/other__n">
        <e>       <p><l>other</l>     <r>other<s n="n"/><s n="sg"/></r></p></e>
        <e>       <p><l>other</l>     <r>other<s n="sg"/><s n="n"/></r></p></e>
        <e r="LR"><p><l>ethren</l>    <r>other<s n="n"/><s n="pl"/></r></p></e>
        <e>       <p><l>others</l>    <r>other<s n="n"/><s n="pl"/></r></p><par
n="gen__apos"/></e>
</pardef>
```

As seen above the two entries are essentially the same. Although the attributes in the entry are mentioned in a different order, they are still the same. This issue detects such repeated entries for all paradigms and warns the developers.

6. repeatedEntriesMainSection : Same as repeatedEntriesPardef.

7. repeatedTagEntries : Same as repeatedAttributesPardef, but include repeated entries in the <s> and <par> tags in the Main Section.

```
<e><p><l>y</l><r>y<s n="adj"/><s n="adj"/></r></p></e>
```

8. transferDirection : This is issue is responsible for making sure that a valid transfer direction is supplied (if one is).

```
<e r="XYZ" lm="arithmetic">
       <p><l>arithmetic</l><r>arithmetical</r></p>
<par n="expensive__adj"/></e>
```

   Sometimes transfer direction is mentioned along with entries in the <e> tag. The transfer direction can only be one of the three valid options. Currently, there is no check if the transfer direction that is specified is a valid one or not. This issue checks for the presence of an invalid transfer direction both in the main section & pardef and warns the developer about it. List of valid transfer direction :
   a. Left -> Right (LR)
   b. Right -> Left (RL)
   c. None

9. unusedParadigms : As the name suggests, this issue is used for detecting unused paradigms in the pardefs section. There are various paradigms which are defined but are never used in the entire dictionary, this issue is responsible for handling such entries.

10. blankSpaceDetection : This issue detects “extra” blank spaces that might be present in the entries.

```
<e lm="Middle Ages"><i>Middle Ages</i><par n="house__n"/><par n="gen__apos"/></e>
```

    The main section consists of certain multi-work entries. In such entries it is vital that the blank space be denoted using <b/>. Currently, if an entry is present where there is just white space instead of <b/> it is still accepted and translated. This issue works to enforce the usage of <b/> to denote white spaces.

11. unwantedTag : This issue detects extra/unwanted <b/> tags that might creep into entries.

```
<l>as<b/>minhas<b/></l><r>o<b/>meu<s n="prn"/>
```

    The second <b/> tag present in the above entry is not required and the developer can be

suggested to remove it. Entries where there are two <b/> tags present back to back, such as “<b/><b/>” can also be diagnosed and pointed out.

12. partiallyInLemma : Now, this issue is rather an interesting one. One common error when you have a pardef that defines part of the lemma, is to write that part twice (once in the pardef, once in the entry using the pardef), e.g.

```
<pardef n="enk/e__n">
 <e><p><l>a</l> <r>e<s n="n"/><s n="f"/><s n="sg"/><s n="def"/></r></p></e>
 <e><p><l>e</l> <r>e<s n="n"/><s n="f"/><s n="sg"/><s n="ind"/></r></p></e>
 ...
</pardef>
...
<e lm="slette"><i>slette</i><par n="enk/e__n"></e>
```

Here the correct entry should be

```
<e lm="slette"><i>slett</i><par n="enk/e__n"></e>
```

This issue is responsible for detecting such constructs and warning the users about their presence.

Bidix Lint

The bidix lint, current detects 5 issues as listed out on the wiki. Here is a list of the issues and their detailed descriptions:

1. compareSdefs : This issue is responsible for detecting and reporting the sdefs that may be present in the bidix but are absent in the monodixes (both transfer languages).

2. transferDirection : This issue is responsible for detecting and reporting incorrect transfer directions.

   ```
   <e r="XY"><p><l>ĥina<s n="adj"/></l><r>Chinese<s n="adj"/></r></p></e>
   ```

   Sometimes transfer direction is mentioned along with entries in the <e> tag. The transfer direction can only be one of the three valid options. Currently, there is no check if the transfer direction that is specified is a valid one or not. This issue checks for the presence of an invalid transfer direction both in the main section & pardef and warns the developer about it. List of valid transfer direction :
   a. Left -> Right (LR)
   b. Right -> Left (RL)
   c. None

3. verifyInvariablePart : This issue is responsible for detecting homogeneity in definition of the invariable part across the monodix and the bidix.

   ```
   <e>
     <p>
       <l>eltiriĝi<s n="vblex"/></l><r>back<g><b/>out</g><s n="vblex"/></r>
     </p>
   </e>
   <e lm="back out">
     <i>back</i><par n="accept__vblex"/><p><l><b/>out</l><r><g><b/>out</g></r></p>
   </e>
   ```

   Multiwords with inner inflection consist of a word that can inflect an invariable element. For these entries we need to specify the inflection paradigm just after the word that inflects. The invariable part must be marked with the element <g> in the right side. If the <g> tag is present in the monolingual dictionary, it should also be present in the bilingual dictionary.

4. repeatedEntries : Checks and reports repeated entries in the bidix.

```
<e r="LR" v="val"><p><l>guapo<s n="adj"/></l>   <r>bonic<s n="adj"/></r></p></e>
<e r="LR" v="val"><p><l>guapo<s n="adj"/></l>   <r>bonic<s n="adj"/></r></p></e>
```

   Such entries are detected by the lint, irrespective of order in which the <s n=”xyz”> attributes are present. The lint also considers the “v” attribute while detecting such repeated entries.

5. unwantedWhiteSpace : This issue detects and reports white spaces that may creep into the entries.

   The main section consists of certain multi-work entries. In such entries it is vital that the blank space be denoted using <b/>. Currently, if an entry is present where there is just white space instead of <b/> it is still accepted and translated. This issue works to enforce the usage of <b/> to denote white spaces.

Transfer Lint

1. checkValidPartClip : In transfer files, one common error is calling, for instance in <clip> an attribute in part="" that does not exist. Part, indicates which part of the lexical form is referred to in the 'clip'. The lint detects and reports such issues

   ```
   <code><clip pos="1" side="tl" part="xyz"/></code>
   ```

   Problem : Check that the attribute listed in part="xyz" is defined using a <def-attr>. The valid parts are defined in def-attrs section of the transfer file. The value of the part attribute is cross-referenced against these values to detect any invalid entries. Other than the entries in def-attrs, certain other parts like:
   a. whole: the whole lexical form (lemma and grammatical symbols). Used only when sending out the lexical unit (inside an <out> element).
   b. lem: the lemma of the lexical unit.
   c. lemh: the head of the lemma of a multiword with inner inflection.
   d. lemq: the queue of a lemma of a multiword with inner inflection.

2. unusedDefCats : There are certain redundant def-cats that maybe present in the transfer rules definition but are never actually used. The lint can now detect and report such def cats.

3. repeatedEntriesCatItem : Simple check which iterates over all the <cat-item>s in a <def-cat> and detects repeated <cat-items>

   ```
   <def-cat n="adj">
         <cat-item tags="adj"/>
         <cat-item tags="adj.comp"/>
         <cat-item tags="adj.comp"/>
   </def-cat>
   ```

4. conflictingCatItem : This detects and reports if the same cat-item has been used in two or more def-cats.

   ```
   <def-cat n="nounx">
         <cat-item tags="np.*"/>
   </def-cat>

   <def-cat n="nouny">
         <cat-item tags="np.*"/>
   </def-cat>
   ```

5. repeatedEntriesAttrItem : Again another simple issue that checks for repeated attr-items in def-attr.

```
<def-attr n="temps">
      <attr-item tags="cni"/>
      <attr-item tags="fti"/>
      <attr-item tags="ifi"/>
      <attr-item tags="fti"/>
</def-attr>
```

6. checkValidPosition : Every mention of pos="xyz" is checked to make sure that xyz is less than or equal to the number of elements in pattern-item.

```
<pattern>
       <pattern-item n="on"/>
       <pattern-item n="num"/>
</pattern>
.....
<with param pos = "3"/>
```

When calling the macroinstructions in a rule, it must be specified which is the main lexical unit (the one which most heavily determines the gender or number of the other lexical units) and which other lexical units of the pattern have to be included in the agreement operations, in order of importance. This is done with the <with-param pos=""/> element. Other than with-param, various other tags like <clip pos=""/> depend on the position of the element. This function makes sure the numbers used align with those in the pattern definition.

7. enforceBreakTag : It is important that there exists a <b/> tag between two consecutive <lu> tags. This function reports the if the <b/> tags tags are absent in any such positions.

```
<lu>
       <clip pos="4" side="tl" part="lemh"/>
</lu>
<lu>
        <lit v="se"/>
        <lit-tag v="prn.enc.ref.p3.mf.sp"/>
        <clip pos="4" side="tl" part="lemq"/>
</lu>
```

8. enforceSide : In <out> the side attribute can either take the value of "sl" or "tl". With the lint in place, the user can enforce the side attribute to be "tl" in all the <out> tags.

```
<out>
        <lu>
              <clip pos="1" side="sl" part="lem"
        </lu>
</out>
```

9. checkValidEqualTag : Another common error in transfer files is trying to compare an attribute to a value it cannot take.

```
<equal><clip pos="1" side="tl" part="a_nom"/><lit-tag v="n.acr"/></equal>
```

   Problem : In the above code, the definition of attribute a_nom matches only "n" and "np" in attr-item definitions.

10. **XSD Validation** : XSD Validation for transfer files, takes care of issues such as :
    a. Repeated def-cats
    b. Repeated def-attrs
    c. Repeted def-list
    d. Valid side : ‘sl’ or ‘tl’
    e. Repeated def-macro

Modes lint

1. installBool : In modes files, the install attribute can only take one of two binary values, ‘yes’ or ‘no’.

```
<mode name="fr-eo" install="no">
```

   This function is responsible for making sure that no other value is used for the same.

2. repeatedProgram : Every modes file consits of various programs and there is always a chance that a certain program may unintentionally get repeated.

```
<program name="apertium-transfer">
        <file name="apertium-eo-fr.fr-eo.t1x"/>
        <file name="fr-eo.t1x.bin"/>
        <file name="fr-eo.autobil.bin"/>
</program>
<program name="apertium-transfer">
```

```
        <file name="apertium-eo-fr.fr-eo.t1x"/>
        <file name="fr-eo.t1x.bin"/>
        <file name="fr-eo.autobil.bin"/>
    </program>
```

3. validateProgram : Every modes files consists of various programs, each having a unique name. It is possible that a program may be wrongly named and passes unnoticed.

    ```
    <program name="apertium-ppretransfer"/>
    ```

    This function validates every program mentioned in the modes file. For the process of validation, it requests for the man page of the concerned program and wait for a response from the prompt. If successful, it moves on. But if at any moment, a certain program is not found, it prompts the user regarding the same.

4. enforceRules : This function is responsible for enforcing certain rules specific to given programs. This is not an exhaustive function and new rules relating to programs (flags and attributes) will be added here.

5. installSwitchNo : If in the definition of a certain mode, the attribute install=”no”, the name should have an appropriate suffix like -morph, -interchunk, etc.

    Detailed Modes :
    a. -anmor or -morph run the morphological analysers
    b. -disam runs up until morphological (CG) disambiguation
    c. -syntax runs up until syntactical (CG) disambiguation
    d. -tagger runs up until probabilistic (apertium-tagger) disambiguation (or, if no .prob, up until the last disambiguation step)
    e. -biltrans runs up until the bidix
    f. -lex runs up until lexical selection
    g. -transfer runs up until (1-stage) transfer
    h. -chunker runs up until the first stage of 3-or-more-stage transfer
    i. -interchunk runs up until the second stage of 3-stage transfer
    j. -interchunk1 and -interchunk2 are used when the pair has 4-stage transfer
    k. -postchunk runs up until the last stage of transfer
    l. -dgen run up until generation (using lt-proc -d to include debug symbols)

6. locateFile : Assuming you're working with modes.xml present in the same directory as the other files for the given language pair.

```
<program name="apertium-interchunk">
       <file name="apertium-eo-fr.fr-eo.antaux1_t2x"/>
       <file name="xyz.bin">
</program>
```

This function checks and prompts incase a file defined in a “program” is missing in the PWD.

7. emptyProgram : Not so much a risk, but this function prompts if a given program does not have any file associated with it.

Tagger lint

1. defLabelClosed : In modes files, the closed attribute in <def-label> can only take one of two values : [true, false]. This function lints and reports if any other value has been used instead of the two valid ones.

```
<def-label name="THATCNJ" closed="true">
```

2. validateLabelSequence : In the <forbid> part of the mode file, there are label sequences with unique label combinations. Each label used here has to specified in the <tagset> part defined before it. This function lints and detects and labels in <forbid> that are absent in <tagset>

Contributing to the lint : Overview

Contributing to the lint is pretty simple and easy. Follow this 3 step process to introduce new features to the lint.

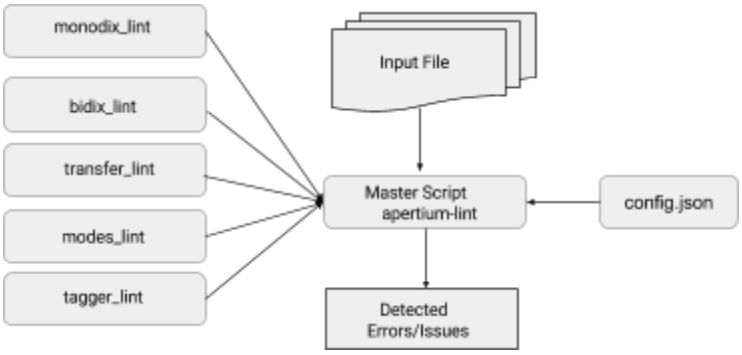
1. Create an entry in the config.json with the required fields.

2. Create a function to tackle this issue in the corresponding lint module (dictionary, transfer, etc) depending on the nature of the issue.

3. Enable the issue in the config.json and the lint will automatically pick it up and execute it.

Contributing to the lint : Detailed

If you're reading this, I assume you wish to contribute to the lint. Awesome! Before you get started contributing,let me explain to you, how the lint works.



The lint consists of 6 files, 5 of which are each associated with a certain filetype and 1 (apertium-lint.py) is the main file which links bundles everything up. To add any "features" to the lint, you'll essentially be modifying one of the 5 files associated with the specific filetypes.

Let's call these 5 files, our mods. Each mod has at least **four** similar functions.

- *main* : responsible for initiating the mod's functions
- *readConfig* : responsible for reading the config file and making all the issues accessible as a dictionary (errorsConf) within the mod

- *XYZErrors* : responsible for iterating over all the errors present in the config file and invoking the functions associated with them
- *parseABC* : responsible for parsing the input file into a manageable format (usually a dictionary) so that the data can be toyed around with.

Every parseXYZ function creates a dictionary with a certain structure so as to encompass all the information present in the input file. The structure of the dictionary for every file type can be found in fileType.txt (monodix.txt, bidix.txt, etc) present in the lint repository. It's very crucial to know how the parseXYZ function parses a certain file to make the contribution process much easier.

Moving on, each "feature" in the mod has a certain function associated with it. For example, if you wish to introduce a new feature in the transfer lint, you'll have to add a function in the transfer mod (transfer_lint.py).

Here's a brief idea of how you could go about adding a new feature to the transfer lint (or any other mod) :

1. Browse through trasnfer.txt to understand how the parseRules function creates a dictionary from the transfer file.
2. Add an entry in config.json. Every function in every mod has an entry associated with it in the config.json. (Refer to Config File to understand how the config file works). Add an appropriate error message and enable the function.
3. **Important :** The name of the function added in the mod should be the same as your entry in the config file.
4. The function in the mod does all the manipulation and lints out the issues
5. Now, to display what we've linted out, use the error message associated with the function as documented in the config file using errorsConf.
6. Comment, comment and comment! It's very important that you tell what you function does, so feel free to write descriptions as long as essays to make it crystal clear what this new awesome feature lints out
7. Publish it on the wiki! All the issues linted out are mentioned on the wiki page. It's really important to update the wiki page as it's the primal POC for anyone who wishes to use the lint.
8. Clean and push to SVN.