An Open-Source Library and Tool for AMR Graphs
Michael Wayne Goodman  Nanyang Technological University
Introduction

If you work with Abstract Meaning Representation, consider the Penman package for both Python and command-line usage:

• Reads and writes AMR graphs
• Inspects, constructs, and manipulates trees and graphs
• Reformats for consistency
• Restructures and normalizes graphs
• Validates graphs with a semantic model
Introduction

Furthermore, it is:

- Well-tested
- Well-documented
- Under a permissive open-source license (MIT)
Abstract Meaning Representation

(1) I swam in the pool today.

(s / swim-01
   :ARG0 (i / i)
   :location (p / pool)
   :time (t / today))
Decoding and Encoding Graphs

(a / alpha
 :ARG0 (b / beta)
 :ARG0-of (g / gamma
   :ARG1 b))

Penman | Michael Wayne Goodman
Page 4/18
Using the penman Command

Demo
Using the penman Command

Start with an example file:

```sh
$ cat ex.txt
(w / want-01 :polarity - :ARG0 (t / they) :ARG1 (g / go-02 :ARG0 t))
```

You can pipe the contents to penman to reformat:

```sh
$ cat ex.txt | penman
(w / want-01
   :polarity -
   :ARG0 (t / they)
   :ARG1 (g / go-02
       :ARG0 t))
```
Simple Reformatting (Command)

You can also give it a file argument and formatting options:

$ penman ex.txt --indent 3 --compact

(w / want-01 :polarity -
 :ARG0 (t / they)
 :ARG1 (g / go-02
 :ARG0 t))

Or view the graph as a triple conjunction:

$ penman ex.txt --triples
instance(w, want-01) ^
polarity(w, -) ^
ARG0(w, t) ^
instance(t, they) ^
ARG1(w, g) ^
instance(g, go-02) ^
ARG0(g, t)
Tree Operations (Command)

Rearrange the branches of the tree structure:

$ \text{penman} \ ex.txt --\text{rearrange}=\text{random} \\
w / \ \text{want-01} \\
\quad \text{:ARG0} (t / \ \text{they}) \\
\quad \text{:ARG1} (g / \ \text{go-02} \\
\quad \quad \text{:ARG0} t) \\
\quad \text{:polarity} -$\) 

Relabel the nodes:

$ \text{penman} \ ex.txt --\text{make-variables}=\text{’a\{i\}’} \\
a0 / \ \text{want-01} \\
\quad \text{:polarity} - \\
\quad \text{:ARG0} (a1 / \ \text{they}) \\
\quad \text{:ARG1} (a2 / \ \text{go-02} \\
\quad \quad \text{:ARG0} a1)$
Using Models (Command)

Check for model validity (–amr uses the AMR model):

$ penman ex.txt --amr --check
(w / want-o1
   :polarity -
   :ARG0 (t / they)
   :ARG1 (g / go-o2
       :ARG0 t))

$ sed 's/:polarity/:polar/' ex.txt | penman --amr --check
# ::error-1 (w :polar -) invalid role
(w / want-o1
   :polar -
   :ARG0 (t / they)
   :ARG1 (g / go-o2
       :ARG0 t))
Graph Operation (Command)

Reify edges to nodes or reconfigure the graph:

$ \textit{penman} \ ex.txt \ --amr \ --reify-edges$

\begin{verbatim}
(w / want-01
 :ARG1-of (_ / have-polarity-91
 :ARG2 -)
 :ARG0 (t / they)
 :ARG1 (g / go-02
 :ARG0 t))
\end{verbatim}

$ \textit{penman} \ ex.txt \ --amr \ --reconfigure=random$

\begin{verbatim}
(w / want-01
 :ARG0 (t / they
 :ARG0-of (g / go-02))
 :polarity -
 :ARG1 g)
\end{verbatim}
Using penman in Python

Demo
Loading and Inspecting Data (API)

The Python API can do some things the penman command cannot, such as graph inspection.

```python
>>> import penman
>>> amrs = penman.load('ex.txt')  # load returns a list
>>> amrs[0]
<Graph object (top=w) at 140705147194816>
>>> for triple in amrs[0].triples:
...     print(triple)
...
    ('w', ':instance', 'want-01')
    ('w', ':polarity', '-')
    ('w', ':ARG0', 't')
    ('t', ':instance', 'they')
    ('w', ':ARG1', 'g')
    ('g', ':instance', 'go-02')
    ('g', ':ARG0', 't')
```
More Data Inspection (API)

The graph properties can be inspected individually:

```python
>>> amrs[0].top
'w'
>>> amrs[0].variables()
{'g', 't', 'w'}
>>> [inst.target for inst in amrs[0].instances()]
['want-01', 'they', 'go-02']
>>> amrs[0].reentrancies()  # variables mapped to number of reentrancies
{'t': 1}
```
Manipulation (API)

Or edited:

```python
>>> amrs[0].triples.remove(('w', ':polarity', '-'))
>>> amrs[0].triples.extend([
...     ('g', ':ARG4', 'p'),
...     ('p', ':instance', 'park')])
>>> amrs[0].metadata['snt'] = 'They want to go to the park.'
>>> print(penman.encode(amrs[0]))
# ::snt They want to go to the park.
(w / want-01
     :ARG0 (t / they)
     :ARG1 (g / go-02
         :ARG0 t
         :ARG4 (p / park)))
```
Removing Senses (API)

A longer example: removing sense suffixes to reduce sparsity

```python
>>> import re
>>> sense = re.compile(r'\-\d+(\$|~)')
>>> def desense(branch):
...     role, tgt = branch
...     if role == '/':
...         tgt = sense.sub(r'\1', tgt)
...     return role, tgt  # modified target
...  
>>> t = penman.parse('(s / swim-01~e.1 :ARG0 (i / i))')
>>> for _, branches in t.nodes():
...     branches[:] = map(desense, branches)
...  
>>> print(penman.format(t))
(s / swim~e.1
 :ARG0 (i / i))
```
Conclusion
Penman is open source (MIT) and easy to get:
Install the latest version from PyPI:
  • pip install penman

Read the documentation:
  • https://penman.readthedocs.io/

Contribute to Penman:
  • https://github.com/goodmami/penman
Thanks!