AIML - Sets and Maps in AIML 2.0

AIML - Sets and Maps in AIML 2.0

Draft
July 23, 2014
Richard Wallace
ALICE A.I. Foundation

The AIML 2.0 draft standard includes a new feature for representing sets and maps. AIML sets are collections of words and phrases, and AIML maps are functions that transform the element of one set into an element of another set. For example, we can define a set of animals {dog, snake, kangaroo, horse} and a map from animals to number of legs {dog:4, snake:0, kangaroo:2, horse:4}.

The set and map feature has important implications for bot learning and reasoning. They also decrease the workload on the botmaster. Before <set> existed, a botmaster might have to write a different category for each member of a set. Now the effect of large numbers of categories can be compressed into a single category.

Pattern-side <set>

AIML 2.0 defines a pattern-side <set> tag. The meaning of <set> in the <pattern> (or <that>) element is completely different than the meaning of <set> in the <template> element. On the template side, <set name="age">40</set> means set the predicate named "age" to 40. On the pattern side, the <set> tag has no attributes. The pattern-side <set> encloses the name of a set, like:

- `<set>color</set>` – a set of colors
- `<set>company</set>` – a set of companies
- `<set>verb3sp</set>` – a set of 3rd person verbs, simple form, present tense.

The AIML 2.0 draft standard does not specify where these sets are defined. This detail is left up to the implementation of the AIML interpreter. In Program AB, the each bot has a subdirectory called sets/ where the sets are defined in text files. These files have the format of one set element per line of text, for example the file color.txt begins with:

```
Air Force blue
Alice blue
Alizarin crimson
Almond
Amaranth
Amber
American rose
Amethyst
…
```

In pattern matching, the <set> element works much like the wildcard elements _ or *. Specifically,

1. - The <set> element can match one or more words
2. - If the input words are not found in the set, the match fails.
3. - The AIML matching order is modified as follows:
    a. Try underscore _ first.
    b. Try to find an exact word match.
    c. **Try to find a <set> match -** *this step differs from AIML 1.1*
    d. Try star * last.
4. - Like the case for wildcard matching, the AIML pattern matcher first tries a one word set match, then a two word match, then three words and so on, until there are no more input words. This implies that a shorter match has precedence over a longer one, when one set element is the prefix of another one. Care

should be taken, for example, with a set of verbs that contains “give” and “give up”--The “give” will match first.
5. - The words that match the <set> element are accessible in the <template> with <star/>, as though the <set> element was another wildcard.

Let’s look at a few simple categories using <set> to see how these rules apply.

```
<category><pattern>MY FAVORITE COLOR IS <set>color</set></pattern>
<template>
<set name="favoritecolor"><formal><star/></formal></set> is a nice color.
<think><set name="it"><set name="topic"><star/></set></set></think>
</template>
</category>

<category><pattern>MY FAVORITE COLOR IS *</pattern>
<template>I didn't recognize <star/> as a color.</template>
</category>

<category><pattern>MY FAVORITE COLOR IS GREEN</pattern>
<template><set name="favoritecolor">Green</set> is my favorite color too! <think><set name="it"><set
name="topic">Green</set></set></think></template>
</category>

<category><pattern>I LIKE <set>color</set></pattern>
<template><srai>MY FAVORITE COLOR IS <star/></srai></template>
</category>
```

These categories permit the following sample dialogs.

```
Human: My favorite color is red
Robot: Red is a nice color.
```

Comment: The input matched the pattern MY FAVORITE COLOR IS <set>color</set>. The bot was able to access “Red” through the <star/> tag.

```
Human: My favorite color is Air Force blue
Robot: Air Force Blue is a nice color.
```

Comment: The set may contain phrases of more than one word. The phrase “Air Force Blue” is in the color set.

```
Human: My favorite color is green
Robot: Green is my favorite color too!
```

Comment: There is another category with the pattern MY FAVORITE COLOR IS GREEN. Because exact word matching has priority over <set> matching, the input “My favorite color is green” matches this more specific pattern. The bot responds with “Green is my favorite color too!”

```
Human: My favorite color is Paris in the spring
Robot: I didn't recognize Paris in the spring as a color.
```

Comment: The phrase “Paris in the spring” was not in the color set. The <set> match failed, and the pattern matcher moved on to the category with the pattern MY FAVORITE COLOR IS *.

```
Human: I like violet blue
Robot: Violet Blue is a nice color.
```

Comment: The final category applies a reduction whenever the user says “I like X” and X is the name of a color.

Template-side <map>

The AIML <map> tag transforms the element of one set into an element from another set. The first set is called the domain of the map, and the second set is called the range. (Note that the AIML <map> tag is distinct from the AIML <oob> <map> tag. The OOB <map> tag finds a location on a map. The AIML 2.0 <map> tag has an attribute "name" that specifies the name of a map.

`<oob><map>Los Angeles</map></oob>` - means open a mapping application and display Los Angeles.

`<map name="city2state">Los Angeles</map>` - looks for a map called "city2state" and returns the mapping of "Los Angeles", presumably "California".

As is the case for pattern-side <set>, the AIML 2.0 draft specification does not say how the maps are defined. This is left up to the implementation. In Program AB, the bot has an associated maps/ subdirectory where the maps are defined in text files. The format of these files is one mapped pair per line, separated by a colon, for example:

```
state2capital set:

Maine:Augusta
California:Sacramento
New York:Albany
Illinois:Springfield
```

Simple aiml for that:

```
<category>
<pattern>What is the capital of <set>state</set></pattern>
<template><map name="state2capital"><star/></map></template>
</category>
```

The template-side <map> element uses the attribute "name" to specify the named map.
If the named map does not exist, <map> will return a default value ("unknown" in Program AB). If the map exists, the interpreter tries to find the specified element in map domain. If the element is not in the domain, the <map> returns another default value (currently also "unknown" in Program AB).

If the map exists and the domain element is found, the <map> tag returns the associated element from the map range.

To understand how <map> works, let's look at a test category:

```
<category><pattern>* <set>verb2sp</set> *</pattern>
<template>Who <star index="2"/> <star index="3"/>?  <star/>.
What does <star/> <map name="verb2sp2verb1sp"><star index="2"/></map>? <star index="3"/>.
</template>
```

Meaning of pattern-side <set> and template-side <map>:

`<set>verb2sp</set> -` the matching words are members of a set called verb2sp, defined in the sets/ directory. [The notational convention here means 2nd person verbs (2), simple form (s), present tense (p)].

`<map name="verb2sp2verb1sp">X</map> -` using the map "verb2sp2verb1sp", return the map entry for key X.
The map is defined in the maps/ directory.

`<star index="2"/> -` the matching words found in the set may be accessed as though <set></set> was another *.

Simple demo conversation:

```
Human: joe does work
Robot: Who does work?  joe.
```

```
What does joe do? work.

Human: jane seems tired
Robot: Who seems tired?  jane.
What does jane seem? tired.

Human: Steve thinks AIML sets and maps are cool.
Robot: Who thinks AIML sets and maps are cool?  Steve.
What does Steve think? AIML sets and maps are cool.
```

The set verb2sp is

```
has
does
says
gets
makes
goes
knows
takes
sees
comes
thinks
looks
wants
gives
uses
finds
tells
asks
works
seems
feels
tries
leaves
calls
```

The map verb2sp2verb1sp is

```
has:have
does:do
says:say
gets:get
makes:make
goes:go
knows:know
takes:take
sees:see
comes:come
thinks:think
looks:look
wants:want
gives:give
uses:use
finds:find
tells:tell
asks:ask
works:work
seems:seem
feels:feel
tries:try
```

```
leaves:leave
calls:call
```

The simplified categories above do not contain a <learn> or <learnf> tag, but their templates illustrate the potential for improved bot learning.

Reversed AIML

Charles Chevallier coined the term “Reversed AIML” to refer to a feature where a bot can transform factual statements into new AIML categories. For example, from the statement “Joe does work” the bot can write new AIML categories to answer the questions “What does Joe do?” and “Who does work?” The new AIML is the “reverse” of the input statement. Using sets and maps, it becomes much easier for the botmaster to write Reversed AIML categories. By building a new category with a <learn> tag, the bot can answer new questions:

```
Human: Joe does work.
Robot: I’ll remember Joe does work.
Human: Who does work?
Robot: Joe.
Human: What does joe do?
Robot: work.

Joe makes money. → Who makes money?
Mercedes makes cars.  → What makes cars?
```

Exercise: Write the <learnf> expressions into the sample categories in this section, so that the bot can actually answer the new questions.

Answer: Here is one way to do it:

```
<category><pattern>QUESTIONWORD <set>name</set></pattern>
<template>Who</template>
</category>

<category><pattern>QUESTIONWORD <set>name</set> *</pattern>
<template>Who</template>

</category>
<category><pattern>QUESTIONWORD *</pattern>
<template>What</template>
</category>

<category><pattern>* <set>verb2sp</set> *</pattern>
<template><think>
<set name="learnpattern"><srai>QUESTIONWORD <star/></srai> <star index="2"/> <person><star index="3"/>
</person></set>?
<set name="learntemplate"><star/></set>.
<learnf>
  <category>
  <pattern><eval><get name="learnpattern"/></eval></pattern>
  <template><eval><get name="learntemplate"/></eval></template>
  </category>
</learnf>
</think>
Now you can ask me: "<get name="learnpattern"/>?"
<think>
<set name="learnpattern">What does <star/> <map name="verb2sp2verb1sp"><star index="2"/></map></set>?
<set name="learntemplate"><person><star index="3"/></person></set>.
<learnf>
  <category>
  <pattern><eval><get name="learnpattern"/></eval></pattern>
  <template><eval><get name="learntemplate"/></eval></template>
  </category>
</learnf>
</think>
and "<get name="learnpattern"/>?"</template>
</category>
```

External (Remote) Sets and Maps

For large sets and maps with thousands to hundred of thousands of members, it may be undesirable to spend the memory resources needed to load the set on a small device. As an alternative, Program AB allows the specification of external sets and maps using a remote server.

Remote Set

The steps to building and using an external set are:

1. Write an AIML file containing a category for each member setMember of set setName, where the pattern is specified as

`“ISA“+setName+“ “+setMember`

and the template is

`“<template>true</template>“`

For example, if the setName is “color”, the category for “red” would be

```
<category>
<pattern>ISACOLOR RED</pattern>
<template>true</template>
</category>
```

Note that it is fairly simple to write a program (in nearly any language) that can read a file of set members and write this file of AIML categories.

2. Add one more category to indicate that anything not found in the set is not a member, for example:

```
<category>
<pattern>ISACOLOR *</pattern>
<template>false</template>
</category>
```

3. Create a Pandorabot on a Pandorabots server and upload the AIML file. Publish the bot, and make note of the host name and bot id.

4. In the Program AB sets/ directory, create a text file with the name of the set, like color.txt. Add one line to that file that starts with “external:” and has a form like:

```
external:www.pandorabots.com:8e3fc4b44e342400:4
```

- The “external” keyword means this is an external set.
- The “www.pandorabots.com” specifies the host name.
- The number 4 is the maximum length (in words) of any member of the set. This parameter saves processing and networking resources by omitting checks of phrases that are too long to be in the set anyway.

5. When Program AB loads the sets in the sets/ directory, it detects the external set.

6. Use the <set> tag normally in pattern expressions. If color is an external set,

```
<pattern>IS <set>color</set> A COLOR</pattern>
```

will check to see if input words between “IS” and “COLOR” are members of the external set.

Remote Map

The steps to define a remote map are nearly identical to the steps for a remote map.

1. Write an AIML file containing a category for each pair domainElement, rangeElement in map mapName, where the pattern is specified as

```
mapName+“ “+domainElement
```

and the template is

```
“<template>“+rangeElement+“</template>“
```

For example, if the setName is “zip2city”, the category for “94618” would be

```
<category>
<pattern>ZIP2CITY 94618</pattern>
<template>Oakland, California</template>
</category>
```

Note that, as was the case for sets, it is fairly simple to write a program (in nearly any language) that can read a file of set members and write this file of AIML categories.

2. Add one more category to indicate that for anything not found in the domain, the mapping is unknown.

```
<category>
<pattern>ZIP2CITY *</pattern>
<template>unknown</template>
</category>
```

3. Create a Pandorabot on a Pandorabots server and upload the AIML file. Publish the bot, and make note of the host name and bot id. (You can use the same bot as the one for sets).

4. In the Program AB maps/ directory, create a text file with the name of the map, like zip2city.txt. Add one line to that file that starts with “external:” and has a form like:

```
external:www.pandorabots.com:8e3fc4b44e342400
```

- The “external” keyword means this is an external set.
- The “www.pandorabots.com” specifies the host name.
- Note that the maximum element length need not be specified for maps, as it was for sets. This is because the <set> element can be activated on the <pattern> side much more frequently than the <map> element on the <template> side.

5. When Program AB loads the maps from the maps/ directory, it detects the external map.

6. Use the <map> tag normally in pattern expressions. If zip2city is an external set,

```
<template>The city for 94618 is <map name=”zip2city”>94618</set></pattern>
```

will send a request to the remote server of the form “ZIP2CITY 94618” and, if everything is working correctly, the remote server will return “Oakland, CA”.

Built-In Sets and Maps

A few AIML Sets and Maps are built in to the interpreter:

- *number* - the set of natural numbers {0, 1, 2, …}
- *successor* - The function f(x) = x + 1. (Returns “unknown” if x is not a number)
- *predecessor* - The function f(x) = x -1. (Returns “unknown” if x is not a number)
- *singular* - The mapping from plural nouns to their singular form (English only)
- *plural* - The mapping from singular nouns to their plural form (English only)