ChatScript Advanced Concept Manual

Copyright Bruce Wilcox, gowilcox@gmail.com www.brilligunderstanding.com

ADVANCED CONCEPTS

Concept Exclusion

In basic chatscript we learned you can build concepts by augmentation (out of concepts), like

```
concept: ~animals (~birds ~dogs otter)
```

You can also build concepts using exclusion, like

```
concept: ~animals (~birds ~dogs otter !robin)
```

This concept includes all birds except the robin. Using !, you can tell CS that certain words are not members of a concept, even though they may have been added elsewhere in the declaration either directly or via inclusion of some concept. You can also use ! with concepts to remove all members of a concept. E.g.

```
concept: ~wildanimals (!~pet_animals ~animals)
```

Animals which are pets are not considered wild, so here is a clean declaration of that.

Fundamental Meaning Keywords

Fundamental meaning is the basic minimal form of the sentence without all the embellishments of phrases, verbals, clauses, adjectives, and adverbs. Your fundamental sentence consists of main subject, main verb, and optional main object. The absolute minimal sentence always has a main verb. In the case of "Go", we have a command verb and implied subject "you".

Fundamental meaning consists of an actor, an action, and an optional actee. In the active voice sentence "I love you", the actor is "I", the action is "love", and the actee is "you". In the passive voice sentence "I was arrested", there is no actor, the verb is "arrested", and the actee is "I". Wherease in the passive voice sentence "I was arrested by the police", the actor is "police".

Fundamental meaning patterns always have a verb, which as a keyword is designated as
"|arrest|" or whatever word or concept you want to detect. A pattern which includes a fundamental actor is shown as
"pronoun|arrest|". One that includes an actee is"|arrest|police", whereas one that has both actor and actee is "pronoun|arrest|police". So one can write:

```
concept: ~crimeverbs (arrest convict imprison steal)
topic: ~crimesentences (|~crimeverbs|)
```

For command sentences, the implied subject is always "you", so you can write:

```
concept: ~me_told_go (you|~movement_verbs|)
```

Note: these keywords can only be recognized if the system's parser can manage to parse out the main subject, main verb, and main object of the input sentence. This works well for relatively simple sentences.

Patterns as Keywords

Instead of just words and phrases, you can also write a pattern in quotes as a member.

```
concept: ~leaving( sayonara "(going home now)" exiting)
u: (I *~2 ~leaving)
```

You are allowed to use any of the pattern elements excepting RETRY but including "(*testing*!testval)".

Where the pattern is considered marked will depend on the pattern. If the pattern has match variable memorization, it will mark where the first such memorization happened. If the pattern consists of no words (like the variable testing pattern above), then the concept will be marked at word 1 of the sentence. Otherwise, if the pattern is has a normal collection of words, the mark will cover the range of the first thru last word element that matched. For our leaving example above, if the input is "I am going home now, see you tomorrow", then ~leaving marks words 3-5 of the sentence.

The function ˆpick, which can grab a random member of a concept, will skip over all patterns and just return a word or phrase.

Additional data on concepts

Concepts can have part of speech information attached to them (using `dictionarysystem.h` values). Eg.

```
concept: ~mynouns NOUN NOUN_SINGULAR (boxdead foxtrot)
concept: ~myadjectives ADJECTIVE ADJECTIVE_BASIC (moony dizcious)
```

Since the script compile issues warning messages on words it doesn't recognize, in case you misspelled them, you can also add `IGNORESPELLING` as a flag on the concept:

```
concept: ~unknownwords IGNORESPELLING (asl daghh)
```

and you can combine pos declarations and ignorespelling. This is applied recursively to any concepts that are members of this concept. That may be a bit excessive.

Rather than assigning parts of speech you can recursively limit a concept's words to a part of speech using `ONLY_NOUNS`, `ONLY_VERBS`, `ONLY_ADJECTIVES`, or `ONLY_ADVERBS`.

```
Concept: ~verbs ONLY_VERBS (sit sleep)
```

This will not react to noun meanings of sleep. The current ontology files for verbs, adverbs, and adjectives all have the appropriate `ONLY` marked on them.

When you don't want a member concept marked as a consequence, you can use `ONLY_NONE` to block propogate. Thus:

```
concept: ~verbs ONLY_VERBS (~active_verbs sit)
concept: ~active_verbs ONLY_NONE (sleep)
```

will prevent sleep from being required to be a verb form. Note that verb forms do not include verbs used as nouns (ie gerunds).

Normally if you declare a concept a second time, the system considers that an error. If you add the marker `MORE` to its definition, it will allow you to augment an existing list.

```
concept: ~city MORE (Tokyo)
```

Normally concepts (and topics) discard repeated keywords. For concepts, you can force it to allow repeats using `DUPLICATE`

```
concept: ~mapword DUPLICATE (year month day year month day) # the concept has 6 members
```

Concepts can be built from other concepts that do not have specific words.

```
Concept: ~myconcept (!thisword ~otherconcept)
```

Note: the system has two kinds of concepts.

- *Enumerated* concepts are ones formed from an explicit list of members. Stuff in definitions of `concept: ~xxx()` are that.
- There are also *internal* concepts (dynamic concepts) marked by the system. These include part of speech of a word (requires using the pos-tagger to decide from the input what part of speech it was of possibly several), grammatical roles, words from infinite sets like `~number` and `~placenumber` and `~weburl`, and so forth.

The ? operator has two forms. `xxx?~yyy` will look for actual membership in the set whereas `_n?~yyy` will only see if the location of match detection of _n is the same as a corresponding match location for the concept. If the concept has not been marked, then obviously no match is found.

In a pattern of some kind, if you are referencing a sentence location using a match variable, you can match both kinds enumerated and dynamic concepts. But if you are not tied to a location in a sentence, you can't match internally computed ones. So something like

```
if ( pattern 23?~number )
```

will fail. Even

```
if ( pattern practical?~adjective )
```

will fail given that deciding practical is an adjective (it could also be a noun) hasn't been performed by pos-tagging.

All internal concepts are members of the concept `~internal_concepts`.

Alternate ways of populating a Concept

Normally concepts are an enumeration set you define like this:

```
concept: ~myconcept (word1 word2 "phrase of mine")
```

But you can also extend a concept merely by creating facts:

```
    $_tmp = ^createfact(word3 member ~myconcept)
```

Concept membership is actually represented by facts of the above form.

A third way does not actually create members of a concept and does not require mere words and phrases. You can use any pattern whatsoever. And you cannot list it like you would normal members of a concept. You meremly execute a rule early on in your script that marks places in a sentence.

```
u: FAKEIT ( _(find * way)) ^mark(~myconcept _0)
```

The above detects a pattern, records the location start and end of it so that you can then mark where in the sentence ~myconcept will now be detected by any future patterns of yours.

<<<<<<< HEAD

Performance

Using concepts in patterns is “free”, as is

```
    if (_0 ? ~concept)
```

but

```
    if ($word ? ~concept)
```

runs linear with number of concept members, so it should be avoided for large concepts. Pattern matching with concepts/match_variables has all its overhead cost paid for during marking so there is no real cost later. And that marking overhead is trivial. ======= >>>>>>> 64d7d51a5e09623a04d749a68f2b39584e181144