Automatic Programming Via Text Mapping To Expert System Rules

Pedro V. Marcal

Mpact Corp., Oak Park, Ca., USA

pedrovmarcal@gmail.com

Abstract

Starting from semantically parsed text, a program was developed to split up all compound sentences to simple sentences. These sentences were converted to expert systems rules and then processed by a tailored Expert System program. Successful execution of the Expert System program demonstrated a form of Automatic Programming.

Keywords: Automatic Programming, Expert Systems, Knowledge Base, A. I.

Introduction

Computer Programming has always been regarded as a labor intensive process. In the early days of Artificial Intelligence, it was recognized that there was a duality between data (including text) and program and that the two were interchangeable. This was summarized in the mantra, 'Data is Program and Program is Data'. There was therefore many attempts to develop systems for automatic programming. These systems were largely unsuccessful see for example [1]. So this complex task was split up into niche processes. For example programs were written to convert mathematical equations into programs, eg. Matlab and Mathematica and open source Sympy. Another approach was to develop Domain Specific Languages [1]. The main reason for the failure to develop a general system was the computer's inability to understand text. Liu et al. [2] developed a system based on text understanding by requiring the text take the form of a story. With this approach [2] was able to develop a top level schematic programming system while leaving the details to be filled in by the traditional method.

It is obvious that in order to convert text to program, we must first understand the text. Marcal[3] developed a semantic parser in two steps. First, the text was processed by a context free parse in English. The Semantic parse was accomplished by a translation of the text into Simplified Chinese (Mandarin). English and Chinese are orthogonal in meaning. One English word has many meanings because it is based on phonetics. Whereas one Chinese word (set of characters) has only 1 meaning, derived from its ideograph. So there are many Chinese Words with the same meaning. A statistical parsing method was developed in [3] that used Design Of Experiments[4] to reduce the search through the possible combinations of meaning to provide the optimal translation. The statistic was based on ngram of 3 (sequence of 3 words) . This method reduced the computational effort of parsing by two orders of magnitude. This method required good Corpora in both English and Chinese. In addition, using Wordnet and the two Corporas as a basis, the author constructed a Lexical Dictionary that covered most of the English and Chinese Languages. The result of the semantic parse (translation to Chinese) was reflected back to English in the form of an unambiguous sentence. This is the starting point for the current project.

In the early 1970s , Expert systems were regarded as a promising approach to Artificial Intelligence[5]. The writer took part in a project to explain the principles behind a nonlinear Finite Element Program, MARC to the level that enabled interested Engineers to use the program. To this end, the SACON (Stress Analysis Consultant) was developed based on EMYCIN. This project also had important ramifications for the Expert System Developers because for the first time the rules encompassed a

complete Domain. This emphasized the need for completeness in a system. In addition the Domain could be used to develop systems of Computer assisted instructions. In further work, Racz and Marcal[6] extended expert Systems to also call functions and external Programs. In the early days of nonlinear analysis, it was usual to employ Ph. D.s to perform such analysis. The Nuclear Industry had need for a large number of analysis of their components. Two Japanese Companies were able to develop expert systems for the MARC program[7]. It found that Engineering Aides with High School Diplomas were able to perform such nonlinear analysis (I assume under close supervision). The problem with the Expert Systems was that it required specialists to elicit the understanding of the experts and then codify it in the form of rules. It was in fact another form of programming and twice as laborious because the coding was one removed.

Objective

The objective of the current work is to start with the semantic parse of text and split any compound sentences to a sequence of simple sentences which we will call hyper-sentences. These hyper-sentences form simple concepts of the Subject,Verb, Object (SVO) type. We will call these concepts hyper-concepts. It was found that these hyper-concepts are automatically mapped into expert systems rules, called hyper-rules. An expert system was developed to process these hyper-rules in the usual way. The successful development of an expert system constitutes the form of automatic programming that we seek. The hyper-concepts are of interest in their own right and can be classified in the usual hierarchical way, similar to the Wordnet scheme with hypo and hyper relations. In order to assist in this, we adopt the Conceptual Dependency (CD) of Schank[8-9]. Schank and his colleagues showed that the verbs in hyper-concepts could be represented by sixteen hypo-verbs and that these verbs can be classified in three states, viz Property , Physical and Mental respectively. In CD theory these verbs designate actions that are labeled ATRANS, PTRANS and MTRANS respectively.

The following are some examples of the hyper-verbs.

. TRANS transfer (of possession,location, and of ideas)

. MOVE, PROPEL, GRASP movement and forces

.INGEST, EXPEL absorb and its opposite

.ATTEND, SPEAK, RECORD sense, verbal output and write or record

.BUILD construct.

.COMPUTE compute

.STMEM,LTMEM short term remember and long term remember

.TIME passage of time

.LIVE biological

.DO any other verbs that can not be classified above.

Finally, we adopt Schank's construction of a hyper sentence instead of the traditional SVO, we use PP (Picture Painter), ACT (Action), Ppo(Picture Painter Object). The picture Painters are in turn modified by Picture Aiders (Adjectives, prepositions). The ACT are in turn modified by ACT Aiders (AA,adverbs)

Since any of these verbs can be classified in the three states above, we have expanded the original CD to have a total of 48 possible classification. In the original CD theory Schank did not attempt to classify the Subject and Object phrases. In the current work, we expanded CD to also classify these phrases

using a Wordnet type generalization. With such an approach it was found that the hyper concepts took on their own properties. These hyper-concepts were found to follow Zipf's Law with similar types of properties as that found for words which were originally found to obey Zipf's Law.

Theoretical Considerations

There is little additional theory to consider above that already discussed. The difficulties lay in its algorithmic implementation as a computer code. The following is a computer flow of the program. It is conveniently separated into two programs. The first develops the hyper-rules while the second processes the rules in an expert system. The programs are written in Python 2.7. Python's string processing features has made the coding easier and its dictionary with its seamless in-core and out of core storage has proven invaluable.

Rule development for each sentence in sequence.

1. Context-free Parse sentence.
2. Semantic Parse by translation into Chinese.
3. Chunk the phrases and transfer their lexical meaning back to English. (for ease of use).
4. Split the phrases into hyper-sentences with hyper-concepts.
5. Define the cd hypo concept for each hyper sentence.
6. Convert each hyper sentence into a hyper rule.
7. Separate rules into two categories. In the first the verbs are not modified by any adverbial or ACT Aiders. These rules are defined once and for all. The second category contains ACT Aiders and these form conditions in the hyper-rules. We call these eligible rules (for expert system processing)
8. Search for rule conclusion in the hyper-concepts belonging to the same original sentence or following the original sentence. These are marked by prepositions such as then (PP Aiders).

We note that each hyper-rule carries its own words and its conceptual dependency. Because each phrae may be expressed in a myriad of different word combinations, the CD takes on special significance for processing in an expert system. This is in fact the key to converting hyper-concepts into hyper-rules for expert systems. Every paragraph exists for a reason. The paragraph usually answers one of the following questions viz. what, where, when, how? These are then the objective or in expert system parlance the golden rule for each paragraph. In most cases, the first appearance of such a preposition (PPA) denotes a golden rule. If a golden rule is not obvious the expert system prompts the user or asks the user to accept its best estimate.

There is a certain amount of pre processing required to prepare the hyper-rules for efficient processing by the expert system. Lists such as menu items are collected for convenience. It is important that critical eligible rules with multiple conditions be identified, and their transfer locations be identified as system switches. Once transferred to a system switch, the eligible rules are processed in sequence until another switch is encountered. Then control is transferred back to the original switching rule, for execution of the next rule.

Development of Expert system.

1. Pre process rules for efficiency. Each eligible hyper rule is assigned an event. Each event contains a detailed analysis of the rule as to how it affects the expert system.

2. Identify Golden Rules. (noted in event)
3. Identify switching rules. (as compound rules).(noted in event)
4. Identify system switches. (noted in event)
5. Execute rules in sequence. Record each executed rule in a journal.
6. Contact the user for unresolved rules that cannot be processed further. It is at this stage that the dialog is constructed to allow the user to query the actions taken and recorded in the journal. Usually, the actions taken to remedy the situation requires a new text and a repeat of the hyper-rule construction in the first step above.
7. When all the eligible rules have been satisfied, save the conclusions and the journal and exit the program.

Case Study

In this case study, we repeat the problem used by Liu et al [2] to automatically program the solution. The text describes a bar in the following.

This is a bar with a bartender, who makes drinks.
The bar has a menu containing some drinks, which include : a sour apple martini, a margarita, and rum and coke.
When a customer orders a drink, the bartender tries to make it. When the bartender is asked to make a drink, he makes it and gives it to the customer only if the drink is in the menu's drinks.
Otherwise, the bartender says to the customer, ' Sorry I don't know how to make that drink '.
In [2], the solution is neatly encased in the following Class using its internal format.

```
class bartender :
   def make(drink) :
        if (drink in menu.drinks) :
           bartender.make(drink)
           bartender.give(drink, customer)
        else :
           bartender.say( \
                "Sorry I don't know how to make that drink.", customer)
```

The solution here follows the same lines, except that there is less need for collection of concepts that relate to each other. This is implied in the rules and is automatically recognized by the expert system.

The parsed hyper-concepts are listed in Appendix A. These then are turned into hyper-rules which are listed in Appendix B.

The hyper-rules are paraphrased in the following.

Rule (4000,0) defines bar as a structure

Rule (4000,1) defines bartender makes drink.

Rule (4001,0) Defines existence of menu.

Rule (4001,1) Defines menu as containing drinks.

Rule (4001,2) Defines drinks list.

Rule (4002,0) Customer requests drink, Golden Rule.

Rule (4002,1) Bartender makes it, but there is a restriction (either option A or B)

Rule (4003,0) Bartender figures out rule.

Rule (4003,1) Action if option.

Rule (4003,2) Action if option.

Rule (4003,3) Condition for option A, System Switch A.

Rule (4003,4) Give drink to customer as per option A.

Rule (4004,0) Condition for Option B, implied System Switch B.

Rule (4004,1) Bartender speaks an apology.

Rule (4004,2) Excuse is does not know how to make.

Rule (4004,3) That drink.

End of Rules.

The processing of the rules gave the same results as the automatic coding by Liu et al [2].

Hence we conclude that we have achieved our objective of automatic programming of rules for an expert system.

Discussion and further work

The ability to convert text to programs is very important. Mainly because most of our complete history and culture is recorded as text. In this project we have achieved this in a general way. This process may also be extended to obtain summaries from text by systematically asking the questions. What, where, how, why? We would then need to extend this to querying the internet to provide related text. The important action would be to systematically store the texts processed so that the rules can be retrieved as and when they are required. The Watson program[10] does an extensive job in collecting a knowledge base. The difference with the current program is that [10] does not do such a detailed job of parsing and labeling as it is done here..

Another direction that this development can proceed is to set up a mixture of mathematical equations sprinkled with text to control the results of the computing. This process is similar to the current coding in CAE, but here robotized.

Finally for this process to act in real time, it must be accelerated by parellization. It currently takes on average about 33 secs. to process a sentence on a PC.

Conclusions

A program has been developed for generating hyper-rules from text.

These hyper-rules can be processed in a tailored Expert System to achieve automatic programming.

We have achieved the first step in our objective to develop an automatic way of converting text to programs.

References

[1]. C. Rich, R.C. Waters,' Readings in Artificial Intelligence and Software Engineering.' Morgan Kaufman Publishers, 1986.

[2] H. Liu, H. Lieberman,'Metafor: Vizualizing Stories as Code',Proc. IUI'05, San Diego, CA, 2005.

[3]. P.V. Marcal, 'Development of a General Semantic Reader (GPSR)' ,available on Research Gate by following my research, Jan. 1, 2012

[4] P.V. Marcal, and J. Fong, 'A Design-of-Experiments approach to Statistical Parsing of a Natural Language Abstracting Code for Fatigue Data Event Databases', Proc ICCES, June, 2014, Korea.

[5]. P.V. Marcal, "Knowledge Engineering and Artificial Intelligence"; Advanced Topics and New Developments in Finite Element Analysis, ASME WAM, Monterey, California, (July, 1978). Result of work with R.J. Melosh and E.A. Feigenbaum on the development of SACON.

[6].S. Racz, and P.V. Marcal, "SACON2, An Expert System for Automating the FEM Process", Proc. WCCM, 2006

[7]. S. Hiromi, I. Ishihara,H. Kobayashi,M. Kajiwara,'Developm,ent of Expert System MARCAS Supporting the Preparation of MARC Input Deck', Proc.MARC, Users Conference, CA., 1988.

[8] R. Schank, and C.K. Riesbeck, 'Inside Computer Understanding.', editors, Lawrence Erlbaum Associates , 1972

[9]. A.G.Francis, Jnr.'A Pocket Guide To CD Theory', Lecture Notes, College of Computing, Georgia Inst. Technology, GA, 1974,

[10] D. Ferruci, Watson, The Deep Qa Project, IBM Research, Feb, 2011.

Appendix A: details of parsing

```
 *** WriteDict *** hyper_concept
hyper_concept  ('4000', '0')   value
('4000', '0')
['cd_concept', 'object', 'is', 'equal', 'bar', 'ACTION', 'PAo',
'what_alliance', 'human']
[['PP', '0'], ['this', 'PP', 'object', 'ATRANS']]
[['ACT', '1'], ['is', 'ACT', 'equal', 'ATRANS']]
[['PPo', '2'], ['bar', 'PP', 'structure', 'PTRANS']]
[['PAo', '3'], ['with', 'P', 'possess', 'MTRANS']]
[['PAo', '4'], ['bartender', 'PP', 'human', 'PTRANS']]
['PP', 'this', 'object', 'ATRANS', 'ACT', 'is', 'equal', 'ATRANS', 'PPo',
'bar', 'structure', 'PTRANS']
hyper_concept  ('4000', '1')   value
('4000', '1')
['cd_concept', 'human', 'make', 'build', 'material', 'ACTOR']
[['PP', '6'], ['who', 'PP', 'human', 'ATRANS']]
[['ACT', '7'], ['make', 'ACT', 'build', 'PTRANS']]
[['PPo', '8'], ['drink', 'PP', 'food', 'MTRANS']]
['PP', 'who', 'human', 'ATRANS', 'ACT', 'make', 'build', 'PTRANS', 'PPo',
'drink', 'food', 'MTRANS']
hyper_concept  ('4001', '0')   value
('4001', '0')
['cd_concept', 'bar', 'have', 'ingest', 'signal', 'ACTION']
[['PP', '0'], ['bar', 'PP', 'structure', 'PTRANS']]
```

```
[['ACT', '1'], ['have', 'ACT', 'ingest', 'ATRANS']]
[['PPo', '2'], ['menu', 'PP', 'record', 'MTRANS']]
['PP', 'bar', 'structure', 'PTRANS', 'ACT', 'have', 'ingest', 'ATRANS',
'PPo', 'menu', 'record', 'MTRANS']
hyper_concept  ('4001', '1')   value
('4001', '1')
['cd_concept', 'signal', 'contain', 'do', 'material', 'ACTION']
[['ACT', '3'], ['contain', 'ACT', 'do', 'PTRANS']]
[['PPo', '4'], ['some', 'PA', 'quantity', 'ATRANS']]
[['PPo', '4'], ['drink', 'PP', 'food', 'MTRANS']]
['ACT', 'contain', 'do', 'PTRANS', 'PPo', 'drink', 'food', 'MTRANS']
hyper_concept  ('4001', '2')   value
('4001', '2')
['cd_concept', 'object', 'include', 'do', 'material', 'ACTION', 'PAo',
'substance.n.01', 'material']
[['PP', '6'], ['which', 'PP', 'object', 'ATRANS']]
[['ACT', '7'], ['include', 'ACT', 'do', 'PTRANS']]
[['PPo', '9'], ['sour', 'PA', 'sense', 'ATRANS']]
[['PPo', '9'], ['apple', 'PP', 'food', 'PTRANS']]
[['PPo', '9'], ['martini', 'PP', 'food', 'MTRANS']]
[['PAo', '11'], ['margarita', 'PP', 'food', 'MTRANS']]
[['PAo', '14'], ['rum_and_coke', 'PP', 'food', 'MTRANS']]
['PP', 'which', 'object', 'ATRANS', 'ACT', 'include', 'do', 'PTRANS',
'PPo', 'martini', 'food', 'MTRANS']
hyper_concept  ('4002', '0')   value  ### This becomes the golden rule
('4002', '0')
['cd_concept', 'human', 'order', 'plan', 'material', 'THINK']
[['PP', '1'], ['customer', 'PP', 'money_value', 'PTRANS']]
[['AA', '0'], ['when', 'AA', 'age_destination', 'PTRANS']]  ### trigger for
###golden rule. When asked customer replies with A) drink in menu eg. ###
### Margarita or B) drink not in menu eg vodka martini.
[['ACT', '2'], ['order', 'ACT', 'plan', 'MTRANS']]
[['PPo', '3'], ['drink', 'PP', 'food', 'MTRANS']]
['PP', 'customer', 'money_value', 'PTRANS', 'ACT', 'order', 'plan',
'MTRANS', 'PPo', 'drink', 'food', 'MTRANS']
### at this point we activate a switch with value A or B respectively.
hyper_concept  ('4002', '1')   value
('4002', '1')
['cd_concept', 'human', 'make', 'build', 'knowledge', 'ACTOR']
[['PP', '5'], ['bartender', 'PP', 'human', 'PTRANS']]
[['ACT', '6'], ['try', 'ACT', 'compute', 'MTRANS']]
[['ACT', '6'], ['make', 'ACT', 'build', 'PTRANS']]
[['PPo', '7'], ['it', 'PP', 'object', 'MTRANS']]
['PP', 'bartender', 'human', 'PTRANS', 'ACT', 'make', 'build', 'PTRANS',
'PPo', 'it', 'object', 'MTRANS']
hyper_concept  ('4003', '0')   value
('4003', '0')
['cd_concept', 'human', 'ask', 'transmit', 'None', 'ACTOR']
[['PP', '1'], ['bartender', 'PP', 'human', 'PTRANS']]
[['AA', '0'], ['when', 'AA', 'age_destination', 'PTRANS']]
[['ACT', '2'], ['is', 'ACT', 'equal', 'ATRANS']]
[['ACT', '2'], ['ask', 'ACT', 'transmit', 'MTRANS']]
['PP', 'bartender', 'human', 'PTRANS', 'ACT', 'ask', 'transmit', 'MTRANS']
hyper_concept  ('4003', '1')   value ### System Switch A
```

```
('4003', '1')
['cd_concept', 'human', 'make', 'build', 'None', 'ACTOR']
[['ACT', '4'], ['make', 'ACT', 'build', 'PTRANS']]
['ACT', 'make', 'build', 'PTRANS']
hyper_concept  ('4003', '2')   value
('4003', '2')
['cd_concept', 'human', 'make', 'build', 'knowledge', 'ACTOR']
[['PP', '6'], ['he', 'PP', 'human', 'ATRANS']]
[['ACT', '7'], ['make', 'ACT', 'build', 'PTRANS']]
[['PPo', '8'], ['it', 'PP', 'object', 'MTRANS']]
['PP', 'he', 'human', 'ATRANS', 'ACT', 'make', 'build', 'PTRANS', 'PPo',
'it', 'object', 'MTRANS']
hyper_concept  ('4003', '3')   value
('4003', '3')
['cd_concept', 'human', 'give', 'trans', 'knowledge', 'UNDEF', 'PAo',
'source', 'human', 'prepD', '->', 'PAo']
[['CNJ', '9'], ['and', 'CNJ', 'coordinating_object', 'MTRANS']]
[['AA', '14'], ['only', 'AA', 'qualification_value', 'PTRANS']]
[['ACT', '10'], ['give', 'ACT', 'trans', 'PTRANS']]
[['PPo', '11'], ['it', 'PP', 'object', 'MTRANS']]
[['PAo', '12'], ['to', 'P', 'recipient', 'MTRANS']]
[['PAo', '13'], ['customer', 'PP', 'money_value', 'PTRANS']]
['ACT', 'give', 'trans', 'PTRANS', 'PPo', 'it', 'object', 'MTRANS']
hyper_concept  ('4003', '4')   value
('4003', '4')
['cd_concept', 'None', 'is', 'equal', 'signal', 'UNDEF', 'PAo',
'aspect_what_property', 'material']
[['CNJ', '15'], ['if', 'CNJ', 'subordinating_qualification', 'MTRANS']]
[['ACT', '16'], ['is', 'ACT', 'equal', 'ATRANS']]
[['PPo', '17'], ['menu', 'PP', 'record', 'MTRANS']]
[['PAo', '18'], ['of', 'P', 'possess', 'property']]
[['PAo', '19'], ['drink', 'PP', 'food', 'MTRANS']]
['ACT', 'is', 'equal', 'ATRANS', 'PPo', 'menu', 'record', 'MTRANS']
hyper_concept  ('4004', '0')   value ### Swystem Switch B
('4004', '0')
['cd_concept', 'human', 'say', 'speak', 'None', 'ACTOR', 'PAo', 'source',
'human', 'prepD', '->', 'PAo']
[['PP', '2'], ['bartender', 'PP', 'human', 'PTRANS']]
[['AA', '0'], ['otherwise', 'AA', 'property_lest', 'PTRANS']]
[['ACT', '3'], ['say', 'ACT', 'speak', 'MTRANS']]
[['PAo', '4'], ['to', 'P', 'recipient', 'MTRANS']]
[['PAo', '5'], ['customer', 'PP', 'money_value', 'PTRANS']]
['PP', 'bartender', 'human', 'PTRANS', 'ACT', 'say', 'speak', 'MTRANS']
hyper_concept  ('4004', '1')   value
('4004', '1')
['cd_concept', 'language', 'know', 'move', 'None', 'UNDEF']
[['PP', '7'], ['sorry', 'PA', 'transmit', 'ATRANS']]
[['PP', '7'], ['i', 'PP', 'human', 'ATRANS']]
[['PP', '7'], ['do', 'PP', 'None', 'ATRANS']]
[['ACT', '8'], ['know', 'ACT', 'move', 'MTRANS']]
['PP', 'do', 'solfa_syllable.n.01', 'ATRANS', 'ACT', 'know', 'move',
'MTRANS']
hyper_concept  ('4004', '2')   value
('4004', '2')
```

```
['cd_concept', 'human', 'make', 'build', 'None', 'ACTOR']
[['AA', '9'], ['how', 'AA', 'None', 'PTRANS']]
[['ACT', '10'], ['make', 'ACT', 'build', 'PTRANS']]
['ACT', 'make', 'build', 'PTRANS']
hyper_concept  ('4004', '3')   value
('4004', '3')
['cd_concept', 'money_value', 'drink', 'ingest', 'None', 'ACTION']
[['PP', '11'], ['that', 'PP', 'object', 'ATRANS']]
[['ACT', '12'], ['drink', 'ACT', 'ingest', 'ATRANS']]
['PP', 'that', 'object', 'ATRANS', 'ACT', 'drink', 'ingest', 'ATRANS']
```

Appendix B. hyper-rules used by expert system

```
 *** WriteDict *** rule_collector
Here is a brief explanation of the rules.
The first label in the rule '<define>' says its a constant rule. The rest
of the line gives the PP and the PPo phrases. The second line gives the
hyper-concept. The next lines give the prepositional phrases (PA)
In the case of an eligible rule the first label is a tag for which Adverb
influences the processing of the current rule. For example the '<if_age>'
label (hypo-classification of the word 'when') is first encountered as an
eligible rule. The components of the rule follow the same order as before.
It is also labeled as a golden rule. The same label is then used to tag the
following rules which are executed by the system. This continues until a
switch changes the control to '<if_only>' (option A of the drink order.)
Then finally control is switched to '<if_alt>' (option B of the drink
order.)

rule_collector  ('4000', '0')   value
['<define>', [['this', 'PP', 'object', 'ATRANS'], ['bar', 'PP',
'structure', 'PTRANS']]]
['<define>', ['PP', 'this', 'object', 'ATRANS', 'ACT', 'is', 'equal',
'ATRANS', 'PPo', 'bar', 'structure', 'PTRANS']]
['<define>', [['this', 'PP', 'object', 'ATRANS'], ['with', 'P', 'possess',
'MTRANS'], ['bartender', 'PP', 'human', 'PTRANS']]]
rule_collector  ('4000', '1')   value
['<define>', [['who', 'PP', 'human', 'ATRANS'], ['drink', 'PP', 'food',
'MTRANS']]]
['<define>', ['PP', 'who', 'human', 'ATRANS', 'ACT', 'make', 'build',
'PTRANS', 'PPo', 'drink', 'food', 'MTRANS']]
rule_collector  ('4001', '0')   value
['<define>', [['bar', 'PP', 'structure', 'PTRANS'], ['menu', 'PP',
'record', 'MTRANS']]]
['<define>', ['PP', 'bar', 'structure', 'PTRANS', 'ACT', 'have', 'ingest',
'ATRANS', 'PPo', 'menu', 'record', 'MTRANS']]
rule_collector  ('4001', '1')   value
['<define>', ['ACT', 'contain', 'do', 'PTRANS', 'PPo', 'drink', 'food',
'MTRANS']]
['<list>', [['some', 'PA', 'quantity', 'ATRANS'], ['drink', 'PP', 'food',
'MTRANS']]]
['<signal_list>', [('4001', '0'), ['PPo', 'menu', 'record', 'MTRANS'],
[['some', 'PA', 'quantity', 'ATRANS'], ['drink', 'PP', 'food', 'MTRANS']]]]
rule_collector  ('4001', '2')   value
['<define>', [['which', 'PP', 'object', 'ATRANS'], ['martini', 'PP',
```

```
'food', 'MTRANS'], ['margarita', 'PP', 'food', 'MTRANS'], ['rum_and_coke',
'PP', 'food', 'MTRANS']]]
['<define>', ['PP', 'which', 'object', 'ATRANS', 'ACT', 'include', 'do',
'PTRANS', 'PPo', 'martini', 'food', 'MTRANS']]
['<define>', [['sour', 'PA', 'sense', 'ATRANS'], ['apple', 'PP', 'food',
'PTRANS']]]
rule_collector  ('4002', '0')   value
['<if_age>', [['customer', 'PP', 'money_value', 'PTRANS'], ['drink', 'PP',
'food', 'MTRANS']]]
['<if_age>', [['when', 'AA', 'age_destination', 'PTRANS'], ['order', 'ACT',
'plan', 'MTRANS']]]
['<if_age>', ['PP', 'customer', 'money_value', 'PTRANS', 'ACT', 'order',
'plan', 'MTRANS', 'PPo', 'drink', 'food', 'MTRANS']]
['<system_gold>', ['acquire', 'relate_key', '<if_age>', True,
'instantiate', False, 'iterate']]
rule_collector  ('4002', '1')   value
['<if_age>', [['bartender', 'PP', 'human', 'PTRANS'], ['it', 'PP',
'object', 'MTRANS']]]
['<if_age>', ['PP', 'bartender', 'human', 'PTRANS', 'ACT', 'make', 'build',
'PTRANS', 'PPo', 'it', 'object', 'MTRANS']]
rule_collector  ('4003', '0')   value
['<if_age>', [['bartender', 'PP', 'human', 'PTRANS']]]
['<if_age>', [['when', 'AA', 'age_destination', 'PTRANS'], ['is', 'ACT',
'equal', 'ATRANS']]]
['<if_age>', ['PP', 'bartender', 'human', 'PTRANS', 'ACT', 'ask',
'transmit', 'MTRANS']]
rule_collector  ('4003', '1')   value
['<if_age>', ['ACT', 'make', 'build', 'PTRANS']]
rule_collector  ('4003', '2')   value
['<if_age>', [['he', 'PP', 'human', 'ATRANS'], ['it', 'PP', 'object',
'MTRANS']]]
['<if_age>', ['PP', 'he', 'human', 'ATRANS', 'ACT', 'make', 'build',
'PTRANS', 'PPo', 'it', 'object', 'MTRANS']]
rule_collector  ('4003', '3')   value
['<if_only>', [['and', 'CNJ', 'coordinating_object', 'MTRANS']]]
['<if_only>', [['only', 'AA', 'qualification_value', 'PTRANS'], ['give',
'ACT', 'trans', 'PTRANS']]]
['<system_switch>', ['modify', 'call_all', '<if_only>', True, 'then',
False, '<if_alt>']]
['<then>', ['ACT', 'give', 'trans', 'PTRANS', 'PPo', 'it', 'object',
'MTRANS']]
['<then>', [['it', 'PP', 'object', 'MTRANS']]]
['<then>', [['it', 'PP', 'object', 'MTRANS'], ['to', 'P', 'recipient',
'MTRANS'], ['customer', 'PP', 'money_value', 'PTRANS']]]
rule_collector  ('4003', '4')   value
['<if_only>', [['if', 'CNJ', 'subordinating_qualification', 'MTRANS']]]
['<if_only>', ['ACT', 'is', 'equal', 'ATRANS', 'PPo', 'menu', 'record',
'MTRANS']]
['<if_only>', [['menu', 'PP', 'record', 'MTRANS']]]
['<if_only>', [['menu', 'PP', 'record', 'MTRANS'], ['of', 'P', 'possess',
'property'], ['drink', 'PP', 'food', 'MTRANS']]]
rule_collector  ('4004', '0')   value
['<if_alt>', [['bartender', 'PP', 'human', 'PTRANS']]]
['<if_alt>', [['otherwise', 'AA', 'property_lest', 'PTRANS'], ['say',
```

```
'ACT', 'speak', 'MTRANS']]]
['<if_alt>', ['PP', 'bartender', 'human', 'PTRANS', 'ACT', 'say', 'speak',
'MTRANS']]
['<if_alt>', [['bartender', 'PP', 'human', 'PTRANS'], ['to', 'P',
'recipient', 'MTRANS'], ['customer', 'PP', 'money_value', 'PTRANS']]]
rule_collector  ('4004', '1')   value
['<if_alt>', [['sorry', 'PA', 'transmit', 'ATRANS'], ['i', 'PP', 'human',
'ATRANS'], ['do', 'PP', 'None', 'ATRANS']]]
['<if_alt>', ['PP', 'do', 'solfa_syllable.n.01', 'ATRANS', 'ACT', 'know',
'move', 'MTRANS']]
rule_collector  ('4004', '2')   value
['<if_alt>', [['how', 'AA', 'None', 'PTRANS'], ['make', 'ACT', 'build',
'PTRANS']]]
['<if_alt>', ['ACT', 'make', 'build', 'PTRANS']]
rule_collector  ('4004', '3')   value
['<if_alt>', [['that', 'PP', 'object', 'ATRANS']]]
['<if_alt>', ['PP', 'that', 'object', 'ATRANS', 'ACT', 'drink', 'ingest',
'ATRANS']]
```