Boosting a Chatterbot Understanding with a Weighted Filtered-Popping Network Parser

Javier M. Sastre-Martínez[1,2,3], **Jorge Sastre**[3], **Javier García-Puga**[4]

[1] Institut Gaspard-Monge, Université Paris-Est,
F-77454 Marne-la-Vallée Cedex 2, France

[2] Grup Transducens, Departament de Llenguatges i Sistemes Informàtics, Universitat d'Alacant,
E-03071 Alacant, Spain

[3] Instituto de Telecomunicaciones y Aplicaciones Multimedia, Universitat Politècnica de València,
E-46022 València, Spain

[4] Telefónica I+D,
Calle Emilio Vargas 6, E-28043 Madrid, Spain

Abstract

We describe here an application of the filtered-popping network (FPN) parser in (Sastre, 2009a) for boosting the recognition capabilities of an AIML (Wallace, 2004) chatterbot: the MovistarBot. This conversational agent was developed by Telefónica R&D as an attractive medium for the request of mobile services, such as sending SMSs or downloading games, accessible via short text messages in Spanish through MSN Messenger. AIML being too cumbersome for the fine description of complex sentences, the original chatterbot required services to be requested following a strict command syntax; natural language (NL) requests were answered with the description of the corresponding command syntax, assumed by the presence of keywords. We have manually constructed a recursive transition network (RTN) with output recognizing and tagging a significant variety of requests in Spanish, implemented an automatic RTN weighting procedure for ambiguity resolution and adapted the FPN parser for the automatic translation of the RTN sentences into command requests.

Keywords: AIML, chatterbot, local grammar, recursive transition network, filtered-popping network, parser

1. Introduction

Chatterbots are computer programs designed to simulate intelligent human conversation. Artificial intelligence markup language (AIML) (Wallace, 2004), is an XML-compliant language for the description of a chatterbot's set of conversational rules, where rules are basically pairs input pattern/output answer. Chatterbots are progressively been adopted as virtual assistants for commercial applications. In particular, Telefónica R&D has developed an AIML chatterbot called MovistarBot which can launch a set of mobile services requested through MSN Messenger via short text messages in Spanish. While it is not always necessary to understand what is being said in order to give a human-like answer (e.g.: "tell me more about that"), request sentences need a fine analysis in order to correctly determine the service the user is willing to pay for as well as to correctly extract the specified arguments (e.g.: "Envía el mensaje hola al móvil 555-555-555", which means 'Send the message hello to the mobile 555-555-555', corresponds to the SMS service with arguments `message`="hello" and `phone`="555555555"). AIML is not so appropriate for the exact description of these sentence classes due to their variability, complexity and ambiguity. Rather than that, AIML rules recognizing command-like sentences were defined for launching the services (e.g.: "sms phone_number message_content"), and keyword-based rules were defined for detecting requests in Spanish and showing the right command syntax.

In contrast with AIML, the purpose of local grammars (Gross, 1997) is the exact description of NL sentences; local grammars are recursive transition networks (RTNs) (Woods, 1970) with output combined with powerful linguistic predicates on words (sections 2 & 3), including predicates based on electronic dictionaries (section 4). Local grammars can be reused in the definition of other ones by means of call transitions, allowing for a better grammar structuring. AIML can also reuse other rules by means of the `<srai>` tag (Wallace, 2004); however, there is no clear rule structuring since this tag relaunches the evaluation of the whole set of rules for an input reformulation. AIML does neither provide predicates on tokens, hence rules for each inflected form of each synonym of each word to be recognized must be defined. Several NL processing (NLP) systems have been developed for the construction and application of local grammars, namely Intex (Silberztein, 1993), Unitex (Paumier, 2006) and Outilex (Blanc and Constant, 2006). They represent local grammars as graphs (section 5), visual objects equivalent to RTNs with output but even more intuitive. However, these systems are single-user oriented, Intex is not open-source and Unitex and Outilex use grammar application algorithms that are exponential in the worst case: Unitex a top-down parser and Outilex an Earley-like parser equivalent to that in (Sastre and Forcada, 2009b). While a grammar developer may abort a long lasting parsing process, this is not an option for an application that should transparently analyze several user sentences concurrently.

Using the Unitex grammar editor, we have built a RTN with output that tags Spanish sentences requesting for mobile services in order to determine the requested service and the values of the specified arguments (e.g.: "Envía el mensaje<sms/> <message>hola</message> al <phone>555-555-555</phone>"). We have developed a procedure for automatically assigning weights to grammar transitions, obtaining a weighted RTN (WRTN) that ranks the different interpretations of ambiguous sentences (section 6). We have adapted a preceding implementation of the filtered-popping network (FPN) parser presented in (Sastre, 2009a), an Earley-like parser with output generation but with a polynomial worst-case cost, and reused some of the Unitex source code in order to build a multi-user oriented NLP engine (section 7). As the other systems, the engine features powerful linguistic operators on words (sections 2 & 3), some of them based on electronic dictionaries (section 4), and blank sensitive/insensitive ε-moves (section 8). We have build a corpus of possible user sentences for testing the system (section 9). We give relevant implementation details of the system in section 10, and performance measures in section 11. Concluding remarks close the paper.

2. Tokens

Considering the alphabet of Unicode characters composing a NL text and its disjoint subsets of letters, symbols and blanks, we define a word as a consecutive sequence of letters and a token as either a word or a single symbol. Blanks do not constitute tokens: they are word separators. Symbols do not require to be separated from other tokens since they constitute tokens alone.

3. Lexical Masks

As stated in (van Noord and Gerdemann, 2001), finite-state machines defined on alphabets of predicates on the tokens of a NL (e.g.: any determiner) are more natural than those defined on the NL character alphabet. Calls to subgrammars recognizing sets of tokens could be used instead of predicates; however, defining predicates for commonly used sets of tokens is a better structured approach and more practical than defining the equivalent automata—if they exist—in many cases (e.g. any verb). We have implemented most of the predicates used in the Intex and Unitex systems in order to reuse their grammars, namely:

- universal mask: <TOKEN> represents any token,
- literal masks: match the specified word, respecting the case and diacritic marks if the word is quoted (e.g.: "UN" holds for "UN" but not for "un") and ignoring them if not (e.g.: FIANCÉE holds both for "FIANCÉE" and "fiancee"),[1]
- character class masks: <NB> represents any digit, <PNC> any punctuation symbol, <MOT> any word, <MAJ> any uppercase word, <MIN> any lowercase word and <PRE> any word whose letters are all lowercase except the first one,
- dictionary-based masks: described in section 4.

Negated versions of the two latter categories have also been implemented as for the Intex and Unitex systems.

4. Dictionary Masks and Tools

Dictionary masks match subsets of words of a DELAF dictionary. DELAF dictionaries (Silberztein, 1993) are text files containing an entry per meaning of each inflected form of each word. Each entry (e.g.: envía,enviar.V+Trans_msg:3Ps:2Ys) contains the inflected form (envía=send) and its properties, namely the canonical form (enviar=to send), semantic class identifiers (at least one for the part-of-speech: V=verb & Trans_msg=synonyms of "to send" in the context of sending an SMS) and inflection codes (3Ps=3rd person, present, singular & 2Ys=second person, imperative, singular). The canonical form—for instance the infinitive form in the case of verbs—identifies the set of inflected forms of the word. Semantic classes can be defined in order to ease the grammar construction (e.g.: Trans_msg). Inflection codes depend on the part-of-speech and identify the particular inflected form, for instance by specifying the tense, mood, gender and number for the case of verbs; entries contain several lists of inflected codes when representing several inflected forms (e.g.: 3Ps:2Ys for envía). Words spelled alike but having different meanings are described in separate entries with a different canonical form and/or semantic codes (e.g.: enviado=sent as verb and enviado=correspondent as noun). Dictionary masks have been useful for representing numerous words by means of short codes, for instance <V+Trans_msg:Y2:Y3> represents 40 words that can be used for asking the chatterbot to send an SMS (e.g.: envía, envíe, enviad, envíen, manda, mande, mandad, manden, etc).[2]

We have used the Spanish DELAF dictionary provided with the Unitex system (Blanco, 2000) under the LGPLLR license,[3] which contains more than 600.000 entries. We firstly implemented a trie (Fredkin, 1960) data structure for representing the dictionary. We implemented efficient methods for adding, reading, modifying and removing dictionary entries. We implemented a tool which extracts dictionary subsets of entries matching at least one dictionary mask of a set. This tool allows for testing the dictionary coverage on specific subsets of the lexicon, such as the set of determiners, as well as for examining the content of semantic classes or simply validating dictionary masks. We implemented another tool for the automatic addition/removal of dictionary entries from sets of semantic classes by also specifying a set of dictionary masks; creating a semantic class in the Spanish DELAF such as the synonyms of verb "to send" requires to add a semantic code to 587 entries, which can be done with this tool in a few seconds by specifying a new semantic code and the set of canonical forms of each synonym: +Trans_msg <V.enviar>+<V.mandar>+etc.

[1] Intex and Unitex ignore the case but not the diacritic marks.

[2] In Spanish, we address somebody using the third person instead of the second in order to show respect, courtesy or distance.

[3] The terms of the LGPLLR license can be found at http://igm.univ-mlv.fr/~unitex/lgpllr.html

Due to the dictionary size, loading it in memory as a trie requires several seconds. This becomes time-consuming when having to load it many consecutive times during the system development. We have used the dictionary compression tool included in Unitex, which is based on the partial minimization of finite-state automata (FSA) presented in (Revuz, 1992), and added support for this compressed dictionary representation. The resulting dictionary takes only two megabytes, which are loaded in an imperceptible amount of time. In contrast with the trie format, the compressed format can be read but not modified. However, Unitex does only provide compressed dictionaries but no decompression tool: dictionary source files are to be requested to the Unitex author. We have implemented such a decompression tool and used it for verifying the correctness of our compressed dictionary implementation.

Diacritic-mark and case insensitivity is a must when processing user input. We have developed a tool for the character normalization of every inflected and canonical form inside a dictionary by means of a Unicode character look-up table. The tool also joins together groups of newly ambiguous entries due to the loss of information (e.g.: `envío,enviar,V+Trans_msg:1Ps` and `envió,enviar.V+Trans_msg:3Js` become `envio,enviar:V+Trans_msg:1Ps:1Js`, `J`=preterite).

5. Weighted Graphs

Graphs (see Fig. 1) are an equivalent representation of RTNs with output where RTN transitions are replaced by boxes containing transition input labels and RTN states by unlabeled and undirected edges linking the boxes. Boxes with multiple entries correspond to multiple transitions between the same source and target states. A box may define an output for every input they contain, in which case appears beneath the box. A triangle attached to one side of each box indicates the box direction. Empty boxes (only having the triangle) or box entries with the code `<E>` represent ε-moves (transitions without consumption). Greyed box entries represent call transitions. The edge connected to a single box represents the initial state, and every edge connected to the circle with a square inside represents an acceptor state. This representation was firstly introduced by the Intex system. As done in the Outilex project, we have extended the source code of the Unitex graph editor in order to define multi-line box outputs, the first line for output XML tags and the second for the box weight. Before applying a graph, we transform it with the Unitex system into a FSA-like determinized WRTN with output: since machines with output might not be determinizable, triplets input/output/weight and subgraph calls are regarded as FSA input symbols.

6. Automatic Weight Assignment

We have developed a procedure for the automatic assignment of weights to grammar transitions depending on the specificity of their lexical masks: universal mask transitions are given a null weight, dictionary mask transitions an average weight and literal mask transitions a high weight. Upon ambiguity, the most descriptive interpretation is chosen. In our use case, some of the input segments

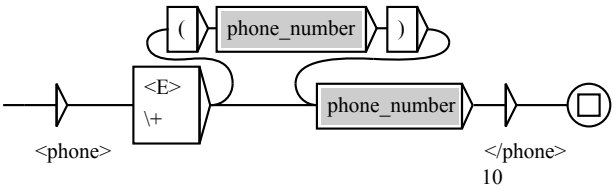

Fig. 1: Weighted graph which matches a phone number with an optional country code (e.g.: (34) 555-555-555), delimits it with XML tags `<phone>` and adds 10 points to the current interpretation; called subgraph `phone_number` recognizes any digit sequence with or without digit separators.

to be extracted have an unknown content (e.g.: the text of the SMS to be sent). These segments are matched by a repeated application of the universal mask, and therefore they don't increase the interpretation score. Ambiguous cases arise when left and/or right contexts of these segments contain optional parts (e.g.: in "Envía el mensaje hola al móvil 555-555-555", "el mensaje" and "al móvil 555555555" may or may not be a part of the text to be sent). We describe these optional contexts with more specific masks so that not recognizing them as a part of the unknown content segment produces higher scores. Since manually defined weights are not overwritten, it is still possible to define transition preferences, though the automatic assignment has given excellent results so far.

7. Engine Workflow

We have developed a NLP engine which either translates user sentences into commands or simplifies them for easing the construction of AIML rules. We describe below each one of its execution steps.

7.1. Initialization

The WRTN and the dictionary are loaded once and then shared by every parsing thread. The trie-string optimization presented in (Sastre and Forcada, 2009b) is applied to XML output tags for fast tag copy and comparison. Unquoted literal masks are normalized as for dictionaries.

7.2. Input Tokenization

Given an input character sequence $\sigma_1 \ldots \sigma_l$, a sequence of input token objects is built, each object composed by a pair of input indexes (i, j) for token $\sigma_{i+1} \ldots \sigma_j$ and an integer identifying the token type (e.g.: lowercase word) for the evaluation of character-class masks. The parser skips blanks by iterating over the token sequence but the original blanks are kept when extracting input segments.

7.3. Grammar Application

The WRTN with output is applied to the token sequence following the algorithm described in (Sastre, 2009a), which results in a weighted FPN (WFPN) recognizing every input translation. Even in cases in which the number of outputs increases exponentially w.r.t. the input length, the algorithm has a polynomial asymptotic cost and linear in many cases thanks to the compressed representation of the set of translations as a WFPN. This WFPN is a

WRTN with a map κ of states to token indexes so that given r and r', the start and end states of a path p, p recognizes a translation of token segment $\kappa(r)+1 \ldots \kappa(r')$, token 1 being the first token. WFPN pops are filtered: a WFPN popping transition (return from a call) from an acceptor state r to a return state r' can be taken iff $\kappa(r) = \kappa(r')$, forbidding translation paths of disconnected input segments.

7.4. WFPN Pruning

Since not every exploration of the WRTN may lead to the recognition of the whole input, the WFPN may contain paths that are not a part of some complete input translation. These paths are removed by pruning the WFPN. In order to efficiently perform this operation, WFPN pop transitions must be explicitly represented as well as the reverse of each transition. Basically, the procedure is as follows:

- if the "global" acceptor state of the WFPN has no incoming pop transitions, remove every state and transition,[4]
- unmark every WFPN state,
- mark the "global" acceptor state an enqueue it,
- while the queue is not empty,
  - dequeue next state r,
  - for each state r' with an outgoing transition to r, pop transitions included but not pushing ones (call initializers), if r' is unmarked then mark it and enqueue it,
- for each WFPN unmarked state r, remove every transition coming to or going from r and then remove r.

Since the WFPN size is at most polynomial w.r.t. the input length, this operation has a polynomial worst-case cost. Unrecognized inputs result in empty WFPNs, in which case the engine proceeds as described in section 7.7.

7.5. WFPN Decompression

The remaining WFPN paths are explored in order to compute the set of weighted translations, which is equivalent to compute the translations of the empty string considering every WFPN transition as an ε-move having its original label as output instead of input. We apply the breadth-first algorithm in (Sastre and Forcada, 2009b) with the following particularities:

- partial outputs of execution states (ESs) are weighted maps of XML tag names to input intervals,
- the initial ES has a zero-weighted empty map,
- popping from an acceptor state r to a return state r' is possible iff $\kappa(r) = \kappa(r')$,
- when computing the set of derived ESs from an ES x, derived ESs are modified copies of x except for the last derived ES which keeps the original x data object,
- when deriving an ES x' from an ES x due to a transition t from a state r,
  - if t has an associated weight, increment x' map weight accordingly,
  - if t generates an XML tag with name n,
    - if it is an opening tag, add the mapping n to (i, j) with i equal to the left bound of token $\kappa(r) + 1$ (or l, the right input bound, if the number of tokens is less than $\kappa(r) + 1$) and j equal to the input end,
    - if it is a closing tag and n is already mapped, redefine j as the right bound of token $\kappa(r)$ (or 0, the left input bound, if $\kappa(r) = 0$),
    - if it is a closing tag and n is not mapped, add the mapping with i equal to the left input bound and j as for the precedent case.

Decompressing a WFPN has an exponential worst-case cost due to sentences having an exponential number of interpretations w.r.t. their length; however, this cost can be dramatically reduced by limiting the number of sentence interpretations represented in the grammar, even if many local ambiguities are represented. Note that for WFPNs representing a single translation, a unique ES data object is created and its partial output incremented as weights and XML tags are found.

7.6. Command Generation

A command is generated for the map with the highest weight that corresponds to a known service, which is determined by the presence of a mapped service tag name (e.g.: `sms`). Expected service argument maps are searched and their corresponding input intervals copied and reformatted, if necessary (e.g.: phone numbers are copied without digit separators). Arguments are optional: the most complete command sentence is generated so that the chatterbot can ask for the missing arguments instead of obligating the user to retype the whole request (e.g. "Quiero enviar un mensaje al 555-555-555" yields `sms_phone 555555555`). The processing of recognized sentences ends here.

7.7. Infinitive Form Translation

Unrecognized sentences are returned preceded by command `unknown` and with every word that might be a verb replaced by its infinitive form, retrieved from the dictionary. AIML rules can then consider infinitive forms only.

8. Sensitivity to Blanks

The presence of blanks between tokens is restricted in some cases, for instance command arguments usually cannot contain blanks but argument lists must be blank-separated. Since blanks are not tokens, consuming a token implicitly consumes the blanks that precede it. Analogously to Intex and Unitex, we have implemented blank-forbidden and mandatory ε-moves: the former can be taken iff the precedent and next token input intervals are connected, and the latter iff they are not. For instance, we connect two grammar structures recognizing two consecutive command arguments with a blank-mandatory ε-move in order to avoid the arguments from being attached.

[4] By construction, every WFPN path representing a complete input translation ends by popping this "global" acceptor state.

9. Testing Corpus

We have built a corpus of 334 possible sentences, most of them service requests but also other sentences in order to control over-recognition. Service requests are formed by a few compounds that may permute (e.g. "Envía el mensaje hola al móvil 555-555-555", "Envía al móvil 555-555-555 el mensaje hola"), each compound having a finite variability (e.g. "al móvil 555-555-555", "al 555555555", etc). The corpus considers every possible permutation, using different compound variants in each one instead of considering every possible combination, thus the corpus is representative in spite of its size. We have aligned the corpus with the answers the system should return and implemented a tool for verifying the answer returned by the system for every sentence in the corpus.

10. Implementation Details

The NLP engine and tools have been programmed in C++ using the standard template library (STL). We have implemented a multi-platform compilation script which generates executable files for local use and testing of the engine as well as a shared library for its online use: messages sent to the chatterbot are first sent to a Tomcat servlet as a UTF-8 stream through the Internet, then the servlet uses the library through the Java Native Interface in order to process the message and finally the answer is sent to the chatterbot. This architecture allows for independence and easy integration of the NLP engine and the chatterbot since the engine is completely transparent to the chatterbot and they can be run in separate machines. The engine has successfully been tested in both Windows and GNU/Linux platforms and both big- and little-endian architectures.

11. System Performance

At a first stage the engine did not support weighted grammars and the grammar was developed trying to consider a unique interpretation of each sentence: the one a human would choose in the context of providing mobile services. Local ambiguities could still be present since they would be solved at some point of the analysis, resulting in a linear FPN and therefore avoiding the potentially exponential cost of FPN decompression. In case of global ambiguity, the first interpretation found would be returned. With a grammar supporting the SMS service (around 500 states and 1300 transitions), the engine was able to preprocess around 2000 sentences per second on a GNU/Linux Debian platform with a 2.0 GHz Pentium IV Centrino processor and a 2 GB RAM. However, extending the grammar for more complex services was not feasible due to the increasing ambiguity. We added weighted grammar support for interpretation discrimination and extended the grammar in order to support queries on a catalog of games, songs and images as well as download requests, queries on a notification system and notification subscriptions (e.g.: notifications on soccer events), and other queries on other topics such as available offers, prepay cards, video calls, etc. Thanks to the reuse of formerly defined grammar structures, the number of states and transitions was only tripled but the number of preprocessed sentences per second decreased by one-tenth. Though 200 sentences per second is still acceptable, we expect that for a considerable amount of services the system will be too slow. We expect to regain the efficiency by means of an algorithm able to extract the highest ranked interpretation from the WFPN without having to decompress the entire WFPN.

12. Conclusion

We have proved the viability of the FPN parser presented in (Sastre, 2009a) in order to serve as a significant refinement of an AIML chatterbot, enabling the chatterbot to extract key data from complex and ambiguous sentences for launching a set of commercial services. We have adapted a former implementation of this parser, developed a grammar recognizing a set of sentences requesting for mobile services and automatically assigned weights to the grammar transitions for ambiguity resolution. The whole system has been used as a message preprocessor in order to boost the recognition capabilities of the MovistarBot, an AIML chatterbot developed by Telefónica R&D, as well as to simplify the construction of its AIML rules.

Acknowledgements: We thank the reviewers and Profs. E. Laporte and M. L. Forcada for their useful comments.

13. References

Blanc, O. and Constant, M. (2006). *Outilex, a linguistic platform for Text Processing*. In Interactive Presentation Session of Coling-ACL06. Morristown, NJ, USA: Association for Computational Linguistics.

Blanco, X. (2000). Les dictionnaires électroniques de l'espagnol (DELASs et DELACs). *Lingvisticæ Investigationes*, 23(2):201–218.

Fredkin, E. (1960). Trie memory. *Communications of the ACM*, 3(9):490–499.

Gross, M. (1997). The construction of local grammars. In Emmanuel Roche and Yves Shabes (eds.), *Finite-State Language Processing*. Cambridge, MA, USA: MIT Press, pages 329–352.

Paumier, S. (2006). *Unitex 1.2 User Manual*. Université de Marne-la-Vallée. `http://www-igm.univ-mlv.fr/~unitex/UnitexManual.pdf`.

Revuz, D. (1992). Minimisation of acyclic deterministic automata in linear time. *Theoretical Computer Science*, 92(1):181–189.

Sastre, J. M. (2009a). Efficient parsing using filtered-popping recursive transition networks. *Lecture Notes in Computer Science*, 5642:241–244.

Sastre, J. M. and Forcada, M. L. (2009b). Efficient parsing using recursive transition networks with output. *Lecture Notes in Artificial Intelligence*, 5603:192–204.

Silberztein, M. D. (1993). *Dictionnaires électroniques et analyse automatique de textes. Le système INTEX*. Paris: Masson.

van Noord, G. and Gerdemann, D. (2001). Finite-state transducers with predicates and identities. *Grammars*, 4(3):263–286.

Wallace, R. (2004). *The elements of AIML style*. ALICE AI Foundation. `http://www.alicebot.org`.

Woods, W. A. (1970). Transition network grammars for natural language analysis. *Communications of the ACM*, 13(10):591–606.