Automatic evaluation of punning riddle template extraction

Try Agustini and Ruli Manurung
Faculty of Computer Science
Universitas Indonesia
Depok 16424, Indonesia
try.agustini@gmail.com and maruli@cs.ui.ac.id

Abstract

This paper reports an empirical study to automatically evaluate the ability of T-PEG (Hong and Ong 2009) to extract joke templates by providing it with a corpus of punning riddles produced by another system, STANDUP (Manurung et al. 2008). This setup allows us to compare the extracted templates against the underlying data structures used by STANDUP in generating the corpus. In our setup, T-PEG is modified with a generalization component that clusters extracted templates based on structural similarity. These clusters are then compared against the underlying rules used by STANDUP to measure how well T-PEG is able to induce the schema used by STANDUP to generate the jokes. Whilst far from conclusive, an overall precision of 0.61 and recall of 0.763 suggests that T-PEG is able to extract some salient information regarding the underlying lexical relationships found within a punning riddle.

Background

The automatic construction of jokes, specifically punning riddles, as artefacts of linguistic creativity, has received quite a bit of attention in recent years. See (Ritchie 2005) for a good overview. Compared to other forms of linguistic creativity, such as stories and poems, punning riddles obey more formal structures, and hence are more amenable to automated construction.

Most of the existing systems such as JAPE (Binsted 1996) and STANDUP (Manurung et al. 2008) work from a predefined set of rules often called schemata, which forms the foundation of a joke, and which arguably encodes the humorous knowledge which makes the resulting text recognizably a joke. The schemata are typically handcrafted by the researchers based on analysis and observation of collections of exemplar jokes such as punning riddles.

T-PEG (Hong and Ong 2009) is a system which works along similar lines, but with a crucial difference: the schemata, or in T-PEG terms the 'templates', are automatically extracted from a corpus of exemplar punning riddles. Although (Hong and Ong 2009) report a manual evaluation of the extracted templates, it is subjective in nature and limited in scope. In this paper we present our work in automatically evaluating the template extraction functionality of T-PEG by providing it with example jokes produced by STANDUP, from which we can carry out a more extensive empirical study due to the fact that we have access to the underlying data structures used by STANDUP in generating the samples.

The rest of the paper is structured as follows. We first provide a brief overview of automatic punning riddle generation, describing the mechanisms of STANDUP and T-PEG, before discussing our proposed setup of how to carry out an automatic evaluation of the extracted templates. We then present the results and discussion thereof.

Punning riddle generation

STANDUP

Riddle generation in systems such as JAPE and STANDUP consists of several stages, where at each stage a particular kind of rule is selected and instantiated, i.e. schemas, description rules, and templates. Instantiation is performed by matching against a lexical database. In STANDUP, various lexical resources are utilized, among others Unisyn[1] for phonetic information and WordNet (Fellbaum 1998) for lexical semantic information.

A schema is composed of a header, which specifies a symbolic label and its input parameters, a set of lexical preconditions that specify phonetic, syntactic, and semantic relations that must hold between key lexical items, and a question and answer specification, which determines the output for the next stage (Manurung et al. 2008).

For example, the `newelan2` schema is defined as follows:

- Header: `newelan2(NP, A, B, HomB)`
- Lexical preconditions: `nouncompound(NP,A,B)`, `homophone(B,HomB)`, `noun(HomB)`
- Question specification: `shareproperties(NP, HomB)`
- Answer specification: `phrase(A,HomB)`

It states that a punning riddle can be generated if a set of four lexical items, i.e. `NP`, `A`, `B`, and `HomB`, can be found within a lexical database such that `A` and `B` form the compound noun `NP`, `B` is a homophone of `HomB`, and `HomB` is

[1] `http://www.cstr.ed.ac.uk/projects/unisyn`

a noun. For example[2], an instantiation in which NP = *computer screen*, A = *computer*, B = *screen*, and HomB = *scream*[3] could give rise (after the two further stages) to a riddle such as *"What do you call a shout with a window? A computer scream."*

Once a schema has been instantiated as above, the question and answer specifications are non-deterministically matched against a set of description rules, which encode linguistic variations that are warranted given the schema instantiation. These rules have a structure similar to schemas, in that they have a header, some preconditions, and an output expression called the template specifier. For example, one description rule is:

- Header: `shareproperties(X,Y)`
- Preconditions: `meronym(X, MerX), synonym(Y, SynY)`
- Template specifier: `merHyp(MerX, SynY)`

In the example joke given above, the question specification `shareproperties(computer screen, scream)` would match the header for this rule, where the values (`computer screen`, `scream`) would be bound to the local variables `X,Y` of the rule. Subsequently, the preconditions check further lexical relations to determine whether the rule is applicable. It may also instantiate additional variables, e.g. `MerX` and `SynY`, where in the example above `MerX` is bound to `window`, a meronym of `computer screen`, and `SynY` is bound to `shout`, a synonym of `scream`.

The answer specification `phrase(A,HomB)` trivially concatenates the instantiations, e.g. *computer scream*.

Finally, all these instantiations are passed on to the template-filling stage, where the template specifier `merHyp(MerX, SynY)` is non-deterministically matched against a set of canned text such as *"What do you call a * with a * ?"* where * indicates a slot to be filled with the instantiated words.

T-PEG

In the preceding section, we can see that the role of schemas, description rules, and templates is crucial in defining the humorous effect of the resulting riddle. In both JAPE and STANDUP, these rules were manually handcrafted. For instance, STANDUP has 11 schemas. (Hong and Ong 2009) present T-PEG (Template-based Pun Extractor and Generator), a system that generates punning riddles in a manner similar to JAPE and STANDUP, i.e. working from a set of symbolic rules that define a punning riddle, it instantiates certain key variables by selecting appropriate sets of words from a combination of lexical resources. The crucial difference is that whereas the symbolic rules used by JAPE and STANDUP were manually crafted, T-PEG relies on templates that are automatically extracted from a collection of exemplar jokes.

[2]The following worked example is taken from (Manurung et al. 2008).

[3]Homophony can be generalized to include pairs of words whose phonetic similarity is above a certain threshold.

Given a sample punning riddle, T-PEG constructs a template by firstly identifying nouns, verbs, and adjectives with the help of a part-of-speech tagger, and replacing them with what they term *regular variables*. Additionally, *similar-sound variables* are identified as words that are homophones of regular variables. Regular variables follow a naming convention where Xn indicates the n-th word in the question part and Yn indicates the n-th word in the answer part. Similar-sound variables are indicated by appending a dash and a number. All pairs of variables are then checked against a lexical resource to detect whether any semantic relations hold between them. In T-PEG, the lexical resources used are Unisyn, WordNet, and ConceptNet (Liu and Singh 2004).

For example, given the joke "How is a window like a headache? They are both panes", T-PEG can extract the template "How is a <X3> like a <X6>? They are both <Y3>", where the following word relations must hold:

- `Y3-0 SoundsLike Y3`
- `X3 ConceptuallyRelatedTo Y3`
- `Y3 ConceptuallyRelatedTo X3`
- `Y3 PartOf X3`
- `X6 ConceptallyRelatedTo Y3-0`
- `X6 IsA Y3-0`
- `Y3-0 ConceptuallyRelatedTo X6`

From the example above we can see that the notion of a template in T-PEG is equivalent to the conflation of a schema, description rule, and template in STANDUP.

The constructed templates are then filtered based on a graph-connectedness heuristic, i.e. if the variables are nodes and the word relationships are edges, a template must form a connected graph to be deemed a valid template.

Once the template has been extracted, generation proceeds in a similar fashion to STANDUP. In this paper we are less interested in the generation aspect of T-PEG, as it is largely similar to JAPE and STANDUP, and more interested in its ability to automatically learn or extract templates. Specifically, we are interested in assessing the ability of T-PEG to correctly identify the necessary and sufficient conditions for generating punning riddles. (Hong and Ong 2009) report a manual evaluation where a subset of the extracted templates was manually assessed by a linguist, whose job was to determine if the extracted templates were able to capture the 'essential word relationships in a pun'. The evaluation criteria are based on the number of incorrect relationships as identified by the linguist, and includes missing relationship, extra relationship, or incorrect word pairing. A scoring system from 1 to 5 is used, where 5 means there is no incorrect relationship, 4 means there is one incorrect relationship, and so on. They report an average score of 4.0 out of a maximum 5.

Automatic evaluation of template extraction

There are two issues concerning the manual evaluation of template extraction presented in (Hong and Ong 2009).

Firstly, we believe this evaluation is rather subjective. Although punning riddles are relatively simple and straightforward to analyse, the linguists were not the original authors of the jokes, and thus there is room for misinterpretation or incorrect emphasis. Furthermore, it is unquestionable that relying on the manual judgment of a linguist is both time-consuming and costly. The manual evaluation reported in (Hong and Ong 2009) was carried out on 27 templates generated from 27 jokes, which is a rather small sample from which to draw any conclusion.

Our observation is that if one had access to a large corpus of punning riddles that had somehow been annotated with the 'correct' word relationships that underlie the joke, one could assess T-PEG's template extraction functionality by comparing the resulting template against the reference word relationships. Unfortunately, we know of no such annotated resource that currently exists. However, we can use an existing punning riddle generator such as STANDUP to produce an approximation of such a resource, since we can access the underlying data structure of a generated punning riddle. In the joke generation module of STANDUP, the `JokeGraph` object of a generated punning riddle provides full access to the underlying lexical relationships.

Another issue we attempt to address is the fact that T-PEG makes no attempt at generalization of the extracted templates. Given fifty exemplar punning riddles, it will attempt to construct fifty templates. Hong and Ong state that this is beneficial 'to increase coverage'. However, we contend that if we are interested in building systems that computationally model the mechanisms of linguistic humor, coverage is not enough. A creative generative system should be able to generate artefacts from a limited set of symbolic rules. Thus, T-PEG should be able carry out some abstraction over the extracted templates, to yield a set of highly-productive patterns.

These two goals form the rationale of our proposed setup, which we discuss in the next section.

Proposed setup

As discussed above, the purpose of our experiment is to automatically evaluate the ability of T-PEG to correctly extract templates that underlie a collection of punning riddles. The proposed setup is as follows. Firstly, STANDUP is used to generate a large number of punning riddles. For each riddle, we note the actual rules used by STANDUP to generate them, which are used during the evaluation phase. The riddles are then given to T-PEG, which yields a template for each riddle. These templates are then organized into clusters using *agglomerative clustering* by calculating the similarity between templates using a structural similarity metric based on the semantic similarity evaluation function presented in (Manurung, Ritchie, and Thompson 2012). We then apply a simple majority rule to label the clusters, and then evaluate the template extraction process using the widely-used notions of precision and recall.

To achieve this, T-PEG first had to be modified by replacing its lexical and conceptual resources with those that were used in STANDUP, thus ensuring that the template extraction module would be able to identify the original lexical relationships in STANDUP.

Template clustering

The agglomerative clustering process starts with all templates belonging to singleton clusters. The distance of each cluster to all other clusters is then computed. The distance between two clusters is defined as the average distance of each pair of elements contained within the two clusters, also known as *average linkage clustering*. The two clusters with the shortest distance are then merged together. This process is repeated until k clusters remain, where k is provided as an input parameter.

In defining the notion of distance between two templates, we turn to the structure-mapping work of (Love 2000) and (Falkenhainer, Forbus, and Gentner 1989) that has defined a computational model of semantic similarity in terms of conceptual and structural similarity. Structural similarity measures the degree of isomorphism between two complex expressions. Conceptual similarity is a measure of relatedness between two concepts.

More specifically, we use the semantic similarity evaluation function used in (Manurung, Ritchie, and Thompson 2012), which implements a greedy algorithm based on Gentner's structure mapping theory (Falkenhainer, Forbus, and Gentner 1989). It takes two complex expressions, in our case two T-PEG extracted templates, and attempts to 'align' them in an optimal manner. It then applies a scoring function based on Love's computational model of similarity (Love 2000) to compute a score based on various aspects of the alignment. This function yields a distance of zero between two conceptually and structurally identical templates, and further distances for increasingly different template pairs.

Cluster labeling

We then automatically label the clusters using a simple majority rule. First, we define the *underlying schema* of a template to be the label of the STANDUP schema that was used to generate the punning riddle from which the template was extracted.

A cluster is then automatically labelled by identifying the underlying schemas of all its member templates, and simply choosing the schema that created the majority of templates within that cluster. If there are several underlying schemas that produced the same number of templates in a cluster, one is randomly selected. As an example, if a cluster contains 10 templates whose underlying schema is *lotus*, and 6 templates whose underlying schema is *newelan1*, then that cluster is labelled as representing the *lotus* schema.

Precision and recall

Using these cluster labels, we can compute measures which correspond to the widely-used notions of *precision* and *recall* in pattern recognition. In classification tasks, these measures are defined as follows:

$$Precision = \frac{tp}{tp + fp} \qquad Recall = \frac{tp}{tp + fn}$$

where in our case, given a cluster c with label l, tp (true positive) is the number of extracted templates that appear as members of c whose underlying schemas are l, fp (false positive) is the number of templates in c whose underlying schemas are not l, and fn (false negative) is the number of templates not in c but whose underlying schemas are l.

Precision and recall can be computed for each cluster, or as an aggregate measure over all resulting clusters.

Experimental setup

The experimental setup is as follows. Firstly, STANDUP is used to generate 20 jokes for each of 10 schemas, namely `bazaar`, `lotus`, `doublepun`, `gingernutpun`, `rhyminglotus`, `newelan1`, `newelan2`, `phonsub`, `poscomp`, and `negcomp`, resulting in a collection of 200 exemplar jokes. These jokes are then analysed by T-PEG, which yields 200 joke templates.

We then apply agglomerative clustering until 10 clusters are formed (since 10 STANDUP schemas are used). Our hypothesis is that for T-PEG to be deemed successful in extracting templates, it should be able to correctly organize the 200 templates into 10 clusters that correspond to the 10 STANDUP schemas. The precision and recall metrics should provide appropriate quantitative measures as to this goal.

Results and discussion

Table 1 shows the results of applying agglomerative clustering on the 200 templates into 10 clusters. The first column indicates the cluster number. The second and third columns specify the cluster membership, by indicating the number of templates with a given underlying STANDUP schema found within that cluster. The fourth column indicates the cluster size. The last column indicates the label assigned to that cluster using the majority rule described above. For example, cluster 1 contains 21 templates, 19 of which have `rhyminglotus` as their underlying schema, and 2 of which have `bazaar` as their underlying schema. Accordingly, it is assigned the label `rhyminglotus`.

Table 2 shows the precision and recall values computed for the clustering results. The first two columns indicate the cluster numbers and assigned labels, which correspond to the information in Table 1, and the last two columns indicate the precision and recall values computed for each cluster. Finally, the last row indicates the overall precision and recall. This aggregate result take into account the weighted averages given the different cluster sizes. Note that we collapsed clusters 1 and 6 because they were both labeled as `rhyminglotus`, and similarly, clusters 2 and 3 are collapsed due to the fact that they are both labeled as `bazaar`.

Finally, Table 3 shows the confusion matrix of how the templates are classified according to their underlying schema. The rows indicate the original underlying schemas, and the columns indicate the cluster labels. For example, of the 20 templates extracted from punning riddles generated using the `bazaar` schema, 14 are correctly found within a cluster labeled `bazaar`, 2 are found in a cluster labeled `rhyminglotus`, and 4 are found in a cluster labeled `lotus`.

No.	Schema	Amount	Total	Label
1	rhyminglotus bazaar	19 2	21	rhyminglotus
2	bazaar	11	11	bazaar
3	bazaar	3	3	bazaar
4	lotus newelan1 gingernutpun newelan2 doublepun bazaar	20 19 17 16 13 4	89	lotus
5	doublepun newelan2 gingernutpun	7 4 3	14	doublepun
6	rhyminglotus	1	1	rhyminglotus
7	newelan1	1	1	newelan1
8	phonsub	20	20	phonsub
9	poscomp	20	20	poscomp
10	negcomp	20	20	negcomp

Table 1: Results of agglomerative clustering

No.	Label	Precision	Recall
1 & 6	rhyminglotus	20/22=0.91	20/20=1
2 & 3	bazaar	14/14=1	14/20=0.7
4	lotus	20/89=0.225	20/20=1
5	doublepun	7/14=0.5	7/20=0.35
7	newelan1	1/1=1	1/20=0.05
8	phonsub	20/20=1	20/20=1
9	poscomp	20/20=1	20/20=1
10	negcomp	20/20=1	20/20=1
	Overall	0.61	0.763

Table 2: Precision and recall measures

From these results, we can see that only templates with `phonsub`, `poscomp`, and `negcomp` as their underlying schemas are perfectly identified. Templates with the underlying schemas `rhyminglotus` and `lotus` are correctly clustered together, but suffer some impurities with other templates also being deemed to belong to their clusters. Most notably, the cluster labeled `lotus` contains a very large number of templates from other schemas such as `bazaar`, `doublepun`, `gingernutpun`, `newelan1`, and `newelan2`. The cluster labeled `newelan1` contains only one template. No clusters were labeled as `gingernutpun` and `newelan2`.

A purely random baseline, in which the 200 extracted templates are randomly assigned to 10 different clusters, would yield an expected precision and recall of 0.1. Whilst this is an artificially low baseline, the results of an overall precision of 0.61 and recall of 0.763 suggests that T-PEG is able to extract some salient information regarding the underlying lexical relationships of a pun. However, certain underlying schemas are very difficult to distinguish from each other.

Upon further analysis, we can see that the problems arise from the fact that the templates extracted by T-PEG conflate

	bazaar	rhyminglotus	lotus	doublepun	gingernutpun	newelan1	newelan2	phonsub	poscomp	negcomp
bazaar	14	2	4							
rhyminglotus		20								
lotus			20							
doublepun			13	7						
gingernutpun			17	3						
newelan1			19			1				
newelan2			16	4						
phonsub								20		
poscomp									20	
negcomp										20

Table 3: Confusion matrix of clustering results

the various rules in STANDUP, i.e. schemas, description rules, and canned text templates. To illustrate the issues, observe the following two jokes, both generated using the `lotus` schema, and their resulting extracted templates:

Joke 1:

What do you call a cross between a firearm and a first step?
A piece initiative

The resulting template is:

What do you call a cross between a <X8> and a <X11> <X12>?
A <Y1> <Y2>

with the following word relationships:

- IsCompoundNoun(X11, X12)
- IsCompoundNoun(Y1-0, Y2)
- Synonym(X8, Y1)
- Synonym(Y2, X11:X12)
- Hypernym(X11:X12, Y1-0:Y2)
- Homophone(Y1-0, Y1)

From the joke we can see that the instantiations for the regular variables are X8=firearm, X11=first, X12=step, Y1=piece, and Y2=initiative. Furthermore, the similar-sound variable Y1-0 is bound to peace, because in WordNet, "peace initiative" is an instance of an initiative.

Thus, the word relationships state that "first step" and "peace initiative" are compound nouns, firearm is a synonym of piece, initiative is a synonym of "first step", which in turn is a hypernym of "peace initiative", and that lastly, peace and piece are homophones.

Joke 2:

What do you call a cross between an l and a correspondent?
A litre writer.

The resulting template is:

What do you call a cross between an <X8> and a <X11>?
A <Y1> <Y2>

with the following word relationships:

- IsCompoundNoun(Y1-0, Y2)
- Synonym(X8, Y1)
- Synonym(X11, Y1-0:Y2)
- Homophone(Y1-0, Y1)

From the joke, we can see that the instantiations for the regular variables are X8=l, X11=correspondent, Y1=litre, and Y2=writer. Furthermore, the similar-sound variable Y1-0 is bound to letter, because in WordNet, "letter writer" is a synonym for "correspondent".

Thus, the word relationships state that "letter writer" is a compound noun, l is a synonym of litre, correspondent is a synonym of letter writer, and litre and letter are deemed to be homophones.

From these two jokes and their resulting extracted templates, we can make several observations. Firstly, T-PEG correctly extracts the core lexical preconditions stated for the `lotus` schema, in that the 'punchline' must contain a compound noun Y1-0:Y2 ("peace initiative" and "letter writer", respectively), where the first word is replaced with a homophone, Y1 (in the first joke, piece, and in the second, litre).

However, the two jokes use different description rules for the question part. Whereas the former joke used a synonym (firearm) and a hypernym (first step) to describe the punchline, the latter used two synonyms, namely l and correspondent. Since T-PEG makes no distinction between word relationships arising from schemas or description rules, the choice of description rule, which is a somewhat trivial linguistic variation, leads the agglomerative clustering to falsely conclude that two jokes from different underlying schemas in fact use the same pattern.

Additionally, T-PEG extracted 'noisy' word relationships that play no part in the joke construction. Whereas the word relationships of the extracted template for the second joke capture exactly the necessary and sufficient conditions, in the former joke, nothing hinges on the fact that a "first step" is a compound noun, nor that initiative is a synonym of "first step". Such extraneous word relationships further pull the templates into wrong clusters.

From this experiment, we can conclude that although T-PEG may be successful at learning some joke templates given sample punning riddles, it is still making faulty assertions as to the constraints that specify what makes the riddle 'work' as a joke.

However, we speculate that much work can be done to repair such errors. The data redundancy contained within the specific templates that are clustered together, for instance, can be further analysed to form core relationships that are at the heart of the punning riddle structure. This is an avenue of future work that we intend to explore.

Summary

The manually constructed rules of STANDUP are specifically designed to maximize generative powers whilst retaining a fairly limited set of symbolic rules. T-PEG, on the other hand, tries to address the issue of having to handcraft rules by automatically extracting templates from example jokes. However, the evaluation of this functionality was

fairly limited given the difficulty and cost of manual evaluation. This paper has attempted to carry out a fairly novel methodology of automatically evaluating the performance of one creative system (T-PEG) using another creative system (STANDUP) to produce sample data with complete underlying annotations for comparing against. Although the results are far from conclusive, it corroborates the results of the manual evaluation in (Hong and Ong 2009) that claims T-PEG was successful in extracting templates from sample jokes. Furthermore, the experiment could shed more light on where T-PEG was still lacking in its ability to extract the underlying generative rules of the punning riddles.

Furthermore, the benefit is twofold: the template clustering process proposed in this work has attempted to address the generalizability of T-PEG. It is not sufficient to say that a huge number of templates will ensure maximum coverage. For a generative system to be deemed creative, it should be able to generate a high ratio of good quality artefacts from a limited set of symbolic rules.

By breaking down the patterns into schemas, description rules, and templates, and by stating the conditions when they can be composed together, STANDUP is able to produce a very wide range of different jokes, and in some sense can "explain" the craftsmanship behind its joke production facilities, as each individual component represents a specific function of the joke. T-PEG's templates, on the other hand, are monolithic structures that cannot be broken down into its functional components. This distinction is to be expected, given that the rules found within STANDUP were manually constructed, whereas T-PEG rules are automatically extracted. Nevertheless, this points towards a promising direction of future work in the automatic extraction of rules from creative artefacts.

As stated in the previous section, we believe that the template clustering process opens up the possibility of future work, i.e. by further leveraging the data redundancy contained with the resulting clusters. Further still, given that the clustering process makes use of the notion of distances between templates and clusters, one could imagine a technique that selects the template closest to the centroid of the cluster as being the most representative template for further generation. Finally, we would like to explore more sophisticated generalization techniques that would enable us to tease out the distinctions between component rules such as schemas, description rules, and canned text templates.

Acknowledgments

We would like to thank the creators of the STANDUP and T-PEG systems for making their software available for further experimentation and development. We would also like to thank the anonymous reviewers for their feedback and suggestions.

References

Binsted, K. 1996. *Machine Humour: An Implemented Model of Puns*. Ph.D. Dissertation, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, UK.

Falkenhainer, B.; Forbus, K. D.; and Gentner, D. 1989. The structure-mapping engine: Algorithm and examples. *Artificial Intelligence* 41:1–63.

Fellbaum, C., ed. 1998. *WordNet: An Electronic Lexical Database*. MIT Press.

Hong, B. A., and Ong, E. 2009. Automatically extracting word relationships as templates for pun generation. In *Proceedings of the 1st Workshop on Computational Approaches to Linguistic Creativity (CALC-09)*.

Liu, H., and Singh, P. 2004. Conceptnet: A practical commonsense reasoning toolkit. *BT Technology Journal* 22(4):221–226.

Love, B. C. 2000. A computational level theory of similarity. In *Proceedings of the 22nd Annual Meeting of the Cognitive Science Society*, 316–321.

Manurung, R.; Ritchie, G.; Pain, H.; Waller, A.; O'Mara, D.; and Black, R. 2008. The construction of a pun generator for language skills development. *Applied Artificial Intelligence* 22(9):841–869.

Manurung, R.; Ritchie, G.; and Thompson, H. 2012. Using genetic algorithms to create meaningful poetic text. *Journal of Experimental & Theoretical Artificial Intelligence* 24(1):43–64.

Ritchie, G. 2005. Computational mechanisms for pun generation. In *Proceedings of the 10th European Natural Language Generation Workshop*, 125–132.