Computational Mechanisms for Pun Generation

Graeme Ritchie
Department of Computing Science
University of Aberdeen
Aberdeen AB24 3UE
Scotland
gritchie@csd.abdn.ac.uk

Abstract

Computer pun-generators have so far relied on arbitrary semantic content, not linked to the immediate context. The mechanisms used, although tractable, may be of limited applicability. Integrating puns into normal text may involve complex search.

1 Introduction

Recently, there has been a growing interest in *computational humour*, as indicated by two international workshops [Hulstijn and Nijholt, 1996; Stock *et al.*, 2002]. A number of computer programs (mostly quite small) have been constructed which generated very short humorous texts (see Section 4 below). All but one of these programs generate some form of *pun*, where we take a pun, informally and pre-theoretically, to be a supposedly humorous written or spoken text which relies crucially on *phonetic similarity* for its humorous effect. There is no accepted strict definition of a pun, even amongst humour scholars, but the computer-generated examples are almost certainly puns by any reasonable definition; whether they are funny or not is a separate question.

The purpose of this paper is to consider the task of pun generation from the wider perspective of NLG, particularly applied NLG, and to discuss the following:

- how the mechanisms used in pun generation systems compare with conventional NLG;
- two classes of pun, with different potential roles in NLG;
- the computations that might achieve such puns;
- possible limitations of these computations.

We shall start with the claims made for the usefulness of computer-generated jokes, give a very brief summary of past work and then present some observations and arguments.

2 Motivation

It could be argued that computer modelling of humour is worthwhile because it might shed light on human use of humour, and hence could contribute to a cognitive model of humour. Here we shall leave that aside, and consider a case which has been more explicitly argued: that certain practical computer systems will be more effective, or more pleasant to use, if they display humour.

It has been claimed for some time that humour enhances communication in various ways. In one study, subjects were persuaded more effectively by material including humour [Lyttle, 2001]. In another, human subjects gave more favourable reports of working with computer systems which employed humour (albeit pre-coded, rather than computer-generated) [Morkes *et al.*, 1999].

Binsted [1995] argues that a user-interface which used humour would be more congenial to interact with. Stock [Stock, 2002; 2003] suggests that computerised humour will have even wider applicability, in advertising, entertainment and education. Nijholt [2002] points out that if virtual agents are to show rich "personalities" in their interactions with humans, some form of humour is essential. McKay [2002] suggests the use of automated humour in a system for second-language learning, and O'Mara and Waller [2003] propose that machine-assisted communication by those with language disabilities (particularly children) could be helped by some software support for humour.

So far, no computer system for humour-generation has yet been shown to have these benefits. Nevertheless, for the purposes of this paper we shall assume, from the writings cited above, that a case can be made for the desirability of computer-generated humour in practical applications.

These authors have argued generally for the practical use of *humour*, not merely *puns*. However, the type of humour-generation that is likely to be available in the near future is pun-generation. In the following sections, therefore, we shall focus solely on puns, considering the ways in which they might be generated within a broader NLG system.

3 Two classes of puns

There are various ways in which puns, or jokes in general, could be classified, depending on the aspects which are of interest. For the discussion here, we wish to make a distinction between two loose groupings (not usually distinguished in the literature):

Self-contained puns: These are pieces of text which can be used as humorous items in a wide variety of circumstances. Any semantic links which they might have to the context are not directly part of the joke structure, and their only preconditions for use are general knowledge (of the surrounding culture, etc.) and a social situation

in which joke-making is acceptable. Example (1) (from [Binsted, 1996]) is a self-contained pun, as are all of the puns ((5) – (13)) shown in Section 4.1 below.

(1) What do you get when you cross a murderer with a breakfast food? A cereal killer.

Contextually integrated puns: This type of pun occurs within some broader discourse, with the use of a text which, in addition to conveying some information or emotion, has further linguistic properties which make it a pun (and may thereby augment the effects of the text, either emotionally, persuasively or otherwise). Also, the status of the text as a pun may depend on contextual (possibly non-linguistic) factors. Ritchie [2004, p.115] offers (2) and (3) as puns; both of these, when first delivered, were contextually integrated puns.

(2) A shopper is walking along, and a leek falls from his shopping bag to the ground, unnoticed. Another shopper calls out, "Hey! Your bag's leaking!"

(3) A minor football team known informally as 'Caley Thistle' (where *Caley* rhymes with *alley*) soundly defeats Celtic (then the top team in the country) in a major competition. The next day, a newspaper headline reads: "Super Caley Go Ballistic, Celtic Are Atrocious". (*The Sun*, 9 February 2000)

In (2), the pun status depends upon the substring *leak* being phonetically similar (identical) to the word *leek*, and the latter word being directly related to the surrounding context. In (3) the whole headline has some phonetic similarity to the song-title *Supercalifragilisticexpialidocious*. In both cases, the utterance conveys contextually appropriate information.

For both these types of puns, the text's status as a pun is *in addition to* a full set of normal linguistic properties: the punning texts are usually syntactically well formed texts which have an internally consistent semantics.

The difference between the two classes can be viewed as follows: in self-contained puns the semantic content is arbitrary, and of secondary importance to the features that make the text a pun, whereas in contextually integrated puns the semantic message is non-arbitrary, and the pun features have to be compatible with it.

There are some puns which might seem to be borderline examples. In everyday social situations, someone may make a remark simply in order to make a pun, which might seem therefore to count as a self-contained pun, but such an utterance is very rarely completely unconnected to the context, even if its sole purpose is humour. For example, someone who moves their position in a room to avoid cold air from an open window might make the remark in (4), punning on the USA expression *draft-dodger*, 'someone who avoids military conscription'.

(4) I'm just a draught-dodger.

Although this remark may communicate little useful information, and is made solely as an attempt at humour, it is still a contextually integrated pun, in the sense that what it conveys relates to the context, and it would not be a pun without this connection – if the speaker had *not* moved to avoid cold air, it would not be felicitous, and if someone later recounted the remark alone, with no account of the context, that would not count as a joke.

4 Previous work: self-contained puns

4.1 The programs

Since 1992, there have been a small number of programs which created puns (cf. [Ritchie, 2004, Ch 10]). (Jokes depend on cultural knowledge, and puns rely heavily on linguistic knowledge, so some of these examples may be puzzling to readers from other cultural or linguistic backgrounds.)

Lessard and Levison [1992] devised a program which created a type of pun, the *Tom Swifty*, exemplified by (5).

(5) "Turn up the heat," said Tom coldly.

The form consists of a quoted utterance, and an adverbially modified attribution of this remark to Tom. The meaning of the utterance, or some subpart of it, is semantically linked to the adverb. The generation is based on finding a configuration of a root word (e.g. *cold*) which can be made into an adverb (*coldly*) and a sentence somehow linked to the root word.

The JAPE program [Binsted, 1996; Binsted and Ritchie, 1997] generated punning riddles of certain types, illustrated by (1), (6), (7), (8) (cf. [Lessard and Levison, 1993]).

(6) How is a nice girl like a sugary bird?
Each is a sweet chick.

(7) What is the difference between leaves and a car?
One you brush and rake, the other you rush and brake

(8) What do you call a strange market?
A bizarre bazaar.

This was achieved by various rules which specified allowable combinations of lexical entries and surface words; more details are given in Section 4.2 below.

The HCPP [Venour, 1999] could create two-utterance texts, where the second part was always a short noun phrase, as in (9) (punning on *doc*/*dock*) and (10) (*carrel*/*carol*).

(9) The surgeon digs a garden. A doc yard.

(10) Joan hears wailing in the booth. Carrel singing.

Venour describes his method in terms which borrow both from Binsted and from Lessard and Levison.

The puns produced by the WISCRAIC program [McKay, 2002] came in three forms, exemplified by (11), (12), (13).

(11) Who broke the woman's hart?
The cruel deer-keeper.

(12) The performing lumberjack took a bough.

(13) Your mate Johnny is a hard-up deer-keeper. He really needs doe!

Like the earlier programs, WISCRAIC operated by finding sets of items (e.g. *take a bow*, *bough*, *lumberjack*, *performing*) which were related in specific ways, then slotting them into stock textual forms.

The output of all these programs consists of *self-contained puns* in the sense given in Section 3 above.

4.2 An architecture for self-contained puns

Some of the pun-generators (Lessard and Levison, Venour) reviewed above were implemented using the Vinci language generator [Levison and Lessard, 1992], others (Binsted, McKay) using various Prolog facilities, including definite clause grammars [Pereira and Warren, 1980]. However, the general flow of processing in these systems is broadly the same, and can be described using the model used by [Binsted, 1996] (see also [Ritchie, 2003]). We will look at it in more detail, partly to make more concrete some aspects of these puns, and partly to show how these mechanisms relate to more conventional NLG.

The JAPE processing falls into three stages: *content selection*, *constituent building*, *surface string construction*, each of which is controlled by particular types of symbolic rule. (Binsted calls the constituent-building stage *SAD generation*, where 'SAD' stands for *small adequate description*, because in JAPE it leads to short descriptive noun phrases.)

We shall describe each of these stages in turn.

Content selection

In this phase, a small number of items are selected from the lexicon as the core of the pun. The generator has rules about suitable combinations of lexical items and/or surface words, and uses these to search for a cluster of workable items.

Binsted calls the content selection rules *schemas*. A pun-generator program would have a number of schemas (JAPE-3 had 15), each corresponding to some subclass of pun. A schema contains two rather different types of information:

Precondition. A schema has a condition which a sequence of lexical entries or surface strings (intuitively, the parameters for the schema) must meet in order for this type of pun to be generated. The precondition consists of a conjunction of terms, where each term is a predicate (e.g. `homophone`) applied to variable arguments, and all variables are existentially quantified. (None of the programs appear to need constant values as arguments to predicates.) For example, one schema (from [Ritchie, 2003]) has the precondition:

```
noun_phrase(NPLex),
component_lexemes(NPLex, LexA, LexB),
written_form([LexA], WordA),
homophone(WordA, HomWord),
written_form([HomLex], HomWord)
```

where some of the variables (`NPLex`, `LexA`, `LexB`, `HomLex`) are to be matched against *lexemes* (abstract identifiers for lexical entries) and others (`WordA`,`HomWord`) are matched against surface strings. One possible set of bindings is `NPLex` = `serial_killer`, `LexA` = `serial`, `LexB` = `killer`, `WordA` = 'serial', `HomWord` = 'cereal', `HomLex` = `cereal`.

Although many examples can be produced using predicates which are computable using standard lexical information (e.g. `homophone`), some of the puns (particularly those in McKay's system) need semantic relationships which, while not seriously problematic, might not be standard lexical relations; for example an action – *take a bow* – which is "typical" of an entity with a particular property – *performing*. Covering these might need a more encyclopaedic knowledge base.

Interface to later stages. This part of a schema specifies what is to be done with (some or all of) the values which satisfy the preconditions. It contains a formula indicating how further processing of the items is to be carried out, specifically what sorts of constituents they are to be built into.

For example, the JAPE schema illustrated immediately above has the *output specification* (taking some liberties with notation to save space):

```
<same, sad(share_properties, [HomLex, NPLex]),
       sad(make_phrase, [HomLex, LexB]) >
```

This (rather obscurely) indicates that the values `HomLex`, `NPLex` are to be used to create a phrase which, semantically, has some mix of their properties (e.g. *a murderer with fibre*), and that `HomLex`, `LexB`, are to be made directly into a phrase (hence *cereal killer*). Also, the final text should convey that the abstract relation `same` holds between these two constituents. That is, the output specification of a schema provides a recipe for producing a structure which, while not actually a text, is much closer to the eventual surface form. The various building procedures (e.g. `share_properties`) have a certain amount of non-deterministic freedom (see below), so that there may be more than one way in which the linguistic form of the phrase can be constructed, without affecting the basic joke; for example, `share_properties` might create *a crunchy murderer*.

Constituent building

Given the items found in content selection, it may be necessary to form constituents (noun phrases, verb phrases, sentences) which either contain these items or bear some systematic relationship to them. In this stage, linguistic representations of these chunks are constructed, following the output specification of the schema.

The constituent building stage is a relatively arbitrary mapping from the formulae provided by the schema to something which is more of a surface semantic structure. This is done by pattern-action rules which match against the various 'output specifications' that can be come from the schema handler, and produces something either in surface linguistic form (e.g. a string of words) or some skeletal linguistic item whose exact details (e.g. person, number) are left to the final (surface string) stage. Thus some variation is possible in the linguistic form of phrases, within the limits set by the initial schema match. That is why Binsted introduced this middle stage (not present in an earlier version of JAPE): to allow some linguistic variety where this does not alter the underlying joke. For example, (14) requires the same schema (and surface template) as (8), but uses a different constituent for the question; this difference would occur in the constituent-building phase.

(14) What do you call an odd mall?
A bizarre bazaar.

Surface string construction

In the third and final stage, a complete surface text is built by transforming the semantic constituents into surface strings and concatenating them, in some order, with various fixed strings of words. Minor adjustments such as number agreement, or choosing between *a* and *an*, are carried out. All this is driven by *templates*, which are like DCG rules with some pre-conditions attached.

4.3 Discussion of the mechanisms

The mechanisms summarised here were designed solely to produce puns, but the notions of "schema" and "template" are not radically (or interestingly) different from those established within mainstream NLG (e.g. [McKeown, 1985], [Kukich, 1983]). However, the way that this whole architecture is deployed is unusual. Normally in NLG, we can reasonably make a distinction between some background *knowledge base* (e.g. linguistic rules, lexicon, general facts about the domain) and some *message* which is to be conveyed. A typical schema-based system then matches its schemas against the message to determine applicability, with the background knowledge being called upon only as needed in the match.

In contrast, the pun generators have *only* a knowledge base, and do not start from a message to be conveyed: they are random generators of arbitrary puns. Hence the schemas are tested against the whole lexical knowledge base, to see if *any* sequence of items will meet the stated preconditions. Rather than asking, as in informative NLG, 'is the message of this general shape?', the pun generator asks 'are there any items which can be put together in this way?'. Thus content selection (which, conventionally, would precede schema-matching) is inherent in this search for matching items.

Also, the remits passed from the schema-instantiator to the constituent builder may be extremely vague, and not directly related to any communicative goals. For example, generating "Tom Swifties" requires, for the utterance part of the text, a constituent to be constructed which is, roughly speaking, 'any sentence which uses this specific word'.

These processing characteristics follow naturally from the fact that the pun generators produce self-contained puns, and hence have no communicative goals: any text will do, regardless of content, providing it constitutes a pun.

Hence, although schemas and templates are standard NLG notions, the freedom allowed to the processing model is less normal. Is this model of wider use? One possible application where this arrangement (arbitrary content but constrained form) might be of use is a language-teaching tool in which the illustrative examples are not all pre-programmed by hand but are machine-generated (either in advance or in response to the interaction with the human learner). It is conceivable that a such a system might need to create phrases or sentences which are not embedded in any real context, and for which the information conveyed is not critical, but whose linguistic structure must manifest some particular property (e.g. using some specific word, or being in passive voice). In that case, the unusual priorities embodied in the JAPE-style architecture (linguistic form over content) might be relevant.

The three-stage pun generation architecture allows quite efficient processing, but the initial lexicon search could be costly if done crudely. However, given the well-defined nature of that search, it can be optimised in various ways; preliminary unpublished results on the STANDUP project [Waller *et al.*, 2005] suggest that fast schema-instantiation from a realistically large lexical database is quite feasible.

4.4 Usefulness of self-contained puns

Although a self-contained pun is of use only where it would be helpful to interject a joke for its own sake, it could have a limited degree of contextual connection, in the following way. The status of the text as a pun is not context-dependent (otherwise it would be a contextually integrated pun), but it could mention topics which are currently salient. For example, Loehr [1996] describes a very small prototype system in which the user interface occasionally tells the user a (JAPE-generated) joke. Loehr explores limited contextual linking, based on keywords (for example, using a joke involving the word *aid* when help was needed).

Such loose subject matter links do not require computer joke-generation, but could be achieved from an database of jokes which had been thoroughly cross-indexed. This might be a viable way to add humour to a user-interface. Some investigation would be needed of what types of semantic links give rise to the most appropriate jokes, and the cross-indexing, to be automated, would have to depend on well-defined mechanisable relations (e.g. from a lexical database).

The main weakness of using self-contained puns in practical applications (such as those suggested by writers cited in Section 2) is the lack of a subtle connection to what is going on at the moment of delivery. Nevertheless, isolated or random jokes could still play a role in some applications:

Virtual agents: If a life-like image is to have a facetious "personality", then producing the occasional joke (even a bad one) might be fitting (as in Loehr's system).

Teaching language: An interactive system for assisting language learners might use jokes as subject matter to be studied or played with (as proposed by McKay, or by [Manurung *et al.*, 2004]). In such cases, the jokes might not need to be tightly related to context.

Again, both these applications could meet the joking requirements using a database of jokes, although proponents of the educational application might argue that the effect will be enhanced if the learner can experiment with joke-building using an interactive system; cf. [Waller *et al.*, 2005].

Notice that even a witty advertising slogan, although typically used in an isolated, context-independent fashion, is more sophisticated than the types of self-contained pun that have so far been computer generated, in that the content of a slogan is not arbitrary: it has to convey a particular message. In contrast, the systems reviewed in Section 4.1 above give no regard to the communicative content of the output text.

5 A general definition

From (2), (3) and other examples, Ritchie [2004] argues that there is a relatively well-defined, but large, subclass of puns which can be summarised thus:

(i) part of the utterance is phonetically similar (perhaps identical) to some other string not present in the utterance;

(ii) either the utterance, or the utterance with that other string substituted in, is contextually appropriate;

(iii) if the two substrings are identical, then they should be lexically analysable in different ways, and the lexical analysis of the one not in the utterance should either be linked semantically to the context, or should involve

grouping together words which are separate within the utterance;

(iv) if the two substrings are merely similar, then the unspoken one should form a complete and recognisable linguistic unit (e.g. a complete word or an established phrase). [Ritchie, 2004, pp.125-126]

This is probably the best available formal definition of this kind of pun (see [Davies, 2004]), but it has some limitations (which Ritchie documents) and some weaknesses:

(i) This covers only *paradigmatic* puns, where the comparison is between two strings, only one of which actually appears in the text. There are also *syntagmatic* puns where *both* the similar/identical strings appear in the text; (7) and (8) are very simple examples.

(ii) Where a sequence of words is involved in the pun, even the unspoken sequence must "make sense" in some way; that is, a pun is not formed when a contextually appropriate remark has a portion which is phonetically identical to some random sequence of words (unless the individual words qualify as puns separately). We shall assume this further condition in our discussions.

(iii) In the case where the two strings are merely similar (but not identical) the definition puts no restriction on which should appear in the actual text (providing the non-present text forms a well-known phrase). However, consider an example like (3), above. The headline would not have worked as a pun had it simply read '*Supercalifragilisticexpialidocious*', as this would not have allowed recovery of the contextually-appropriate message (even though it would still conform to Ritchie's definition). The correct condition may be better stated in terms of the ease with which the absent string can be summoned up or reconstructed, although this is not an unproblematic notion [Hempelmann, 2003]. It may be that there is a trade-off between the degree of phonetic similarity and the extent to which the string outside the text is a well-known phrase. Ritchie also cites the headline (15), but this may work because *un-bolivia-ble* is not a valid word, which may draw attention to the possibility that another word or phrase should be considered.

(15) Some South American stamps are un-bolivia-ble

The word *unbelievable* is not a well-known item in the sense of being famous, but it is cohesive as a morphologically complex word.

(iv) The previous point suggests that the condition stipulating that *either one* of the two similar strings should form a contextually appropriate part of the utterance may, in some cases, be too loose.

(v) Ritchie adopts the widespread assumption that puns are defined on phonetic strings, as this allows a natural description of the central relationship (phonetic similarity). However, many puns are conveyed in written text, and most NLG systems produce textual rather than spoken output. When a pun involves two phonetically identical strings, the question of which string appears in the actual utterance is moot when all the representations are phonetic; when the text is orthographically represented, the choice of string may affect whether the pun is noticeable to the reader. It is arguable that (2) would be more obviously a pun if the final sentence were *Your bag's leeking!* That is, it may improve the effectiveness of the pun to use the textual form which is *not* completely linguistically appropriate (here, there is no verb *to leek*).

Ritchie [2004, p. 199] gives a more formal version of his definition, which makes it clearer what primitive concepts it depends upon. These are:

Phonetic similarity. Study of an extensive range of puns (e.g. [Sobkowiak, 1991]) shows that the degree (and nature) of similarity required for punning is not obvious.

Contextually appropriate. Puns are part of a text which "makes sense in context". Although this is not unproblematic, it is a condition which an NLG system should aim for, even if no punning is intended; that is, it is not a further condition imposed by the aim of punning.

Linked to the context. This is a looser notion than 'contextually appropriate'. The linking just means that some concepts which are salient in the context are somehow related to the concept(s) mentioned in the word or phrase; e.g. in a situation where cooking is being discussed, words like *grill* or *baste* would be linked to the context.

Forming a recognisable word/phrase. Puns can depend on a sequence of words being matched against a single word, or a well-known idiom, motto or quotation. It is not clear when a phrase is sufficiently established to qualify.

The above definition covers contextually integrated puns relatively naturally – see (2), (4) and the examples in [Ritchie, 2004, Ch 9] – and is therefore directly relevant to the possible generation of such puns. Less obviously, the definition could be seen as covering self-contained puns (and hence the puns in Section 4), as follows. The early part of the text (e.g. the question in a riddle) can be treated as the "context", and the latter part (e.g. the answer to a riddle) as forming the pun utterance proper. That is, self-contained puns can be seen as puns with their own "context" built in, rather than linking to some context outside the text.

6 Computing contextually integrated puns

6.1 The usefulness of contextually integrated puns

Many of the advocates of useful computational humour seem to have in mind some form of "wit" by the computer system, rather than the mere spouting of jokes. The hypothetical illustrations given by Binsted[1995], for example, involve droll observations by the computer about *what is happening at the moment*. Such goals take us in the direction of contextually integrated puns. More concretely, if puns delivered within a quasi-natural dialogue are not contextually appropriate in a non-trivial way, there is little point in wasting time with computer generation; as noted earlier, a large and well-indexed database of jokes would be much more straightforward.

Let us consider what would be needed for a system to produce a text which met all the pun conditions given in Section 5. We assume that the NLG system will already be striving for a text which makes sense (in context), so we need only consider how the further pun conditions might be achieved. All of these requirements are decidable in principle, although (as noted above) there are non-trivial problems in defining what should count as sufficient phonetic similarity, what counts as an established phrase, and what counts as linkage to a context.

6.2 Detecting puns

Suppose some communicative system is to enhance its verbal output with puns. It seems reasonable to propose that such a system (particularly in cases where this behaviour is to be part of a life-like "personality") should be "aware" of having made a pun. That is, if the system makes a pun, it should have some record of that having happened, so that it can respond appropriately to, or even anticipate, the user's reaction. Ideally, the system should also be able to *avoid* humour when this would be socially inappropriate.

This suggests that the most minimal form of pun-computation would be checking textual output to see if it contains accidental puns. The system would then be in the position to make sense of any reaction by the user to the pun (or to eliminate any accidental but undesirable humour). This would not involve any special-purpose text construction – the system would merely check the output it had designed to meet its current (non-punning) goals. If unwanted humour is to be eliminated, this is more difficult, as it would require some form of revision component, which is not trivial.

Such a scheme could be seen as a special case of a more general post-checking approach, which tests for other desirable or undesirable properties, such as accidental ambiguity. (General ambiguity might even constitute a form of fortuitous humour, but that goes beyond the basic punning we are currently considering.)

As we shall discuss below, the full pun definition might lead to complex computations. A simplification which should be more tractable (but might miss more complex puns) would be merely to check each lexical item in the text to determine whether it had a homophone which was linked to the context. In order to ensure that the user was aware of the pun, the homophone might have to be substituted into the text (as in our method below, *Substitution with identity*).

Notice that incomplete coverage is something of a flaw in a checking mechanism like this, as the aim is for the system to spot *every* occasion on which it makes a pun; if it misses some, then the user might respond to a pun in a way which the system cannot easily interpret (or an unwanted joke might slip through). On the other hand, a pun-production mechanism, like those discussed below, need not be comprehensive, as long as it can produce *some* puns (and does not cause the text to become faulty in other respects).

6.3 The search problem

Our working definition (Section 5) involves a combination of conditions, some of which result in considerable search: finding *some* substring of the utterance, finding *some* string not in the utterance (a very large set!), finding *some* lexical analysis of the non-utterance string which meets certain conditions, finding *some* well-known word or phrase which is similar. The definition also has a few disjunctions, to further increase the search. Hence, even a non-constructive check (Section 6.2) would involve a significant amount of searching, particularly if naïvely implemented.

Some of the simpler cases considered below (e.g. substituting a contextually-linked homophone) might fit naturally into a NLG system based on constraint-satisfaction (CS) (cf. [Power, 2000]). However, CS methods would not completely remove the complexity problems involved in implementing the entire definition from Section 5. CS reduces processing in cases where the possible values of the variables are well-defined and easily enumerable, and evaluating individual constraints is relatively cheap; i.e. where the main computational load is in testing compatibility among chains of values. Here, the main work is elsewhere. Conditions such as testing whether some substring of some possible output string is similar to some well-known phrase would require computationally expensive enumeration of the basic values (possible substrings of possible output texts), and non-trivial conditions (phonetic similarity) involving very large sets (all words and well-known phrases). CS might be slightly better than a naïve search, but it would not be a panacea.

One complicating factor is that the pun criteria are largely *surface constraints* which apply to what, in a conventional pipeline NLG architecture [Reiter and Dale, 2000], would be the final output of the generator. Hence, it may be difficult for high-level (early) stages of a pipeline generator to make syntactic or lexical decisions which will result in puns, nor is it simple to effect "revisions" to a surface text which has been the outcome of much high-level processing. There is not space here to explore the large issue of surface-level constraints and their consequences for NLG architecture, but see [Reiter, 2000] for some discussion.

Puns are not the only forms in which surface constraints are central: poetry generation [Manurung *et al.*, 2000; Gervás, 2002] makes comparably awkward demands.

6.4 Some possible devices

We can consider some possible ways in which an NLG system might include contextually integrated puns.

Substitution with identity. The crudest approach to actual pun-generation would be to attach a punning module as a final stage. This module would review the entire text as generated and see whether it could be edited to form a pun, while making as few revisions as possible – preferably none – to previous higher-level decisions. The simplest tactic here would be the substitution of a phonetically similar (and internally coherent) string for some subpart of the message, where the substituted string does not represent the same lexical items as those in the original message, and either the content of the substituted string is somehow linked to the context or the substituted version should represent a finer segmentation of the material into words. Even for this very basic method, the search is considerable. A simplified version could be limited to looking for homophones of words in the message, and sub-

stituting these providing they were contextually linked; this is probably the most tractable option.

Substitution with similarity. A slight extension of the previous method would be to relax the condition from phonetic identity to phonetic similarity. However, some care would be needed in deciding under what conditions this should be permitted, in view of the point raised earlier about the "recoverability" of the contextually appropriate string. (This extension worsens the search problem.)

Minor rephrasing. It is conceivable that a surface-based editing component could detect that a minor re-wording would fulfil the pun conditions. However, such changes should not impair the overall message of the text, which means that this is not an easy solution.

Lexical preference. Rather than using a post-editing stage, we could build the pun facilities into a higher level of the NLG system. One possibility would be some form of amended lexical choice, within the NLG process (i.e. not post-editing as in the above mechanisms). The system could, when choosing a lexical item, give preference (if it would not distort the intended message) to a word which has a homophone which is in some way linked to the context. It might also be helpful to use that homophone in the text, to make the pun obvious to the reader, as noted previously. An extension would be to have the lexical choice system seek a *phonetically similar* item (not a strict homophone) which is associated with the context. In this case, inserting that other word in the text (in place of the more directly correct one) would be not just helpful but necessary.

Rephrasing for lexical preference. The previous device could perhaps be generalised, depending on how the generator organises lexical choice, to giving preference to a phrasing (not just the choice of a single word in isolation) which contains a word with a homophone/similar word which meets the pun conditions. This increases the amount of searching that the NLG system would require.

Matching against phrases. A further generalisation would be for the system to seek not an identical/similar *word*, but a *phrase* (similar/identical to a valid wording of the intended message) which met the pun conditions (perhaps, as in example (3), matching one word against several). These conditions (Section 5) for a phrase are slightly different from those for a word. To consider matching all possible well-known phrases (assuming such a database were available) when constructing a sentence would lead to high search costs.

Some of these processes could be quite open-ended, as puns (human-constructed) sometimes involve quite global rephrasing in order to achieve the punning effect. The subeditor who created (3) must have planned the entire headline with the word *Supercalifragilisticexpialidocious* in mind.

The later tactics (*Rephrasing for lexical preference*, *Matching against phrases*) would be more easily incorporated into a generator which was *opportunistic* about what material to include. For example, the ILEX text generator [Oberlander *et al.*, 1998] has a content selection stage in which the knowledge base is traversed to gather facts to be expressed. This is not as free as the process in Section 4.2 above, as ILEX starts from a particular knowledge base object which is to be described, and the search is for facts relevant to that object. The STREAK system [Robin, 1994] can produce variants of a text in which different amounts of supporting information (for the central proposition) are included. The choice of this additional information could allow more freedom to a pun-devising algorithm (although still with significant search).

Even if any of these pun-creating schemes were implemented, it is quite likely that in many cases no suitable match would be found, and text realisation would have to proceed as normal. That is, the pun-condition checking might be wasted effort, just adding to the generation time for the text.

7 Conclusions

We have seen that existing pun generators assume a relatively simple rule-based generation architecture, but the way that this architecture has been used is not typical of NLG tasks. These generators produce self-contained puns, but such jokes would be of potential use only in certain applications (where bald joke-telling is acceptable, or where language is to be studied rather than used in natural communication). The case for using contextually integrated puns to enhance computer systems is more plausible, but in the most general case this could lead to computational complexity. Nevertheless, there may be some tractable "tricks" which an NLG system could use to produce very simple contextually-integrated puns.

Acknowledgements: This paper benefitted from comments from the Aberdeen NLG group, and from Ruli Manurung.

References

[Binsted and Ritchie, 1997] Kim Binsted and Graeme Ritchie. Computational rules for generating punning riddles. *Humor: International Journal of Humor Research*, 10(1):25–76, 1997.

[Binsted, 1995] Kim Binsted. Using humour to make natural language interfaces more friendly. In Hiroaki Kitano, editor, *Proceedings of the International Joint Conference on Artificial Intelligence Workshop on AI and Entertainment*, pages 55–57, Montreal, 1995.

[Binsted, 1996] Kim Binsted. *Machine humour: An implemented model of puns*. PhD thesis, University of Edinburgh, Edinburgh, Scotland, October 1996.

[Davies, 2004] Christie Davies. Review of 'The linguistic analysis of jokes'. *Journal of Literary Semantics*, 33(2):196–197, 2004.

[Gervás, 2002] Pablo Gervás. Exploring quantitative evaluations of the creativity of automatic poets. In Carlos Bento, Amilcar Cardoso, and Geraint Wiggins, editors, *2nd Workshop on Creative Systems, Approaches to Creativity in Artificial Intelligence and Cognitive Science, ECAI 2002*, Lyon, France, 2002.

[Hempelmann, 2003] Christian Hempelmann. *Paronomasic Puns: Target Recoverability Towards Automatic Generation*. PhD thesis, Purdue University, 2003.

[Hulstijn and Nijholt, 1996] Joris Hulstijn and Anton Nijholt, editors. *Proceedings of the International Workshop on Computational Humor*, number 12 in Twente Workshops on Language Technology, Enschede, Netherlands, September 1996. University of Twente.

[Kukich, 1983] Karen Kukich. Design of a knowledge-based report generator. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, pages 145–150. ACL, 1983.

[Lessard and Levison, 1992] Greg Lessard and Michael Levison. Computational modelling of linguistic humour: Tom Swifties. In *ALLC/ACH Joint Annual Conference, Oxford*, pages 175–178, 1992.

[Lessard and Levison, 1993] Greg Lessard and Michael Levison. Computational modelling of riddle strategies. In *ALLC/ACH Joint Annual Conference, Georgetown University, Washington, DC*, pages 120–122, 1993.

[Levison and Lessard, 1992] Michael Levison and Greg Lessard. A system for natural language generation. *Computers and the Humanities*, 26:43–58, 1992.

[Loehr, 1996] Dan Loehr. An integration of a pun generator with a natural language robot. In Hulstijn and Nijholt [1996], pages 161–172.

[Lyttle, 2001] Jim Lyttle. The effectiveness of humor in persuasion: The case of business ethics training. *Journal of General Psychology*, 128(3):206–216, April 2001.

[Manurung *et al.*, 2000] Hisar Maruli Manurung, Graeme Ritchie, and Henry Thompson. A flexible integrated architecture for generating poetic texts. In *Proc of the Fourth Symposium on Natural Language Processing (SNLP 2000)*, pages 7–22, Chiang Mai, Thailand, 2000.

[Manurung *et al.*, 2004] Ruli Manurung, Alistair Low, Lucia Trujillo-Dennis, David O'Mara, Helen Pain, Graeme Ritchie, and Annalu Waller. Interactive computer generation of jokes for language skill development. Talk at Annual Conference of the International Society for Humor Studies, June 2004. Dijon, France.

[McKay, 2002] Justin McKay. Generation of idiom-based witticisms to aid second language learning. In Stock et al. [2002], pages 77–87.

[McKeown, 1985] Kathleen McKeown. *Text Generation*. Studies in Natural Language Processing. Cambridge University Press, Cambridge, UK, 1985.

[Morkes *et al.*, 1999] John Morkes, Hadyn K. Kernal, and Clifford Nass. Effects of humor in task-oriented human-computer interaction and computer-mediated communication: A direct test of srct theory. *Human-Computer Interaction*, 14(4):395–435, 1999.

[Nijholt, 2002] Anton Nijholt. Embodied agents: A new impetus to humor research. In Stock et al. [2002], pages 101–111.

[Oberlander *et al.*, 1998] Jon Oberlander, Mick O'Donnell, Alistair Knott, and Chris Mellish. Conversation in the museum: experiments in dynamic hypermedia with the intelligent labelling explorer. *New Review of Hypermedia and Multimedia*, pages 11–32, 1998.

[O'Mara and Waller, 2003] Dave A. O'Mara and Annalu Waller. What do you get when you cross a communication aid with a riddle? *The Psychologist*, 16(2):78–80, 2003.

[Pereira and Warren, 1980] F. Pereira and D. H. D. Warren. Definite clause grammars for language analysis – a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13:231–278, 1980.

[Power, 2000] Richard Power. Planning texts by constraint satisfaction. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING-2000)*, pages 642–648, Saarbrücken, Germany, 2000.

[Reiter and Dale, 2000] Ehud Reiter and Robert Dale. *Building Natural Language Generation Systems*. Cambridge University Press, Cambridge, UK, 2000.

[Reiter, 2000] Ehud Reiter. Pipelines and size constraints. *Computational Linguistics*, 26(2):251–259, June 2000.

[Ritchie, 2003] Graeme Ritchie. The JAPE riddle generator: technical specification. Informatics Research Report EDI-INF-RR-0158, School of Informatics, University of Edinburgh, Edinburgh, February 2003.

[Ritchie, 2004] Graeme Ritchie. *The Linguistic Analysis of Jokes*. Routledge, London, 2004.

[Robin, 1994] Jacques Robin. *Revision-Based Generation of Natural Language Summaries Providing Historical Backgrouns: Corpus-Based Analysis, Design, Implementation and Evaluation*. PhD thesis, Columbia University, 1994.

[Sobkowiak, 1991] Wlodzimierz Sobkowiak. *Metaphonology of English Paronomasic Puns*, volume 26 of *University of Bamberg Studies in English Linguistics*. Peter Lang, Frankfurt, 1991.

[Stock *et al.*, 2002] Oliviero Stock, Carlo Strapparava, and Anton Nijholt, editors. *Proceedings of the April Fools' Day Workshop on Computational Humor*, number 20 in Twente Workshops on Language Technology, Enschede, Netherlands, April 2002. University of Twente.

[Stock, 2002] Oliviero Stock. Computational humor. In Stefano A. Cerri, Guy Gouardères, and Fábio Paraguaçu, editors, *Intelligent Tutoring Systems: Proceedings of the 6th International Conference*, Lecture Notes in Computer Science, pages 2–3, Berlin, 2002. Springer. Invited talk.

[Stock, 2003] Oliviero Stock. "Password Swordfish": Verbal humor in the interface. *Humor: International Journal of Humour Research*, pages 281–295, 2003.

[Venour, 1999] Chris Venour. The computational generation of a class of puns. Master's thesis, Queen's University, Kingston, Ontario, 1999.

[Waller *et al.*, 2005] Annalu Waller, Dave O'Mara, Ruli Manurung, Helen Pain, and Graeme Ritchie. Facilitating user feedback in the design of a novel joke generation system for people with severe communication impairment. In *Proceedings of 11th International Conference on Human-Computer Interaction*, Las Vegas, July 2005.