Type Coercion from a Natural Language Generation Point of View

Markus Egg and Kristina Striegnitz, Universität des Saarlandes

Abstract

We discuss the question of how phenomena of *type coercion* (TC) can be integrated into Natural Language Generation (NLG). The ultimate goal is the generation of expressions that instantiate type coercion in cases where such expressions are the most natural verbalization for a given piece of information.

Our approach is an extension of the SPUD NLG system developed by Matthew Stone. This system is well-suited for our approach because it allows the easy formulation of syntax-semantics interface constraints (in the form of semantically annotated fragments of TAG syntax trees). These interface constraints are indispensable to avoid overgeneration of TC expressions.

TC is integrated in the NLG system in that we formalize *TC operators* that provide additional options of generating expressions with a well-formed syntactic structure (to verbalize a given semantic input). TC operators are also represented by tree fragments and are subject to conditions on linguistic and other context features.

1 Introduction

The issue we are going to address in our paper is the question of how metonymy and related phenomena (referred to as phenomena of *type coercion*) can be integrated into Natural Language Generation (NLG). The ultimate goal is the generation of expressions that instantiate type coercion in cases where such expressions are the most natural verbalization for a given piece of information.

The relevance of this goal is best illustrated by way of examples. Imagine parents worrying about whether their offspring have done their duties as pupils. One of the possible inquiries stemming from this worry would be (1a), which is interpreted in the same way as (1b). However, this latter variant would be very odd in the given context:

(1) (a) Have you finished your homework?

(a) Have you finished preparing your homework?

Similarly, the appropriate answer to the German (2a) would be (2b), and not something like (2c):

(2) (a)

Wo	*ist*	*der*	*Wein?*
where	is	the	wine

‘Where is the wine?’

(b)

Er	*steht*	*auf*	*dem*	*Tisch.*
it	is-standing	on	the	table

‘It is on the table’ (lit: ‘it is standing on the table’)

(c)

Er	*ist*	*in*	*Flaschen,*	*die*	*auf*	*dem*	*Tisch*	*stehen*
it	is	in	bottles	RELPR	on	the	table	are-standing

‘It is in bottles that are on the table’ (lit: ‘... are standing on the table’)

These examples illustrate a trend in natural language towards brevity of expressions. Eventually, the explanation for this trend can be found in the Gricean (Grice 1975) conversation maxim ‘be brief’, which makes one choose the briefer of two equally informative expressions.

This trend is strong enough to enforce the choice of expressions that instantiate type coercion in cases like (1) and (2). These expressions are short because they are incomplete in that part of the information to be conveyed is not expressed. For instance, such a bit of information is the fact that it is the *preparation* of the homework (and not any other potential activity that involves homework) whose finishing it at stake in (1a), or the fact that it is not the wine but its *containers* that are in an upright position in (2b). World and/or context knowledge then steps in and provides these pieces of information.

From the viewpoint of natural language generation, the goal is to take type coercion into account in order to generate maximally brief expressions rather than their more long-winded alternatives. This means in particular that one must be able to divide a given input (semantic information) into information that is to be verbalized and information that should be left unexpressed, since it can be reconstructed from other knowledge sources.

The paper is structured as follows. After a brief review of the phenomenon of TC from an NLU perspective, we outline the issues to be dealt with in NLG approaches that aim at including TC. We then outline the SPUD NLG system and show in detail how one can integrate TC in the form of so-called *TC operators*. These operators provide additional options of verbalizing a given semantic input in terms of expressions that involve type coercion.

2 Type coercion from different perspectives

In this section, we recapitulate the phenomenon of type coercion. We will show that type coercion for NLG has not received much attention, hence, the issues that are crucial here remain somewhat implicit in the literature. Instead, the literature focuses on type coercion from the viewpoint of natural language understanding.

2.1 Type coercion in natural language understanding

Type coercion is almost exclusively discussed as a problem of natural language understanding (NLU) in the literature, e.g., in papers like Dölling (1992, 1995), Pustejovsky (1995), Moens and Steedman (1988), de Swart (1998), or Piñango (lume).

Formally, type coercion can be characterized by the schema $F(Op(A))$. I.e., rather than applying a functor F directly to its argument A, first a TC operator Op, which is not expressed explicitly in the utterance, is applied to A, then F is applied to the result of this first application. On the basis of this schema, the challenges posed by type coercion can be divided in three parts.

The first task is to *identify expressions that must undergo type coercion*. Characteristically, these sentences have no literal interpretation since their semantics comprises a mismatching functor-argument pair (often due to a violation of selection restrictions, e.g., in (2b)[1], but this need not be the case (see the discussion of this point in Egg 2000).

Second, one must *acquire information from additional knowledge sources* over and above the information from the syntax and semantics of an expression in order to obtain the full interpretation of an expression. This information is then used as the operator Op in the type coercion. E.g., in (1b) only world knowledge about the way in which pupils and homework are related provides the piece of information that the end of the *preparation* of the homework is at stake in this sentence.

The derivation of this additional information in general is at present an unsolved issue, the most systematic discussion of this issue is to be found within the framework of the *Generative Lexicon* (GL) (Pustejovsky 1995). This framework tries to capture TC in terms of an enriched semantic representation of lexemes (especially of nouns and verbs), the so-called *Qualia Structure* (QS). The QS comprises structured information on additional facts about entities, e.g., their origin and purpose. The GL framework furthermore assumes processes of semantic composition that can access this structured information.

Consider for instance sentence (1a): here the fact that homework comes into being by someone preparing it gives us the clue to the full interpretation. In the QS of the noun *homework*, this piece of information would fill the slot talking about the origin of homework.

The final task, then, is the *integration of this additional information with the result of semantic construction* to obtain a fully-fledged interpretation of the expression. To put it more formally, one must distinguish the relevant functor-argument pair in the result of semantic construction, between which the operator Op is to be inserted.

The syntax-semantics interface plays a crucial role in this distinction, since it determines the positions where this additional material is to be integrated with the result of semantic construction. E.g., in a TC case like (3) [= (2b) with the anaphor *it* resolved], A is the VP semantics (simplified to $\lambda x_{\text{THING}}.\mathbf{stand}'(x) \wedge \mathbf{on\text{-}table}'(x)$), while F is the semantics of the NP ($\lambda P \exists! y_{\text{SUBST}}.\mathbf{wine}'(y) \wedge P(y)$; here $\exists! x.P(x)$ expresses the existence of a unique entity in the extension of P):

(3) Der Wein steht auf dem Tisch

[1]Here the subject NP fails to comply to the verb's restriction to NPs that refer to things with a maximal axis.

Since (3) has no well-formed literal reading, the operator *Op* must bridge the mismatch between the incompatible sorts SUBST[ance] and THING. For (3), we assume an operator that maps properties *P* of containers onto the property (of substances) of having a container with the property *P*. Here, CONT is a relation between containers and their content:

(4) $\lambda P \lambda z \exists x. \mathrm{CONT}(x,z) \wedge P(x)$

The resulting interpretation of (3) is (5a), which claims the existence of a unique quantity of wine whose container is standing on the table:

(5) (a) $\exists ! y_{\mathrm{SUBST}}.\mathbf{wine}'(y) \wedge \exists x_{\mathrm{THING}}.\mathrm{CONT}(x,y) \wedge \mathbf{stand}'(x) \wedge \mathbf{on\text{-}table}'(x)$

(b) $\exists ! x_{\mathrm{THING}}.(\exists y_{\mathrm{SUBST}}\mathrm{CONT}(x,y) \wedge \mathbf{wine}'(y)) \wedge \mathbf{stand}'(x) \wedge \mathbf{on\text{-}table}'(x)$

The mismatch between these sorts could also be buffered by coercing the semantics of the *noun* into a property of things (the property of being a container of wine). The relevant operator is presented below as (8). However, this option would give the wrong interpretation (5b) for (3): in this semantic representation the uniqueness presupposition holds for wine containers. This does not rule out that there are other wine quantities around which are not in containers that stand on the table, but this is in conflict with intuitions on (3). This unwanted interpretation is barred in the synsem interface, since the interface does not license the insertion of a type coercion operator at this position.

2.2 Type coercion in natural language generation

Taking the speaker's point of view, type coercion is a means of leaving information (which can be recovered by the hearer through context and world knowledge) implicit. Thus, the problem that type coercion poses for NLG is complementary to the one sketched for NLU: the task is to *distinguish* pieces of information that are to be verbalized from those that are not. I.e., the task of NLG is the division of a given input (a semantic representation) into *F* and *A* on the one side (they are then verbalized) and *Op* on the other side (which is not). In this respect, type coercion plays a similar role in NLG as ellipsis or the phenomenon of indirect anaphor: in order to obtain expressions that sound natural, certain parts of the input to the NLG system (a semantic representation) may not be put into words (Rambow and Hardt 2001; Shaw 1998; Gardent and Striegnitz 2001).

E.g., the input to (1a) (omitting tense and sentence mood) would be something like (6):

(6) $\{\mathbf{finish}'(e, \mathbf{hearer}', e'), \mathbf{prepare}'(e', \mathbf{hearer}', x), \mathbf{homework}'(x),$
$\mathbf{of}'(\mathbf{hearer}', x)\}$

NLG then would have to single out the second element of this input as material that need - and, in fact, should not - be verbalized.

The role of the syntax-semantics interface for TC cases is consequently not the same in NLP and NLG. Rather than acting as a means of fixing TC positions in semantic representations of

TC cases (which then can be filled in with extralinguistically derived information), it *constrains* TC positions that are assumed during generation (where extralinguistically derivable parts of the input are omitted from verbalization). More formally: potential realizations of a given input in terms of a TC expression must have a syntactic structure that licenses TC. Ideally, this is modelled as an actual constraint, rather than as a filter on already generated expressions.

As an example for this constraining role of the syntax-semantics interface, consider for instance the input to (3). The semantics of the PP has been simplified and once again tense and sentence mood are neglected:

(7) $\{\mathbf{stand}'(e,x), \mathbf{container}'(x), \mathbf{wine}'(y), \mathbf{on\text{-}table}'(x),$
$\mathbf{contain}'(e',x,y), \mathbf{uniquely\_identifiable}'(y)\}$

At a first glance, there are two ways of distinguishing material that need not be verbalized in (7). On the one hand, one can spot the TC operator (4) that maps properties P of containers onto the property of having a container with the property P in (7). Formally, the second and the penultimate element of (7) constitute this operator: $\mathbf{contain}'(e',x,y)$ expresses the contain-relation itself, while $\mathbf{container}'(x)$ matches the presupposition of the operator (the first element of the contain-relation must be a container). Verbalizing all other elements of (7) gives us the desired expression (3).

But on the other hand, one might also identify the third and the penultimate of the elements of (7) with the TC operator (8) that is effective in (5b) and (9).

(8) $\lambda P \lambda z \exists x.\mathrm{CONT}(z,x) \wedge P(x)$

(9) Every bottle froze

This operator is kind of inverse to the one we encountered in (3), since it maps properties P of substances onto the property of being a container of a substance with the property P, viz., (8). CONT again relates containers and their content.

Once more $\mathbf{contain}'(e',x,y)$ would express the contain-relation, and $\mathbf{wine}'(y)$ would match the presupposition of the operator that the second element of the contain-relation must be a substance. However, the resulting verbalization would be something like ‘The containers are standing on the table’, which is not the preferred way of getting across the information in (7). In fact, this would not communicate the whole of (7), since the information that there is wine in the container would be omitted from verbalization. I.e., there would be the danger of losing a piece of information, unless this piece can be retrieved from other (e.g., contextual) sources.

We will show in section 3.2 that relating TC potential to syntactic constituent structure in the syntax-semantics interface bars the second, unwanted option without any further assumptions.

The NLG perspective on type coercion forces one to tackle the question of what type coercion operators there are right from the start. This question is quite a challenge for analyses of type coercion, because of the fact that type coercion is language and even genre-specific. E.g., the following sentences are only acceptable in French (Horacek 1996) and in waiters’ jargon, respectively. I.e., simply compiling a list of material that may be omitted from verbalization in NLG will not do:

(10) (a) Le Prix Goncourt est arrivé

(b) * The Pulitzer Prize has arrived (in the sense of ‘the winner of...’)

(11) The ham sandwich is waiting for his check (Nunberg 1979)

Worse still, the context of an expression may license cases of occasional type coercion that would not be understandable out of context:

(12) Max finished Dr Johnson today

(12) is not understandable as it stands. However, if we know that Max is an architect who was assigned the job of designing Dr Johnson’s new house, the interpretation of (12) is straightforward.

Due to the complementary tasks of NLU and NLG in the domain of type coercion, a mere adaptation of proposed approaches to type coercion (from the NLU perspective) to NLG purposes will not do. While one may take over a general framework of describing type coercion for the purposes of NLG, the two main problems TC poses for NLG are not covered by NLU approaches to TC.

First, one has to tackle the problem of identifying the material that need not be verbalized. Even if the syntax-semantics interface constraints the possibilities here, the identification of potential TC operators in a given input for NLG presupposes the availability of suitable models for context and world knowledge and a way of accessing them.

To answer this problem, Horacek (1996) proposes using the information of Pustejovsky’s *Qualia Structure* (QS) to handle cases of metonymy. While the strategy of taking QS information as potential reinterpretation operators might work in some cases, e.g., (13), because the purpose of bottles is to contain liquids, it is not available for cases like (14), since no quale for *wine* would introduce the notion of container:

(13) Amélie downed the bottle

(14) Amélie labeled the wine

In addition, this strategy can not yet model language-specific differences in the range of potential metonymy. To this end, Horacek must take recourse to explicit annotations that express information like ‘in German or English, no coercion is available for *begin* with an object NP whose head is *cigarette*’.

Second, there arises the question of the exact implementation of the TC phenomenon in NLG. It is necessary to specify the step in the generation process where the decision to produce a TC or a non-TC realization for a given input is made, where the material that need not be verbalized is identified, and how this is integrated with the rest of the generation process.

This issue surfaces also in Horacek’s (1996) approach, which handles the separation of the material that is not to be verbalized within general generation procedures that correlate specifically structured pieces of information from the input with linguistic material (mostly lexemes, but also other material like affixes). E.g., one of these procedures correlates a concept for an eventuality of possession with its possessor role and the bearer of this role into the noun *owner*.

In these procedures, pieces of the input can be deliberately omitted from verbalization. This omission characterizes, but is not restricted to, cases of generating TC expressions. It is eventually licensed by appropriately designed lexical entries, in particular, by their QS. The licensing condition is that part of the input structure matches specific parts of a lexical entry. But this requires rather involved subprocedures of pattern matching in order to detect all relevant matches.

In sum, the discussion of TC in the literature, most of which adopts the viewpoint of NLU, is useful for our goal in that it outlines basic properties of the phenomenon of TC. However this discussion does not cover the two main problems we identified for the integration of TC into natural language generation.

3 Not Generating TC Operators

In this section, we will expound how type coercion can be accounted for in a natural language generation system.

The system that we want to implement our approach in is based on the SPUD system developed by Matthew Stone (Stone and Doran 1997; Stone et al. 2001). SPUD is a system dealing with the 'how-to-say-it' part (as opposed to the 'what-to-say' part) of NLG. It starts from a communicative goal and then plans and realizes a text achieving this goal. Its planning process accesses different levels of linguistic knowledge (syntactic, semantic, and pragmatic) as well as background knowledge to construct an utterance which takes constraints imposed by the given situation into account.

As opposed to many other systems designed for dealing with this task, SPUD doesn't expect a detailed representation of the semantic content of the text that is to be produced, but starts from a communicative goal. The exact semantics needed for achieving the communicative goal is built in parallel with the syntactic structure. In this way, the decision of what exactly the semantic content has to look like can be based on linguistic constraints as well as the communicative goal that is to be fulfilled. For an NLG approach to type coercion that means that mechanisms à la Horacek (1996), where fragments of the semantic content are eliminated before planning the actual text, is not possible, since the full semantic content is simply not available at that point. So, we will have to integrate the decision of whether to apply type coercion or not into the process of *planning* the text. In doing so, we can use information provided by the syntax-semantics interface to constrain the choice on where type shifts are possible.

We will now introduce the generation system, before we proceed to present our approach to integrating type coercion in this system.

3.1 A Generation System

Since SPUD uses a Tree Adjoining Grammar we start by giving a brief introduction to this grammar formalism. We then show how the design of SPUD's lexicon connects syntactic, semantic

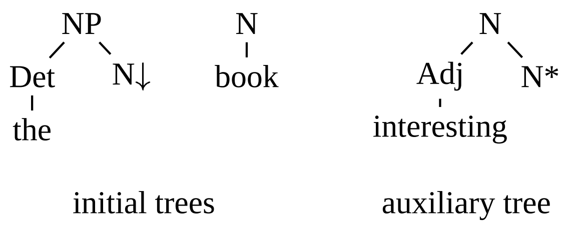


Figure 1: A TAG grammar

and pragmatic information. Finally, we illustrate the main mechanisms driving the generation process.

3.1.1 Tree Adjoining Grammar

For a precise definition of Tree Adjoining Grammars (TAGs) we refer the reader to Joshi and Schabes (1997). In what follows, we only sketch the basic properties of TAG.

The Lexicon of a TAG consists of trees, so called *elementary trees*, which are fragments of parse trees. Figure 1 shows some example lexicon entries. These trees can then be combined to form larger tree structures by two operations called *substitution* and *adjunction*, which we will explain below.

TAG distinguishes two types of trees: *initial* trees, which are used to encode the basic syntactic frame of syntactic functors, and *auxiliary* trees, which encode modifiers, e.g., adjectives, prepositional phrases (PPs), or adverbs. The distinguishing property of auxiliary trees is that they have a unique foot node (marked with $\star$), i.e., a frontier node labeled with the same category as the root of the tree. Furthermore, trees may contain substitution nodes (marked with $\downarrow$), which are leaf nodes labeled with a non-terminal category.

The two operations, substitution and adjunction, are then used to combine trees into bigger trees. Intuitively, substitution inserts an initial tree with root category X at some substitution node with category X in some other tree. Adjunction, on the other hand, caters for recursion and permits inserting an auxiliary tree with root and foot node category X at a node labeled with category X in some other tree. Substitution and adjunction are illustrated in Figure 2.

3.1.2 The Lexicon in SPUD

In the lexicon of SPUD, elementary trees are associated with semantic and pragmatic information (see Figure 3). The lexical entry for the noun *book*, for instance, is associated with the information that some entity x is a book, and the lexical entry for the transitive verb *finish* tells us that there is some event e_1 where an entity x finishes a second event e_2. SPUD uses a flat semantic representation (Hobbs 1985; Copestake, Flickinger, and Sag 1999) and the semantics of a complex tree is just the union of the semantic representations of the elementary trees it consists of. Furthermore, elementary trees may be associated with pragmatic constraints, such as for

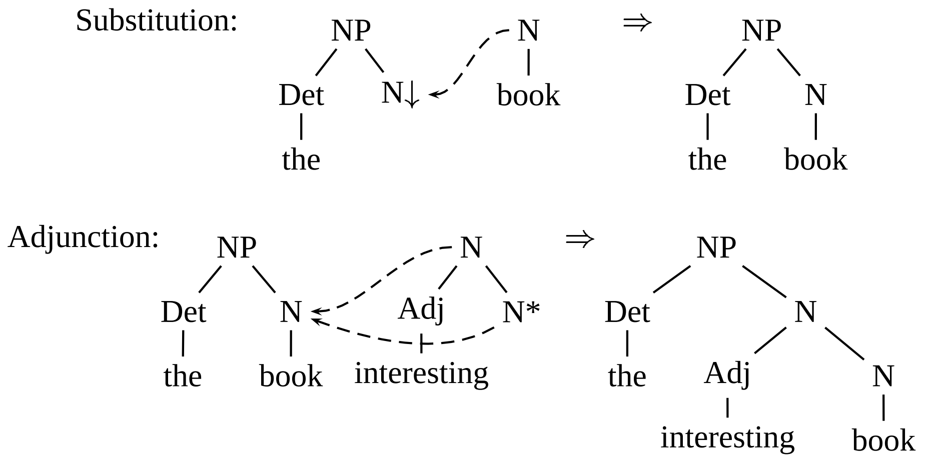


Figure 2: Combining trees: Substitution and adjunction

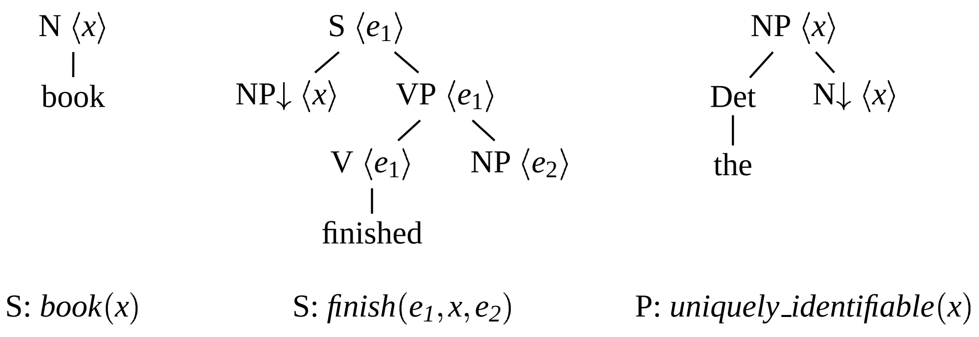


Figure 3: Semantic and pragmatic information in the lexicon.

instance the lexical entry for the definite article *the*. The pragmatic constraints are also collected as the tree is built. A syntax tree is only licensed in a given situation if its pragmatic constraints are all satisfied by the discourse context.

The *syntax-semantics interface* is encoded by associating syntactic constituents with semantic entities. The intuition behind this is that a constituent somehow describes the semantic entity it is associated with. More precisely, nodes in the elementary tree are labeled with feature structures where an attribute *idx* takes a variable over a semantic entity as value. In Figure 3 this is indicated by labeling nodes with $\langle X \rangle$ where X is some variable over a semantic entity.

The entity associated with the S, VP, and V nodes in tree for *finish*, for example, is co-referential with the first argument of *finish*. It describes the finishing event itself, while the entity associated with the subject NP node describes the agent of the finishing event and is therefore co-referential with the second argument of the semantic predicate *finish*.

3.1.3 The Generation Strategy

As said above, SPUD tries to construct an utterance which satisfies a communicative goal in a given situation. Communicative goals are of the form $\textit{describe}(e)$ where e is an entity (usually an event or state) which the speaker has additional information about. The task is then to build a sentence that describes e, i.e., whose root node is associated with e. The semantic content of this utterance is not fully determined before the sentence planning and realization step, but is extracted from the speaker's knowledge about e as it is needed by realization. It is however possible to specify pieces of information that must be conveyed by the utterance.

We are now going to illustrate how the generation algorithm works. At this point we depart from SPUD's original architecture and assume the chart-based generation algorithm described in Striegnitz (2001), instead of SPUD's greedy search algorithm. This won't make much difference for our current presentation, but lets us produce paraphrases in cases where several verbalizations are possible.

The basic generation algorithm can be outlined as follows: Given the communicative goal $\textit{describe}(e)$

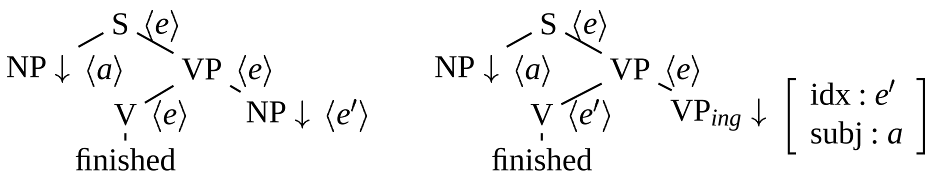
- select those elementary trees from the lexicon that realize properties of e
- for each of them, do substitution as long as there are open substitution nodes
- do adjunction if the tree is not semantically complete and contextually appropriate

Let's have a look at an example. Assume that we start from the following input and background knowledge.

Goal: describe e, where $\{\textit{finish}(e,a,e'),\ \textit{write}(e',a,b),\ \textit{1837}(t),\ e \subseteq t\ldots\}$

Common Knowledge: $\{\textit{woman}(a),\ \textit{named}(a,\text{'}\textit{Jane Austen}\text{'}),\ \textit{mansfield\_park}(b),\ \textit{book}(b)\ldots\}$

First, the algorithm selects trees from the lexicon that realize some property of e. There are (at least) two possibilities in the lexicon, which are shown in Figure 4. As the semantics gets

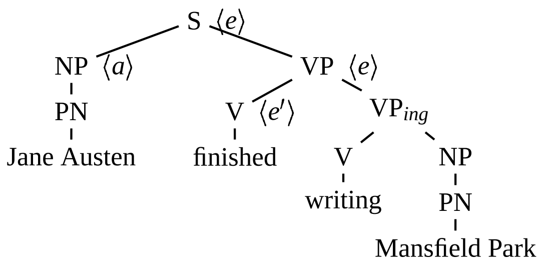


S: $finish(E,X,E')$ S: $finish(E,X,E')$

Figure 4: Lexicon entries realizing finishing events.

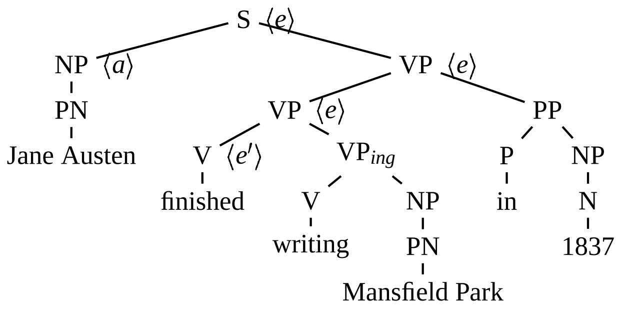
matched onto the semantics provided by the input the variables over semantic entities associated with the nodes of the syntax tree get instantiated as well. The tree on the left subcategorizes for an event denoting noun phrase (open NP substitution node associated with entity e'). In this example, it caters for verbalizations like *Jane Austen finished the writing/the creation of Mansfield Park in 1837*. The one on the right has an open VP substitution node (also associated with e') and is used in *Jane Austen finished writing Mansfield Park in 1837*, for instance. Note that both trees subcategorize for some constituent referring to the event e' that is to be finished. This makes realizations like *Jane Austen finished Mansfield Park*, i.e., realizations involving a type shift, impossible. We will now explain the mechanisms of how the second of those elementary trees can be completed to form an utterance that achieves the communicative goal, while ignoring how exactly this is done by the chart based algorithm. We want to point out, however, that the other paraphrase involving the other lexicon entry for *finish* can, of course, also be extracted from the chart.

Next, the open substitution nodes have to be filled. I.e., a noun phrase referring to Jane Austen and a verb phrase referring to the writing event are needed. They could for instance be realized as follows:



S: $finish(e,a,e'), write(e',a,b), named(a,`JaneAusten'),$
$mansfield\_park(b)$

Now we have a syntactically complete tree. No pragmatic constraints are unsatisfied, however, not all of the semantics has been conveyed. The date of the finishing event is still missing. So the algorithm tries to add this missing information by adjunction, yielding the following tree.



S: *finish*(e,a,e'), *write*(e',a,b), *named*$(a,$'*JaneAusten*'$)$, *mansfield_park*(b), *1837*(t), $e \subseteq t$

3.2 Type Coercion at the Syntax-semantics Interface

We will now turn to integrating the possibility of *not* verbalizing the TC operator into this generation system.

Similar to the strategy proposed by Egg (2000) and Pulman (1997) for *interpreting* type coercing expressions, the we propose to introduce positions at which type coercion can occur the syntax-semantics interface. We will start by showing how this intuition is incorporated into the lexicon. Then, we will illustrate by means of two examples how the generation algorithm works after this lexical extension. Finally, we will briefly talk about the problem of defining TC operators.

3.2.1 Type Coercion in the Lexicon

The basic idea is that verbs specify in their subcategorization frame that at certain positions a type shift may occur. In that case it is possible that an expression describing an entity different from a semantic argument is realized in the position of this argument. This entity must however be related to the original argument via a TC operator.

So, we have to supply two sorts of information, namely *where* type coercion may occur and what are viable TC operators. We account for this as follows: First, we will mark those places in the elementary trees of our lexicon where a shift in type can occur. Then we will provide elementary trees that correspond to coercion operators and carry the contextual constraints that restrict the use of certain operators. Proceeding in this way gives us a very fine grained control over the situations in which we allow type coercion. We can tune our generation system to the domain we are in and avoid the overgeneration problem that we have pointed out for more general approaches.

Figure 5 shows the lexical entry for the verb *finish* subcategorizing for a noun phrase referring to the event that has been finished. The feature *shift* with the value *finish*(E,X,E') indicates that a type shift can occur at this position while handing information about the entities involved in the type shift down to the tree that is substituted at this node.

Coercion can then be triggered by substituting elementary trees as those depicted in Figure 6 specifying the TC operator as their semantic content and contextual restrictions on the use of this operator in their pragmatic constraints. Let's look at the left tree. It changes the open substitution

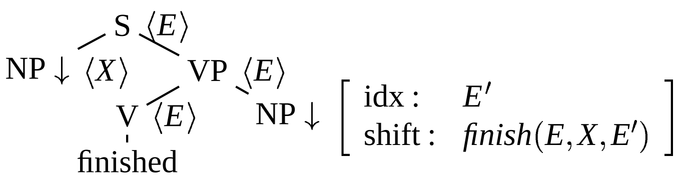
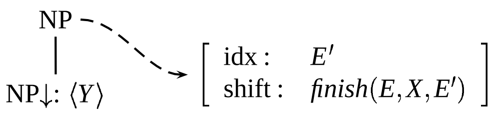


Figure 5: Type shifting information in the lexicon.

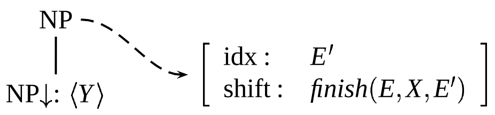
node from one that requires an NP referring to E' to one that requires an NP referring to Y. This TC operator tree carries the semantic content $\mathit{read}(E',X,Y)$ (which is conveyed implicitly, i.e. not verbalized) and is pragmatically licensed to be used, if it follows from the common ground plus the information given by the utterance that Y is a book.

In the case where the link between E' and Y is that E' is a writing event in which Y is being written (the right TC operator tree in Figure 6) the pragmatic constraints are stricter in that they also require that it is known to both hearer and speaker that X is an author.



S: $\mathit{read}(E',X,Y)$

P: $\mathit{Sem} \rightarrow \mathit{book}(Y)$



S: $\mathit{write}(E',X,Y)$

P: $\mathit{Sem} \rightarrow \mathit{book}(Y)$ & $\mathit{author}(X)$

Figure 6: The elementary trees which trigger coercion.

3.2.2 An Example

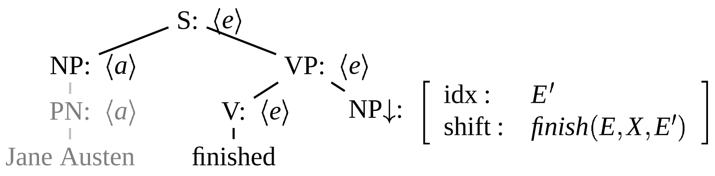
Now, let's look at an example of how our generation algorithm deals with situations in which type coercion is possible.

Assume that we start as before from the following input and background knowledge.

Goal: describe e, where $\{\mathit{finish}(e,a,e'),\ \mathit{write}(e',a,b),\ \mathit{1837}(t),\ e \subseteq t \ldots\}$

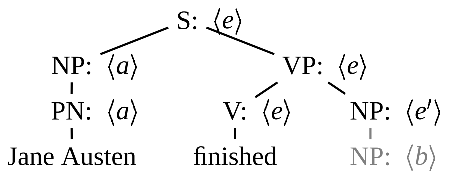
Common Knowledge: $\{\mathit{woman}(a),\ \mathit{named}(a, \text{'Jane Austen'}),\ \mathit{mansfield\_park}(b),\ \mathit{book}(b) \ldots\}$

In the first step, the algorithm selects trees from the lexicon that realize some property of e such as this one:



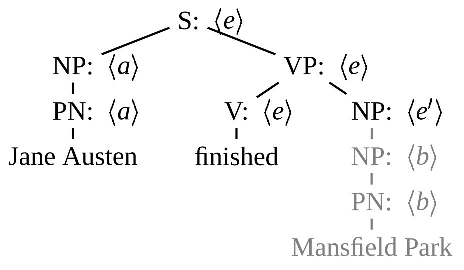
S: $\{\mathit{finish}(e,a,e'),\ \mathit{named}(a, \text{'Jane Austen'})\}$

It has a open substitution node which requires an NP referring to entity e'. In contrast to the previous example, however, producing a noun phrase like *the writing* or *the construction* is not the only possibility at this point, since this tree also licenses the substitution of a TC operator tree. Substituting such a tree, gives us the following:



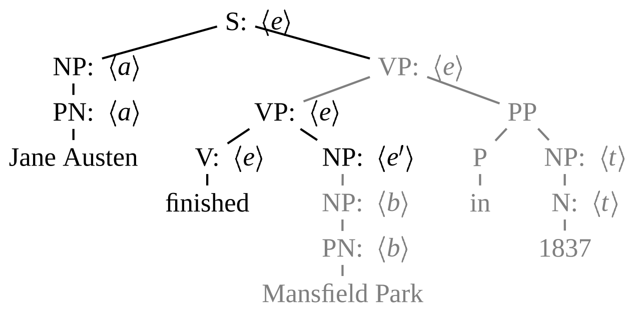
S: $\{\mathit{finish}(e,a,e'),\ \mathit{named}(a,\text{‘}\mathit{Jane\ Austen}\text{’}),\ \mathit{write}(e',a,b)\}$
P: $\mathit{Sem} \rightarrow \mathit{book}(b) \wedge \mathit{author}(a)$

The open substitution node, requiring an NP referring to entity b can then be filled as follows.



S: $\{\mathit{finish}(e,a,e'),\ \mathit{named}(a,\text{‘}\mathit{Jane\ Austen}\text{’}),\ \mathit{write}(e',a,b),$
$\mathit{mansfield\_park}(b)\}$
P: $\mathit{Sem} \rightarrow \mathit{book}(b) \wedge \mathit{author}(a)$

Now, we have a syntactically complete tree. The pragmatic constraints are satisfied as well, since the common knowledge entails that Jane Austen is an author and Mansfield Park a book. As before, the semantics has not been conveyed completely, though. This can be fixed by adjoining the missing information:

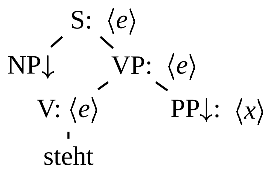


S: $\{\mathit{finish}(e,a,e'),\ \mathit{named}(a,\text{‘}\mathit{Jane\ Austen}\text{’}),\ \mathit{write}(e',a,b),$
$\mathit{mansfield\_park}(b),\ \mathit{1837}(t),\ e \subseteq t\}$
P: $\mathit{Sem} \rightarrow \mathit{book}(b) \wedge \mathit{author}(a)$

3.2.3 Yet another example

For examples like (3), the integration of TC into the generation process is based on an appropriate formulation of lexical entries for verbs that may undergo TC. We set out from the tree fragment that is introduced by *steht* ‘is standing’:

$$\begin{bmatrix} \text{idx}: & x \\ \text{shift}: & \text{to } y \text{ if } \mathit{container}(x) \wedge \mathit{contain}(e', x, y) \end{bmatrix}$$

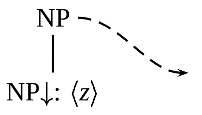


$$\text{S:} \quad \{\mathit{stand}(e, x)\}$$

Since the index of the subject NP and the first argument of the stand-relation must be co-indexed, it is not possible to model directly the intuition that TC like in (3) would have to affect the VP (rather than the subject NP; recall the discussion of these instances of TC in section 2.1). The only position in the tree where it is possible to position the TC tree fragment (by substitution) is below the NP node in the above figure. This indirect way of integrating TC into the generation process is necessitated by the structure of the SPUD system.

Formally, the lexical entry of the verb specifies this coercion potential in that it determines the ‘shift’ value of its subject NP: coercion to another NP is possible if this NP refers to the content of a container, which is the entity to which the original NP refers.

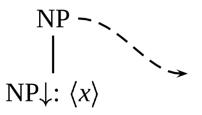
The TC operator is then modelled by the following tree fragment:



$$\begin{bmatrix} \text{idx}: & x \\ \text{shift}: & \mathit{stand}(x)(e'') \end{bmatrix}$$

$$\begin{array}{ll} \text{S:} & \{\mathit{cont}(x, z)(e')\} \\ \text{P:} & \mathit{Sem} \rightarrow \mathit{container}(x) \end{array}$$

This strategy begs the question of how we can fulfill our goal of using syntactic constituent structure to block the generation of inappropriate verbalizations of a given input (as illustrated on the example of (7) in section 2.2). The point here is that these verbalizations would have to be modelled in terms of a TC tree fragment that is inverse to the above fragment in that its top node would carry an index that refers to a substance:

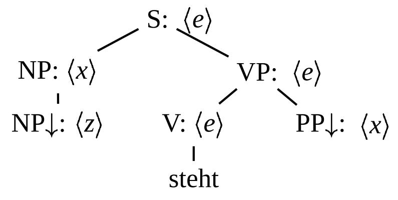


$$\begin{bmatrix} \text{idx}: & z \\ \text{shift}: & \ldots \end{bmatrix}$$

$$\begin{array}{ll} \text{S:} & \{\mathit{cont}(x, z)(e')\} \\ \text{P:} & \mathit{Sem} \rightarrow \mathit{substance}(z) \end{array}$$

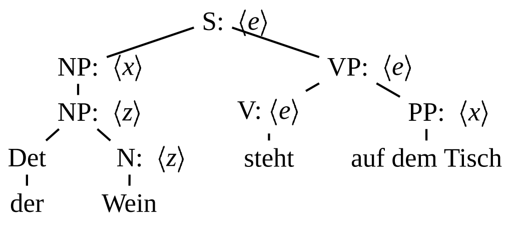
Since nominal projections inherit the index of their NP mother node, there is no way in which such an operator could be employed in order to derive the unwanted verbalization that presumes the possibility of coercing the noun. Since the synsem interface is encoded in these tree fragments, this is the way in which the interface constrains the generation of TC expressions.[2]

The result of this substitution is given in the next tree structure. Here the first argument of the stand-relation and the index of the lower subject NP node are different:



S: $\{stand(e,x), cont(e',x,z)\}$
P: $Sem \rightarrow container(x)$

Finally, after substituting the NP and PP nodes, the result is the following (somewhat simplified in the PP):



S: $\{wine(z), stand(e,x), cont(e',x,z), on\text{-}table(x)\}$
P: $Sem \rightarrow container(x)$
uniquely_identifyable(z)

This satisfies the goal of describing the eventuality *e* in a semantically complete fashion, i.e., all the material in (7) is either verbalized or can be put down to TC operators. Note also that the uniqueness condition is checked in the right way: it is uniqueness of a certain wine quantity (whose index is *z*) that is at stake in the production of (3), not of a wine container (with the index *x*).

3.2.4 A short note on where the TC operators come from

We argued that one has to take into account context dependence as well as domain and language specificity of type coercion, when looking at the phenomenon from an NLG perspective. This

[2] If generation ever gets round to dealing with quantification, it is possible to describe the fact that the operator Op applies to the verb semantics by type-raising the original operator Op:

(i) $\lambda\mathcal{P}\lambda P.\mathcal{P}(Op(P))$

The semantic representation after applying (i) to the semantics of the NP and then the result of this application to the verb semantics is one in which the NP semantics applies to an application of Op to the verb semantics, as desired.

leads to a quite fine grained description of operators, which furthermore varies with the domain of application and the language. The obvious question is: Where do these operators come from and who specifies them? One possibility is to use hand-crafted operator lexica, which could, however, be quite tedious to built. A different path that we think is worth exploring is to derive them from corpora. Lapata (2001) suggests techniques for extracting operators that account for the leeway in interpreting adjectives like *fast* (compare e.g. *a fast plane* and *a fast typist*) from corpora. It is straightforward to transfer this approach to the *begin*/*finish* type cases of metonymy.

4 Conclusion and outlook

Context dependence and domain and language specificity restrict the possibilities of type coercion. When looking at type coercion from a natural language generation point of view, this cannot be ignored, since otherwise the system would overgenerate and apply type coercion in situations where it is not appropriate. We have presented an approach to generating coercing expressions that deals with this problem by using highly specialized triggers depending on syntactic, semantic and contextual information for type coercion.

In actual implemented systems, this will only work if these triggers can be produced efficiently for different domains and languages. We think that deriving them from corpora is a promising strategy. Another open point are the factors that play a role in determining type coercion. As Example 12 in Section 2.2 shows, much more sophisticated reasoning on the context is necessary to fully capture the phenomenon. Our system provides an interface to reasoning tools and therefore in principle allows for this. However, it is not clear, yet, what the deciding factors exactly look like and how they can be captured formally.

References

Copestake, A., D. Flickinger, and I. Sag (1999). Minimal recursion semantics: An introduction. Draft. Available at `http://www-csli.stanford.edu/~aac/papers/newmrs.ps`.

de Swart, H. (1998). Aspect shift and coercion. *Natural Language and Linguistic Theory 16*, 347–385.

Dölling, J. (1992). Flexible Interpretation durch Sortenverschiebung. In I. Zimmermann and A. Strigin (Eds.), *Fügungspotenzen*, pp. 23–62. Berlin: Akademie-Verlag.

Dölling, J. (1995). Ontological domains, semantic sorts and systematic ambiguity. *International Journal of Human-Computer Studies 43*, 785–807.

Egg, M. (2000). *Flexible semantic construction: the case of reinterpretation*. Habilitation thesis, Universität des Saarlandes.

Gardent, C. and K. Striegnitz (2001). Generating Indirect Anaphora. In *Proceedings of IWCS-4*.

Grice, P. (1975). Logic and conversation. In P. Cole and J. Morgan (Eds.), *Syntax and semantics 3: Speech acts*, pp. 41–58. New York: Academic Press.

Hobbs, J. (1985). Ontological promiscuity. In *Proceedings of ACL*, pp. 61–69.

Horacek, H. (1996). On expressing metonymic relations in multiple languages. *Machine Translation 11*, 109–158.

Joshi, A. and Y. Schabes (1997). Tree-Adjoining Grammars. In G. Rozenberg and A. Salomaa (Eds.), *Handbook of Formal Languages*, Chapter 2, pp. 69–123. Berlin: Springer-Verlag.

Lapata, M. (2001). A Corpus-based Account of Regular Polysemy: The Case of Context-sensitive Adjectives. In *Proceedings of NAACL*.

Moens, M. and M. Steedman (1988). Temporal ontology and temporal reference. *Computational Linguistics 14*, 3–14.

Nunberg, G. (1979). The non-uniqueness of semantic solutions: polysemy. *Linguistics & Philosophy 3*, 143–184.

Piñango, M. M. (this volume). Event structure at the syntax-semantics interface: processing and neurological properties.

Pulman, S. (1997). Aspectual shift as type coercion. *Transactions of the Philological Society 95*, 279–317.

Pustejovsky, J. (1995). *The generative lexicon*. Cambridge: MIT Press.

Rambow, O. and D. Hardt (2001). Generation of VP Ellipsis: a Corpus-based Approach. In *Proceedings of ACL*.

Shaw, J. (1998). Segregatory Coordination and Ellipsis in Text Generation. In *Proceedings of Coling/ACL*, pp. 1220–1226.

Stone, M. and C. Doran (1997). Sentence planning as description using tree adjoining grammar. In *Proceedings of ACL*, pp. 198–205.

Stone, M., C. Doran, B. Webber, T. Bleam, and M. Palmer (2001). Microplanning with Communicative Intentions: The SPUD System. Avvailable at `http://arXiv.org/abs/cs/0104022`. Submitted to Computational Intelligence.

Striegnitz, K. (2001.). Pragmatic Constraints and Contextual Reasoning in a Natural Language Generation System: a system description. Draft. Available at `http://www.coli.uni-sb.de/cl/projects/indigen/publications.html`.