Aggregation with Strong Regularities and Alternatives

Helmut Horacek
Universität des Saarlandes
FB 6.2 Informatik
horacek@cs.uni-sb.de

Abstract

Aggregation is typically treated in NLG as a local optimization measure, and methods exist only for building conjoined expressions with 'and'. In contrast to that, solutions to logical problems are characterized by regularly occurring commonalities, including complete subsets of possible value combinations and alternatives. In order to address constellations of this kind, we extend current aggregation techniques, envisioning high degrees of condensation. In particular, we define novel constructs that can express sets of propositions with highly regular variations on slot values concisely, including special forms of disjunctions. Our methods enable the generation of expressions with semantically complex operators, such as 'vice-versa' and 'each', and they support various aspects in interpreting solutions produced by formal systems, such as highlighting commonalities among and differences across solution parts, supporting the inspection of dependencies and variations, and the discovery of flaws in problem specifications.

1 Introduction

Aggregation is a central concept in NLG that is relevant for practically all applications. It is typically treated as an opportunistic optimization measure addressing locally occurring commonalities. In contrast to that, solutions to logical problems are characterized by regularly occurring commonalities, including complete subsets of value combinations and alternatives. In order to address constellations of this kind, we extend current aggregation techniques, envisioning high degrees of condensation. In particular, we define novel constructs that can express sets of propositions with highly regular variations on slot values concisely, including special forms of disjunctions. Among others, this method enables the generation of expressions with semantically complex operators and disambiguation markers, such as 'vice-versa', 'each', 'remaining', and 'distinct'.

The paper is organized as follows. First we review aggregation techniques. Then we define new aggregation constructs. We describe a procedure for building compositions of these constructs, including syntactic realization. Finally, we discuss the application potential of our approach.

2 Previous Approaches

The term *aggregation* was first used in (Mann and Moore, 1980). It is relevant in all processing phases of NLG (Reape and Mellish, 1999). Its most common form, structural aggregation, concerns compositions of several logical assertions that *share information* into a single natural language utterance with coordinated or omitted parts, yielding *conjunction reduction* ("*Foxes* and *wolves* are animals") and *ellipsis* or *gapping* ("*Foxes* are larger than birds and _ smaller than wolves").

Aggregation techniques have been incorporated in early systems, such as KDS (Mann and Moore, 1981). Aggregation manifests itself in optimizations carried out at the discourse level (Horacek, 1992) and domain specific orderings of propositions prior to condensation (Dalianis, 1999; Dalianis and Hovy, 1996). Coordination and lexical aggregation take place in sentence planning, emphasizing stylistic preference rules (Scott and de

be(Pete,boxer,<2,4,5,7,9-11,13-16>)	be(Steve,actor,<3,5,13>)	be(Thelma,actor,<4,8-10,15>)
be(Pete,guard,<1,3,6,8,12,15,16>)	be(Steve,boxer,<8,12>)	be(Thelma,boxer,<1,3,6>)
be(Pete,operator,<1-14>)	be(Steve,guard,<9,14>)	be(Thelma,chef,<1-16>)
be(Roberta,actor,<1,2,6,7,11,12,14,16>)	be(Steve,nurse,<1-16>)	be(Thelma,guard,<2,5,7>)
be(Roberta,guard,<4,10,11,13>)	be(Steve,officer,<6,7,10,11>)	be(Thelma,teacher,<11-14,16>)
be(Roberta,officer,<1-5,8,9,12-16>)	be(Steve,operator,<15,16>)	
be(Roberta,teacher,<3,5-10,15>)	be(Steve,teacher,<1,2,4>)	

Figure 1. Solutions to the puzzle – alternatives in assigning persons to jobs

Souza, 1990) and hypotactic aggregation with detailed lexical decisions (Robin, 1995). However, all these approaches operate on some local level.

The only systematic approach, which has also inspired our method, is Shaw's staged process (1998a; 1998b), which separates heuristic orderings followed by recurrence markings from linguistically justified sentence boundary decisions and reduction operations.

3 The Running Example

We present solutions to a *puzzle*: "There are four people: Roberta, Thelma, Steve, and Pete. Among them they hold eight different jobs, each exactly two. The jobs are: chef, guard, nurse, telephone operator, police officer, teacher, actor, and boxer. The job of the nurse is held by a male. The husband of the chef is the telephone operator. Roberta is not a boxer. Pete has no education past the ninth grade. Roberta, the chef, and the police officer went golfing together. Who holds which jobs?"

This is a fully regular assignment problem. 16 solution variants consist of eight assignments each, which amounts to 128 propositions if unaggregated. In Figure 1, the solution variants are referred to by numbers, as an extra slot. All propositions with the same value of this slot must be true at the same time, those with different values holding alternatively. To save space, variant numbers for the same fact are enclosed between '<' and '>'.

4 New Aggregation Constructs

The solutions to this puzzle consist of highly regular assignments, which is typical for machine-generated solutions to these kind of problems (consider, for example, solutions produced by the model generator KIMBA (Konrad and Wolfram, 1999), presented in (Horacek and Konrad, 1999)). For expressing these constellations concisely and elegantly in natural language, semantically complex operators, such as 'each' and 'vice-versa' prove useful. For generating these expressions, we define novel *"2-dimensional"* coordinations. Normally, *coordination* is done in a *pairwise* fashion. For the new constructs, aggregation is done by building the *cross product* of the values of two slots, including special forms of disjunctions for commonalities across variants (*Choice*, *Except*, and *Assign*):

- The *Permut* construct
  It expresses a set of predications in which the values of two slots comprise all combinations out of the two sets of slot fillers, within the *same* variant. An example sentence is "*Each* of Pete, Steve, and Thelma can be boxer and guard," expressing six of the facts in Figure 1, when abstracting from the variant slot.

- The *Choice* and the *Except* constructs
  It expresses an assignment of one individual to several others, each in a *different* variant. An example sentence is "Thelma holds *one of* the jobs actor, guard, and teacher," comprising job assignments of Thelma other than that to chef in variants 7, 10, and 11 (see Figure 1). An *Except* construct comprises the complementing assignments for each variant of the *Choice* construct, for another individual. An example sentence is "and Roberta holds the *remaining* positions," complementing the above job assignments of Thelma.

(1) base(P,a_1,…,a_n) ::= P(a_1,…,a_n)
(2) $\lambda x_i,x_j$. base(…,x_i,…,x_j,…) (a_i) (a_j) ::= base(…,a_i,…,a_j,…)
(3) <A_{in}> ::= <a_{i1},…,a_{in}>
(4) *Coord*(base(…,<A_{in}>,…,<A_{jn}>,…)) ::= $\forall k(1\le k\le n)$: $\lambda x_i,x_j$. base(…,x_i,…,x_j,…) (a_{ik}) (a_{jk})
(5) *Permut*(base(…,<A_{in}[/a_{id}]>,…,<A_{jm}[/a_{je}]>,…,a_c)) ::=
$\forall k,l(1\le k\le n,1\le l\le m;\ (k,l)\ne(d,e))$: $\lambda x_i,x_j$. base(…,x_i,…,x_j,…,a_c) (a_{ik}) (a_{jl})
(6) *Choice*(base(…,a,…,<A_{in}>,…,<A_{jn}>)) ::= $\forall l(1\le k\le n)$: $\lambda x_i,x_j$. base(…,a,…,x_i,…,x_j) (a_{ik}) (a_{jl})
(7) *Except*(base(…,a,…,<A_{in}>,…,<A_{jn}>)) ::= $\forall k,l(1\le k,l\le n,k\ne l)$: $\lambda x_i,x_j$. base(…,a,…,x_i,…,x_j) (a_{ik}) (a_{jl})
(8) *Assign*(base(…,<A_{in}[/a_{id}]>,…,<A_{jn}[/a_{je}]>,…,<A_{kn}>)) ::=
$\forall l,m(1\le l,m\le n;\ (l,m)\ne(d,e))\ \forall p(1\le p\le n!)$: *Choice*(base(…,<$A_{il}$>,…,<$A_{jm}$>,…,<$A_{kp}$>))
where jm = f(il,kp) with f(x,z) ≠ f(y,z) (mutually exclusive)

Figure 2. Formal definitions of aggregation constructs

- The *Assign* construct
  It comprises a special set of *Choice* constructs which express the set of all bijective functions between two sets of individuals, each in one variant, without repetitions. An example is "Roberta, Steve, and Thelma *each* hold *one distinct* job *out of the set* actor, guard, and teacher," expressing their job assignments in variants 2, 4, 5, 9, 13, and 14 (see Figure 1).

In order to define these constructs, we apply λ-expressions with some special notations (see Figure 2). Assertions can be reexpressed with the predicate as an additional slot (1), to allow the building of λ-expressions (2), that is, the extraction of variables bound by λ, for which constants can be substituted. Moreover, slots can be filled by lists, expressed by capital letters. The first index indicates the slot position, and the second the number of elements in the list (3). Moreover, square brackets indicate optionality, and a dash marks an element as an exception. Based on that, operators are used to express the way composition is done in each case, defined by implicitly conjoined λ-expressions. The *Coord* operator handles pairwise coordination where the substitutions run over the same index, defined for two slots in (4). The *Permut* operator is similar, but both indexes are varied separately, within one variant a_c (5). In addition, one specific combination can be excluded optionally (the pair a_{id} and a_{je}). The *Choice* operator is identical to *Coord*, but for the fact that the variant slot is aggregated (6); for *Choice*, however, aggregated assertions hold *alternatively* and not *simultaneously*. The complementing *Except* operator (7) is defined analoguously. The *Assign* operator is defined as a specific composition of *Choice* operators, with one assignment optionally excluded (8). The variant numbers expressing alternatives are included as an extra slot in the prominent last position, in order to keep the aggregation procedures uniform. Examples for these constructs are given in Figure 3.

In order for these constructs to be meaningfully applicable, three conditions about the corresponding lexical expressions must hold in addition

(9) *Coord*(be(<Steve,Thelma>,<nurse,chef>,all))
(10) *Permut*(be(<Pete,Steve,Thelma>,<boxer,guard>)) (abstracting from variants)
(11) *Choice*(be(Thelma,<actor,guard,teacher>,<10,7,11>))
(12) *Except*(be(Roberta,<actor,guard,teacher>,<10,7,11>))
(13) *Assign*(be(<Roberta,Steve,Thelma>,<actor,guard,teacher>,<2,4,5,9,13,14>))

Figure 3. Aggregation constructs built from the running example

to the proper constellation of regularities (for *Coord* constructs, the second condition is sufficient): (1) slot values aggregated in a cross product fashion must be expressed by NPs, (2) the sentence predicate must not allow collective readings, and (3) the sentence pattern must make reference to the object's category (e.g, "holding a *position*"), to anchor disambiguation markers (e.g., "distinct"). These requirements are special forms of expressibility conditions, and they are tested by advance lexicon look-ups. Note that these expressions are underspecified in terms of scoping. This representation is justified by the fact that all scopings conform with the above conditions are semantically equivalent due to the explicit enumeration of slot fillers and the restriction to distributive reading (only the variant slot has a specific semantic).

5 A Procedure for Aggregation

In order to achieve maximal condensation, which is the intuitively most plausible optimization criterion, investigating a considerable search effort would be required. The results are very sensitive to the ordering imposed on the facts to be aggregated. Therefore, all possible orderings need to be tested to find the maximally condensed text, with reordering applied to aggregated structures for obtaining further condensation. To achieve a balance between optimality and efficiency, we confine ourselves to a linear procedure. Hence, aggregation is performed in a staged process with internal recursions: (5.1) Intermediate structures with single coordinations are built. (5.2) For disjunctions, all variants are expressed compactly and unambiguously, if possible, otherwise (5.3) the set of variants is split, repeating step (5.2) for each subset. (5.4) Multiple coordinations are built out of intermediate structures. (5.5) Linguistic realization is carried out.

5.1 Coordination with a Single Difference

First, propositions are ordered by the following heuristics: (1) The variant slot is the last slot sorted, to support efficient aggregation across variants. (2) The remaining slots are sorted by starting from the element with the *least* number of distinct values, which tends to keep together "regular" substructures, contrasting to the heuristic applied by Shaw. *Coord* constructs are built across propositions that differ in a single slot value (*one-slot distinct*). This is done by testing the first two propositions for identity in all but one of their slots, building a *Coord* construct with a list for the slot with non-equal values. This is repeated incrementally for the next pair of adjacent propositions, or with the *Coord* construct just built and the proposition following. If a set contains all individuals in the given context, it is replaced by 'all'. In the puzzle, condensation is done across variants, yielding the propositions in Figure 1 within *Coord* operators.

5.2 Condensing Disjunctions

In order to express a set of facts holding in several variants concisely, coordination constructs must be built that convey all dependencies among these variants unambiguously. This means that a subset of the propositions must be condensed into one construct expressing disjunctions, with the remaining propositions being independent of the set of variants considered – otherwise, the set of variants is split and condensation attempts are made for each subset separately. In order to test whether this is possible, *Choice* constructs are built by condensing adjacent propositions that differ by their variant slot only. Three constellations are possible (examples appear under variant subsets in which the set of propositions is partitioned and marked as cases in Figure 4):

- There are two *Choice* constructs, *Choice*(x, $\langle a_1,a_2\rangle$,$\langle i_1,i_2\rangle$) and *Choice*(y,$\langle a_2,a_1\rangle$, $\langle i_1,i_2\rangle$). They are condensed into an *Assign* construct of the form *Assign*($\langle x,y\rangle$,$\langle a_1,a_2\rangle$,$\langle i_1,i_2\rangle$). In the puzzle, this step applies to assigning 'Roberta' and 'Thelma' and 'actor' and 'teacher', variants 15 and 16 (case 1).

- There is one *Choice* construct, *Choice*(x,$\langle A_n\rangle$, $\langle I_n\rangle$). If no further propositions are left, or there are n *Coord* constructs *Coord*(y,a_i,$\langle I_m\rangle$), where I_m is equal to I_n with i_i missing, then these *Coord* constructs are condensed into an *Except* construct *Except*(y,$\langle A_n\rangle$,$\langle I_n\rangle$). In the puzzle, this measure applies to assigning 'Thelma' to 'actor', 'guard', and 'teacher' in variants 7, 10, and 11, and 'Roberta' to the remaining positions in each variant (case 2.1.2).

Coord(be(<Steve,Thelma>,<chef,nurse>))
case 1 (variants are <15,16>): be(Steve,operator)
Coord(be(<Pete,Pete,Roberta>,<boxer,guard,officer>))
Assign(be(<Roberta,Thelma>,<actor,teacher>))
case 2 (variants are <1-14>): be(Pete,operator)
case 2.1 (variants are <6,7,10,11>): be(Steve,officer)
case 2.1.1 (variant is 6): be(Thelma,boxer)
Coord(be(<Pete,Roberta,Roberta>, <guard,teacher,actor>))
case 2.1.2 (variants are <7,10,11>): be(Pete,boxer):
Choice(be(Thelma,<actor,guard,teacher>))
Except(be(Roberta,<actor,guard,teacher>))
case 2.2 (variants are <1-5,8,9,12-14>): be(Roberta,officer)
case 2.2.1 (variants are <1,3,8,12>): be(Pete,guard)
Assign(be(<actor,boxer,teacher/boxer>,<Roberta,Steve,Thelma/Roberta>))
case 2.2.2 (variants are <2,4,5,9,13,14>):be(Pete,boxer)
Assign(be(<actor,guard,teacher>,<Roberta,Steve,Thelma>))

Figure 4. Aggregation constructs for the puzzle (with variants slots omitted, corresponding to cases)

- There is no *Choice* construct. In this case, a specific set of *Coord* constructs must be present so that they can be condensed into an *Assign* construct. This is an effective way to *compose* an *Assign* construct, whereas the one in Figure 2 is more convenient for *defining* it. To start with, there must be exactly $n!$ or $n!-(n-1)!$ variants for some n, and exactly n individuals in the slots varied (without the variant slot) in n^2 or n^2-1 *Coord* constructs. With indexing over individuals, checking the conditions is cheap. Only in case of success, expensive conditions for building an *Assign* construct must be checked. For each i of n (or n-1) *Coord* constructs *Coord*(x,a_i,<I_m>), the sets of variants must be pairwise disjoint, which means that each x is assigned to exactly one individual in each variant. Similarly, in each i of the n (or n-1) *Coord* constructs *Coord*(x_i,a,<I_m>), the sets of variants must be pairwise disjoint, too, so that each x_i is assigned differently in each variant. If all this is fulfilled, an *Assign* construct comprising all these individuals is built. If the number of intermediate *Coord* constructs is $n!-(n-1)!$ and not $n!$, the missing fact must be incorporated in the *Assign* construct as an exception. In the puzzle, such a constellation occurs twice. In variants 1, 3, 8, and 12 (4 = 3!-2! variants), all combinations assigning 'Roberta', 'Steve', and 'Thelma' to 'actor', 'boxer', and 'teacher' occur, except that 'Thelma' is not assigned to 'boxer' (case 2.2.1). Similarly, variants 2, 4, 5, 9, 13, and 14 (6 = 3! variants) contain all combinations assigning 'Roberta', 'Steve', and 'Thelma' to 'actor', 'guard', and 'teacher' (case 2.2.2).

5.3 Splitting Sets of Variants

After *Coord* constructs that hold across all variants are extracted, splitting is done along a set of predications that are *two-slot distinct* from one another and appear in each variant exactly once. Due to the ordering imposed, finding such a set can be done locally. If there is none, no splitting is attempted, since it might appear unmotivated and computationally expensive – each variant is then presented separately. In case of several candidates, the selection is oriented on economy criteria, preferring smaller sets of facts, and fewer splits of facts aggregated over variants. Moreover, uneven distributions of variants are favored, since smaller parts are more likely to contain regular substructures.

In the puzzle, the propositions holding in all variants are extracted first, yielding the “fixed” jobs 'chef' and 'nurse'. For the first splitting into

Assign (general form)	<MARK(POSITION)> <LIST(SLOT1)> "each" <P-CAT(CAT)> "one distinct" <N-CAT (CAT)> "out of the set of" <LIST(SLOT2)> [<EXCEPTION(TOP)>]
Assign (for list length 2)	<COORD (TOP)> "or vice-versa"

Figure 5: Example TG/2 patterns, two variants of expressing *Assign* constructs

cases 1 and 2, 'operator' and 'officer' yield binary partitions, 'operator' causing a more uneven partitioning. Case 2 is split again, according to the assignment variants for 'officer'. Its first partition, case 2.1, needs to be split further. All candidate jobs except 'guard' yield binary splits, and 'boxer' is chosen because this requires only two aggregated facts to be split instead of three for 'actor' and 'teacher'. The other partition resulting after the second split, case 2.2, is split again, the only binary choice being the jobs taken by Pete.

5.4 Coordination with Two Distinct Slots

Within subsets of variants, it is attempted to compose facts that hold across all of these variants into larger structures, preferably *Permut* constructs or at least *Coord* constructs with *two-slot distinct* facts. Building *Permut* constructs is done by composing *Coord* constructs that have identical lists $<A_{in}>$ in position *i*, non-equal atomic values a_{jk} in position *j (k* indexing *m Coord* constructs), and identical atomic values in other positions, to yield *Permut*(base(…,$<A_{in}>$,…,$<A_{jm}>$,…)). At most once, one value a_x from $<A_{in}>$ may be missing yielding *Permut*(base(…,$<A_{in}[/a_x]>$,…,$<A_{jm}[/a_y]>$, …)) for a *Coord* construct with a_y in position *j*. For the remaining facts and *one-slot distinct Coord* constructs, condensation into *two-slot distinct Coord* constructs is attempted. This is done similarly to the building of *one-slot distinct Coord* constructs, and lists are built in the positions of two slots with distinct values. If a *one-slot distinct Coord* construct is incorporated, the atomic value in the other slot is copied as many times as the list in the slot with distinct values has elements. Repetitions in one of the slots lead to *mixed* coordinations in the linguistic realization, where coordinated conjunctions are combined with predicate gapping. This advance over previous approaches is due to our staged coordination procedure. However, the disadvantage is the restriction to *two-slot distinct* coordinations (Shaw handles multiple distinctions).

5.5 Linguistic Realization

In order to express the aggregation constructs by natural language text, these specifications are processed by the linguistic realization component TG/2 (Busemann, 1996). TG/2 is a pragmatically motivated system that has been used for several kinds of applications, including multi-lingual reports of air pollution data (Busemann and Horacek, 1996). Its most unique feature is its capability to process specifications from several levels of linguistic elaboration, integrating canned text, template techniques and context-free grammars into a single formalism. This enables a user of TG/2 to express specifications only as detailed as needed to handle the distinctions required for the application at hand. This feature allows us to model linguistically complex coordination phenomena by defining simplified grammar rules for schematic sentence patterns (unlike Shaw's repertoire of linguistic constructs).

The proper application of TG/2 is preceeded by a conversion procedure that reexpresses the position-based encodings in terms of predicate-argument structures. In addition, specifications with case splits are reorganized. Blocks with predicates of one category or propositions within cases that result from splitting are headed by an introductory statement about their commonality, and markers expressing, for example, positions in enumerations, are added. Moreover, nested case structures are converted into lists where each case is labeled explicitly with its distinctive feature, except subcases of minimal size (one sentence).

Subsequently, TG/2 maps aggregation constructs via rules onto sentence patterns, such as those listed in Figure 5. For the first variant of the *Assign* construct, the pattern accesses substructures of that construct (POSITION, SLOT1, CAT,

Thelma is the chef and Steve the nurse.
Then we have three cases to consider.
Case 1: Steve is the operator.
Then Pete is the boxer and the guard, and Roberta the officer.
Moreover, Roberta is the actor and Thelma the teacher, or vice-versa.
Case 2: Pete is the operator and Steve the officer.
One alternative here is that Thelma is the boxer.
Then Roberta is the teacher and the actor, and Peter the guard.
The other alternative is that Pete is the boxer.
Then Thelma takes one of the positions teacher, guard, and actor, and Roberta the remaining positions.
Case 3: Pete is the operator and Roberta the officer.
One alternative here is that Pete is the guard.
Then Roberta, Thelma, and Steve each take one distinct position out of the set teacher, actor, and boxer, but Roberta is not the boxer.
The other alternative is that Pete is the boxer.
Then Roberta, Thelma, and Steve each take one distinct position out of the set guard, teacher, and actor.

Figure 6: Natural language text for the puzzle

SLOT2), and the whole structure (TOP), and it has further references to rules for a discourse marker (MARK), lists (LIST), a function verb for the predicate (P-CAT), the associated noun (N-CAT), and an optional exception (EXCEPTION). The second variant, which is treated as a *Coord* construct followed by a canned text portion, is only applicable to lists of length 2 without an exception. There are two variants for expressing *Except* constructs, one using “remaining” when following a suitable *Choice* construct (see Figure 6, Case 2, fourth sentence). 'Mixed' coordinations may require local reordering, to group together sublists (see Figure 6, Case 1, first sentence). When applied to the subject, the predicate must be repeated when plural changes to singular or vice-versa.

6 Conclusion and Discussion

In this paper, we have developed extensions to existing aggregation techniques that address constellations with regularly occurring commonalities, as typically found in problem solutions produced by formal systems. We have defined novel constructs that can express sets of propositions with highly regular variations on slot values concisely, including special forms of disjunctions. Moreover, we have described a systematic process in which compositions of these constructs are built.

Our methods are particularly useful for presenting results produced by formal systems, where the problem definitions justify expectations about highly regular structures in the associated results. Moreover, predications typically have few slots only, to support efficient reasoning. In such settings, our methods are essential and effective, and they enable the generation of expressions with semantically complex operators, such as 'vice-versa' and 'each', and coordinated conjunctions combined with predicate gapping. Apart from mere conciseness, the presentations obtained highlight commonalities among and differences across parts of problem solutions in a much better way than previous approaches do, thereby supporting the inspection of dependencies and variations, and the discovery of flaws in problem specifications, as shown in (Horacek and Konrad, 1999). For constellations with less regular value combinations and more predicate slots (including, e.g., temporal and local information), the procedure proposed by Shaw is more appropriate. It is procedurally simpler and has a richer repertoire in expressing linguistic phenomena. In principle, our operators could also be applied beneficially for ordinary text generation, especially those operators expressing

alternatives compactly. The major problem, however, would be their effective integration into the overall generation process, since testing their applicability conditions might be expensive and not easy to manage.

Finally, we have limited the use of the new operators to simple cases, without interference with other constructs. For example, our procedures are set up in such a way that a *Choice* operator is built only if all its assignments are distinct from one another. However, there may be constellations where a distinction between alternatives is made elsewhere, and the use of a *Choice* operator would still be possible and effective, as in "Thelma takes one of the positions teacher, guard, and actor, and Roberta is the teacher or the actor if Thelma is the guard". This constellation comprises four alternatives, with Thelma being the guard in two of them. Despite the extra condition involved, we believe that the aggregated form above conveys the underlying situation in a more natural and better understandable way than a mere enumeration.

In the future, we intend to investigate practical demands for extended uses of the new operators. Moreover, we will examine the adequacy of the pattern-based approach for expressing these operators in lexical terms by considering non-European languages, such as Arabic. Finally, we extend the use of these operators to formulas (Horacek, 2002).

References

Busemann, S. 1996. Best-First Generation. In Proc. of the *8th International Workshop on Natural Language Generation*, pp. 101-110, Hearstmonceaux, UK.

Busemann, S. and Horacek, H. 1998. A Flexible Shallow Approach to Text Geneartion. In Proc. of the *9th International Workshop on Natural Language Generation*, pp. 238-247, Niagara-on-the-Lake, Canada.

Dalianis, H. 1999. Aggregation in Natural Language Generation. *Computational Intelligence* 15(4).

Dalianis, H.; and Hovy, E. 1996. Aggregation in Natural Language Generation. In G. Adorni, M. Zock (eds.), Trends in Natural Language Generation – An Artificial Intelligence Perspective, pp. 88-105, Springer.

Horacek, H. 1992. An Integrated View of Text Planning. In R. Dale, E. Hovy, D. Rösner, O. Stock (eds.), Aspects of Automated Natural Language Generation, pp. 29-44, Springer.

Horacek, H. 2002. Presenting Sets of Problem Solutions Concisely. Submitted.

Horacek, H.; and Konrad, K. 1999. Presenting Herbrand Models with Linguistically Motivated Techniques. In Proc. of CIMCA-99, Vienna, Austria.

Konrad, K.; and Wolfram, D. 1999. System Description: Kimba, A Model Generator for Many-Valued First-Order Logics. In Proc. of the *16th International Conference on Automated Deduction* (CADE-16), pp. 282-286 Trento, Italy.

Mann, B.; and Moore, J. 1980. Computer as Author – Results and Prospects. Research Report ISI/ RR-79-82, University of Southern California, Information Sciences Institute, Marina del Rey.

Mann, B.; and Moore, J. 1981. Computer Generation of Multiparagraph English Text. *American Journal of Computational Linguistics* 8(2), pp. 17-29.

Reape, M.; and Mellish, C. 1999. What is Aggregation Anyhow? In Proc. of *7th European Workshop on Natural Language Generation*, pp. 20-29, Toulouse, France.

Robin, J. 1995. *Revision-Based Generation of Natural Language Summaries Providing Historical Background*. PhD thesis, Columbia University.

Scott, D.; and de Souza C. 1990. Getting the Message Across in RST-Based Text Generation. In R. Dale, C. Mellish, M. Zock (eds.), Current Research in Natural Language Generation, pp. 47-73, Academic Press, New York.

Shaw, J. 1998a. Segregatory Coordination and Ellipsis in Text Generation. In Proc. of the *36th Association for Computational Linguistics and the 17th International Conference on Computational Linguistics*, pp. 1220-1226, Montreal, Canada.

Shaw, J. 1998b. Clause Aggregation Using Linguistic Knowledge. In Proc. of the *9th International Workshop on Natural Language Generation*, pp. 18-147, Niagara-on-the-lake, Canada.