Einführung in die Pragmatik und Texttheorie
Summer Semester 2004

Generating Referring Expressions (Dale & Reiter 1995)

Ivana Kruijff-Korbayová

(based on slides by Gardent&Webber, and Stone&van Deemter)

Outline

- The GRE problem
- Interpretation of Gricean Maxims for GRE
- GRE algorithms (Dale&Reiter 1995)
  - Full Brevity
  - Greedy Heuristic
  - Local Brevity
  - Incremental Algorithm
- Limitations and extensions/modifications of the Incremental Algorithm

The GRE Problem

- Referential goal = identify an entity
- How to do that?
  - Generate a distinguishing description, i.e., a description that uniquely identifies the entity
    - If the entity has a familiar name which refers uniquely, the name is enough.
    - However, many entities do not have names.
  - Avoid false implicatures
  - Adequacy and efficiency

GRE and Conversational Maxims

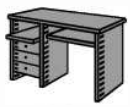
- Quality:
  - RE must be an accurate description (properties true of entity)
- Quantity:
  - RE should contain enough information to distinguish the entity from other entities in the context, but not more
- Relevance
  - RE should mention attributes that have discriminatory power
  - „relevant attributes“
- Manner
  - RE should be comprehensible and brief
- Violation of a maxim leads to implicatures, e.g.,
  - ‘the mean pitbull’ (when there is only one salient dog).
  - ‘the cordless drill that’s in the toolbox’

The GRE Problem

- Terminology:
  - Intended entity
  - Context set of (salient) entities
  - Contrast set of (salient) entities (= set of distractors)
  - Properties true of the intended entity
- Distinguishing description:
  - All properties included in the description are true of the intended entity.
  - For every entity in the contrast set, there is a property in the description that does not hold of that entity.

The GRE Problem: Example

Context set:



a: <chair, cheap, heavy> *b*: <chair, expensive, light> *c*: <desk, cheap, heavy>

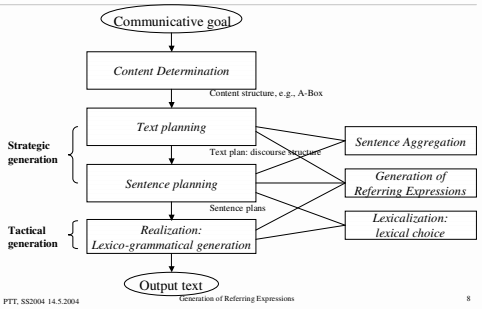
Goal: Generate a distinguishing description for *a*

–Contrast set (set of distractors):	{*b*,*c*}
–Properties true of the entity:	{chair, cheap, heavy}
–A distinguishing description:	{chair, heavy} or {chair,cheap}

The GRE Problem

- GRE tries to find "the best" distinguishing description
- GRE is a microcosm of NLG: e.g., determines
  - which properties to express (Content Determination)
  - which syntactic configuration to use (Syntactic Realization)
  - which words to choose (Lexical Choice)
- How to do it computationally efficiently?

A reference architecture for NLG



GRE as a Set Cover Problem

- Finding a distinguishing description for an entity is essentially equivalent to solving the set cover problem
  - For a property p, RuleOut(p) is a subset of the contrast set C that is ruled out by p, i.e., the set of entities for which p does not hold
  - D is a distinguishing description if the union of RuleOut(d) over all d in D equals C, i.e., D specifies a set of RuleOut sets that together cover all of C
- Thus, algorithms and complexity results for the set cover problem can be used for the GRE problem.
  - Finding optimal set cover (= min size; shortest description) is NP-hard
  - The greedy heuristic algorithm finds a close to min set cover and is polynomial.
  - Dale&Reiter (1995) explore the application of these results to GRE and discuss cognitive plausibility for a variety of algorithms

GRE Algorithms

- Computational interpretations of the requirements reflecting the Gricean Maxims:
  - **Full Brevity** (find the shortest possible DD) NP-hard, worst case complexity exponential in no. of properties
  - **Greedy Heuristic** (variant of Johnson's GH for min set cover) polynominal
  - **Local Brevity** (iterative shortening of an initial DD) polynomial
- Dale&Reiter 1995:
  - **Incremental algorithm** (sequential iteration through an ordered list of attributes) polynomial

Full Brevity

- (Dale 1989, 1992) proposed an algorithm that complies with a very strict interpretation of the Maxims
- It attempts to generate the shortest possible DD through breadth-first search (thus, NP-hard because looking for minimal set cover):
  - Check whether any 1-component DD is successful
  - Check whether any 2-component DD is successful
  - Etc.

  Until success = minimal DD is generated or failure = no description
- In worst case, needs to examine all combinations of properties
- It is possible that algorithms exist which have acceptable performance in "realistic cases" (but would need to be able to discriminate between circumstances when the algorithm can and cannot be applied)

Greedy Heuristic

- (Dale 1989, 1992) proposed an algorithm that was a variant of Johnson's (1974) greedy heuristic for minimal set cover, and generates a close to minimal DD

Initialization: contrast set, empty description

Repeat:

1. Check Success
   if no more distractors, then succesfully generated DD
   else if no more properties, then fail
2. Choose property which eliminates the most distractors
3. Extend description with chosen property

Greedy Heuristic: Example

- Context
  a: <large, red, plastic>
  b: <small, red plastic>
  c: <small, red paper>
  d: <medium, red paper>
  e: <large, green, paper>
  f: <large, blue, paper>
  g: <large, blue, plastic>
- To generate a description for *a*:
  - Selected property: plastic; remaining distractors {*b,g*}
  - Selected property large (or red): remaining distractors {*g*}
  - Selected property red (or large): remaining distractors {}
- Generated description: <large, red, plastic>
- However, true minimal DD is <large, red>

Local Brevity

- (Reiter 1990) proposed an algorithm which aims to produce descriptions satisfying the following criteria:
  - No unnecessary components.
  - Local brevity: not possible to shorten description by replacing a set of existing components by a single new component.
  - Lexical preference for basic-level and other preferred words
- Iterative algorithm:
  Start with an initial description (generated by greedy heuristic)
  Repeat
  1. try to shorten
  2. if cannot shorten, exit with the current description

Incremental Algorithm

- D&R95 propose an algorithm which does not attempt to find an "optimal" combination of properties. Therefore,
  - It is faster, because it does not compare distractor sets.
  - Does not always generate the shortest possible description, i.e., sometimes produces redundant descriptions
- What it does:
  - Iterate through the list of properties in a fixed (preference) order.
  - Include a property iff it is 'useful', i.e., true of target and false of some distractors, i.e. it eliminates some remaining distractor(s).
  - Terminate and return the current description when the set of remaining distractors is empty.
  - Terminate and return nil when the current description is not empty, but there are no more properties to include.
  - No backtracking. No revision of already constructed description.

Justification for Incremental Alg.

- Previous algorithms try to produce "optimally" distinguishing descriptions, but:
- People don't speak this way
  - empirical work shows much redundancy
  - For example,
    - [Manner] 'the red chair' (when there is only one red object in the domain).
    - [Manner/Quantity] 'I broke my arm' (when I have two).
- D&R95 argue that the algorithm produces cognitively plausible descriptions
- Problem:
  - The redundant descriptions are not always produced in a controlled way, e.g., motivated by other communicative goals or for textual reasons

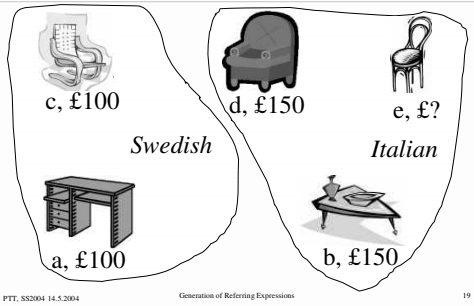
Incremental Algorithm

r = individual to be described
C = contrast set
P = list of properties, in preference order
p is a property from P
L= properties in generated description

Incremental Algorithm

$L := \{\}$
For all $p \in P$ do :
  If $r \in [[p]]$ & $C \not\subset [[p]]$ then do
    $L := L \cup [[p]]$
    $C := C \cap [[p]]$
    If $C = \{\}$ then Return L
Return nil

Example: Domain

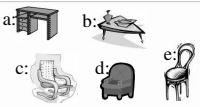


Example: Domain Formalized

- Properties: type, origin, colour, price, material
  - Type: furniture (*abcde*), desk (*ab*), chair (*cde*)
  - Origin: Sweden (*ac*), Italy (*bde*)
  - Color: dark (*ade*), light (*bc*), grey (*a*)
  - Price: 100 (*ac*), 150 (*bd*) , 250 ({})
  - Material: wood ({}), metal ({abcde}), cotton(d)
- Preference order:
  - Type > Origin > Color > Price > Material
- Assumption: all this is shared knowledge.

Incremental Algorithm: Example

furniture (abcde),
desk (ab), chair (cde),
Sweden (ac), Italy(bde),
dark (ade), light (bc), grey (a),
100£ ({ac}), 150£(bd) , 250£ ({}),
wood({}), metal (abcde), cotton ({d})



Now describe:

a = <...><desk {ab}, Sweden {ac}>
d = <...><chair,Italy,150£> (Nonmin., cf <150£,chair>)
e = <...><chair,Italy,....> (Impossible, price not known)

Incremental Algorithm

- Logical completeness: A unique description is found in finite time if there exists one. (Given reasonable assumptions, see van Deemter 2002)
- Computational complexity: Assume that testing for usefulness takes constant time. Then worst-case time complexity is O(np) where np is the number of properties in P.

Incremental Algorithm (elab.)

- Better approximation of Maxim of Quantity (D&R95):
  - Properties represented as Attribute + Value pairs
    - <Origin,Sweden>, <Origin,Italy>, ...
    - <Colour,dark>, <Color,grey>, …
  - More or less specific values (subsumption taxonomy):
    - <Origin,America>, <Origin,Europe>, <Origin,Sweden>, <Origin,Italy>, ...
    - <Colour,dark>, <Color,light>, <Colour,green>, <Color,grey>, …

  Optimization within the set of properties which are values of the same attribute: FindBestValue

Incremental Algorithm (elab.)

r = individual to be described
C = contrast set
A = list of Attributes, in preference order
$V_{i,j}$ = value i of attribute j
L= properties in generated description

Incremental Algorithm (elab.)

$L := \{\}$

For all $A_i \in A$ do :

$V_{i,j} = \text{FindBestValue}(r, A_i)$

If $r \in [[V_{i,j}]]$ & $C \not\subset [[V_{i,j}]]$ then do

$L := L \cup \{V_{i,j}\}$

$C := C \cap [[V_{i,j}]]$

If $C = \{\}$ then Return L

Return Failure

Incremental Algorithm (elab.)

FindBestValue(r,A):

- Find value of A that
  user knows,
  is true of r,
  removes some distractors,
  (If such doesn't exist, go to next Attribute)
- Within this set, select the Value that
  removes the largest number of distractors (e.g., **most specific**)
- If there's a tie, select the more general one
- If there's still a tie, select an arbitrary one

D&R95, p.22, Fig.6

Incremental Algorithm (elab.)

Example:
Context set: D = {a,b,c,d,f,g}
Type: furniture (abcd), desk (ab), chair (cd)
Origin: Europe (bdfg), America (ac), Italy (bd)

Describe a:{desk, America}
(furniture removes fewer distractors than desk)
Describe b:{desk, Europe}
(European is more general than Italian)

Incremental Algorithm: Exercise

Exercise on Logical Completeness: Construct an example where no description is found, although one exists.

Hint: Let Attribute have Values whose extensions overlap.
Context set: D = {a,b,c,d,e,f}
Contains: wood (abe), plastic (acdf)
Colour: grey (ab), yellow (cd)
Describe a:{wood, grey} - Failure
(wood removes more distractors than plastic)
Compare:
Describe a: {plastic, grey} - Success

Incremental Algorithm (elab.)

A complication of the algorithm that has to do with realization:

- A description by a nominal group needs a head noun, but not all properties can be expressed as Nouns
- Example: Suppose Colour most-preferred Attribute, and target = a

Colours: dark (ade), light (bc), grey (a)
Type: furniture (abcde), desk (ab), chair (cde)
Origin: Sweden (ac), Italy (bde)
Price: 100 (ac), 150 (bd) , 250 ({})
Contains: wood ({}), metal ({abcde}), cotton(d)

Describe a: {grey}: 'the grey' ? (Not in English, 'the grey one')

Incremental Algorithm (elab.)

D&R's repair of the head-noun problem:

- Assume attribute type is special, and that its values can be expressed by nouns
- After the core algorithm, check whether Type is represented
- if not, then add the best value of the type Attribute to the description

Same effect achieved if type always included as first property

Incremental Algorithm: Complexity

- According to D&R: O(nd*nl)
  (Typical running time)
  Alternative assessment: O(nv)
  (Worst-case running time)
- Greedy Heuristic: O(nd*nl*na)

nd = nr. of distractors
nl = nr. of properties in the description
nv = nr. of Values (for all Attributes)
na = nr. Of properties known to be true of intended entity

Incremental Algorithm: Limitations

- Redundancy arises, but not for principled reasons, such as
  - marking topic changes, etc. (→ Corpus work by Pam Jordan et. al.)
  - making it easy to localize the object (→ Experimental work by Paraboni et al.)
- No relational properties (→ Dale&Haddock 1991, Horacek 1996)
- No reference to sets (→ van Deemter 2001)
- No differentiation of salience degrees (→ Krahmer&Theune 2002)
- Only nominal descriptions, not other forms of reference (pronouns)
- No interface to linguistic realization
  - No context-dependent handling of relative properties, e.g., "steep hill"
  - No vagueness of properties, e.g., "the long nail" vs. "the 5cm nail"
  - Content determination doesn't know which properties can(not) be realized and how complex the realization is (→ Horacek 1997, Stone&Doran 1997, Stone & Webber 1998)

Conclusions

- Practical application of conversational maxims
  - Operationalization
  - Formalization
  - Algorithm
  - Implementation(s)
  - Evaluation
- Instantiation on the concrete problem of GRE
- Computational vs. empirical motivation/justification/evaluation