Quantifier mappings

Marco René Spruit

May 1995

Introduction

In this paper an attempt has been made to clarify the process of quantifier mappings.

The recent article by **Sanford, Moxey and Paterson**, *Psychological Studies of Quantifiers*[1], has been taken as a starting point. This article refers to a paper by **Clark**, *Words, The world, and their possibilities*[2], when considering quantifiers as functions which take values depending on context and expectation.

This point of view has been examined in more detail. Not only because these articles primarily focus on the overall view but also because understanding the process of quantifier mappings in a discourse environment could be facilitated when described at an implementation level, since functions are to be described deterministically.

Overview

In the remainder of this section the two articles are reviewed briefly, followed by a few comments and directives.

In the next section the discourse schemes for the speaker and the listener, or addressee, are presented. These schemes include context-sensitivity, personalised subject-sensitive expectation and, although not worked out in detail, weight adaptations. Indirectly, focus shifting is supported in the sense that this can result from inadequate evaluation.

In the third section the functions are presented after laying out the conventions. The behaviour of the discourse parameters context and expectation are described deterministically at an implementation level. Also, a function hierarchy is provided to position the described functions in a more global discourse context.

In the fourth section the description is illustrated with a few traces of the internal processing involved in quantifier mappings.

This paper is concluded with a brief recapitulation.

[1] Sanford, A.J., Moxey, L.M., Paterson, K., *Psychological Studies of Quantifiers* (1994), University of Glasgow, Journal of Semantics 10, N.I.S. Foundation (1994), pp. 153-170.

[2] Clark, H.H., *Words, the world, and their possibilities* (1991), in G.R. Lockhead & James R. Pomerantz (eds), *The Perception of Structure: Essays in Honor of Wendell R. Garner* (1991), American Psychological Association, Washington, DC, pp. 263-280.

Articles

[Sanford et al, 1994]

In psychology, a popular view on quantifiers is explaining these as expressions that have *fuzzy projections* on to mental scales of amount. In other words, natural language quantifiers[3] are *approximate representations* of numerical values on a mental scale.

However, recent experiments show that quantifiers may be differentiated in terms of the *patterns of focus* that they produce. They may be of a positive, neutral or negative nature[4]. If this is true, these patterns of focus might also be a reflection of the patterns of inference that they cause. This focusing effect can be achieved through the quantifiers' polarity property[5].

The view that NLQ's are a way of denoting numerical values is discarded because of the variability of the mapping from quantifier to scale and vice versa. This mapping variability is mainly caused by two parameters, *context* and *expectation*.

[Clark, 1991]

Clark's *principle of possibilities* states that an entity is understood with reference to what it could have been. Its domain is restricted by speakers and listeners on the basis of a momentary common ground; it changes with the *occasion of use* (i.e. *context*).

When this principle is applied to natural language it means, for example, that word meanings are not static dictionary entries but products of a lexical process. The meaning of a word comes from the understanding of its context.

In short, it is argued that static dictionary theories are inadequate because they ignore this principle of possibilities. Its most important parameters are *context, expectation* and *social constraints*. These parameters make it impossible to describe quantifier mappings with static dictionary accounts.

---

[3] In this paper, NLQ is an abbreviation for Natural Language Quantifier.

Likewise, NRQ is an abbreviation for NumeRical Quantifier.

[4] For example, the sentence "*Disturbingly few* fans went to the match" is likely to cause a negative focus; Listener will probably think of this anouncement as "not much fans at all". If Listener would have been told that "*A number of* fans went to the match" instead, then Listener would not have been focused by Speaker's interpretation of this information and Listener might have ended up with a quite different interpretation of this anouncement.

[5] An example of a quantifier with negative polarity is *not many*. An example of a positive counterpart for *not many* is *some*.

Summary

In both articles the view on the process of quantifier mappings through static dictionary accounts is discarded. However, in this paper it is argued that the concept of quantifier mappings through dictionary accounts can coexist with the ideas presented in these articles.

It is not entirely unlikely that the concepts of context-sensitivity and personalised subject-sensitive expectation only *seem* not to fit in the concept of quantifier mappings through dictionary accounts because these articles do not attempt to describe the quantifier mapping process in detail.

In this paper it will be shown that the **static** dictionary mapping model can be extended so that the **dynamic** discourse parameters *context* and *expectation* are incorporated. The remaining static properties of the dictionary mapping model disappear by extending the model with **adaption** capabilities after each communication cycle. The procedure of quantifier mappings itself is **determinate** but its incoming variables are outcomes of the current context, personal expectation and communicative anticipation.

Schemes

The following two schemes have been accompanied with a description for the utterance: *"NLQ fans went to the match."*. The *ID*-column refers to the description in the functions section.

Speaker

Event	ID	Description
Numerical fact		Speaker found out *iNRQ* fans went to the match
Initialisation	**F1**	If Speaker plans to open a new topic, Speaker determines weights ($weight_{Speaker}$, $weight_{Listener}$, $weight_{BiasRange}$)
Internal expectation		Speaker thought $iExpectation_{Speaker}$ fans would have gone to the match
Internal scope		Speaker thinks $iMaximum_{Speaker}$ fans could have gone to the match
External expectation		Speaker assumes Listener expected $iAnticipation_{Speaker}$ fans to have gone to the match
Mapping to NLQ	**F2**	Speaker maps *iNRQ* to *sNLQ* based on *iDeviation*, using $iMaximum_{Speaker}$, $iExpectation_{Speaker}$, $iAnticipation_{Speaker}$, $weight_{Speaker}$, $weight_{Listener}$, $weight_{BiasRange}$
Adaptation		Speaker adapts topic weights and amounts appropriately (for example, evaluate *iDeviation*)
NLQ utterance		Speaker says: *sNLQ* + "fans went to the match"

Listener

Event	ID	Description
NLQ perception		Listener hears: *sNLQ* + "fans went to the match"
Initialisation	**F1**	If Listener assumes the perceived sentence is to be the start of a new topic, Listener determines weights ($weight'_{Speaker}$, $weight'_{Listener}$, $weight'_{BiasRange}$)
Internal expectation		Listener thought $iExpectation_{Listener}$ fans would have gone to the match
Internal scope		Listener thinks $iMaximum_{Listener}$ fans could have gone to the match
External expectation		Listener assumes Speaker thought Listener expected $iAnticipation_{Listener}$ fans to have gone to the match
Mapping to NRQ	**F3**	Listener maps *sNLQ* to *iNRQ'* based on *iDeviation*, using $iExpectation_{Listener}$, $iMaximum_{Listener}$, $iAnticipation_{Listener}$, $weight'_{Listener}$, $weight'_{Speaker}$, $weight'_{BiasRange}$
Numerical assumption		Listener assumes *iNRQ'* fans went to the match
Adaptation		Listener adapts topic weights and amounts appropriately (for example, evaluate *iDeviation*)

Functions

Conventions

The functions presented here are described in a procedural manner because of the needed deterministic accuracy. The notation used bears resemblance to programming languages such as Pascal and C. However, the notation used should be readable as formally stated plain text if the following syntax conventions are kept in mind:

- Commands are printed in **BOLD UPPERCASE** format.
- Functions are printed in **Bold TitleCase** format.
- Variables are printed in *italic* format.

  If prefixed with **VAR** in a function declaration, the variable may be modified permanently.
- Variable prefixing:
  - Arrays are prefixed with ***a***.
  - Integers are prefixed with ***i***, for example *iNRQ*.
  - Strings are prefixed with ***s***, for example *sNLQ*.

  Arrays containing tuples of integers and strings are therefore prefixed with ***ais***;

  For example, *aisMapping* refers to an array that maps between integers and strings.
- Variable subscripts denote the variable's reference.

  For example, $iMaximum_{Speaker}$ refers to a maximum value assigned to Speaker.
- Syntax symbols:
  - << Assignment or transfer. For example, *x* << *y* means that the value of *y* is transferred to *x*.
  - == Equality. For example, *x* == *y* means that the value of *x* is equal to *y*.
  - != Inequality. For example, *x* != *y* means that the value of *x* is not equal to *y*.
  - > Is-greater-than. For example, *x* > *y* means that the value of *x* is greater than *y*.
  - >= Is-greater-than-or-equal-to.
  - < Is-smaller-than.
  - <= Is-smaller-than-or-equal-to.
  - [*x*] Array element access. For example, *aisMapping[x]* refers to the x^{th} element in the array.
  - ++ Increase by one.

    Example 1: *i*++ : first evaluate *i,* then increase by one.

    Example 2: ++*i* : first increase by one, then evaluate *i.*
  - -- Decrease by one.
  - ; End of statement.

Hierarchy

The basic hierarchy presented here may provide a more global view of the context of the described functions as an alternative for the schemes in table form on page 5.

The function ID's on the right side of this scheme denote the functionality described in detail in this paper.

```
::DoWeightInitialisation                                (F1)

WHILE ( topic - communication entity[6] )
{
    IF ( bIsSpeaker )
        IF ( ::CommunicationContainsNRQ )
            IF ( ::IsAdjectiveNRQ )
                ::MapToNLQ_Adjective                    (F2)

    IF ( bIsListener )
        IF ( ::CommunicationContainsNLQ )
            IF ( ::IsAdjectiveNLQ )
                ::MapToNRQ_Adjective                    (F3)

    ::DoWeightAdaptation
}
```

F1::DoWeightInitialisation

(

topic **,**

VAR $weight_{Speaker}$ **,**

VAR $weight_{Listener}$ **,**

VAR $weight_{BiasRange}$

)

This function is called to determine the weights for each topic-communication entity-tuple. Once the weights are initialised, they are updated each communication cycle via the adaptation component for as long as this topic-communication entity-line is not interrupted.

$Weight_{BiasRange}$ is a value that reflects the degree of correlation between the various mapping tables. It may be required in NLQ-to-NRQ mappings.

[6] A communication entity can be another person, a group of persons, etc.

See **F33::DoMappingToNRQ_Adjective_Negative()** for more details.

```
BEGIN
        weight_Speaker   << KnowledgeBase( topic , Speaker ) ;
        weight_Listener  << KnowledgeBase( topic , Listener ) ;
        weight_BiasRange << CalculateBiasRange( topic ) ;

        IF ( weight_Speaker == <undefined> ) THEN
                weight_Speaker << 0 ;
        IF ( weight_Listener == <undefined> ) THEN
                weight_Listener << 0 ;
        IF ( weight_BiasRange == <undefined> ) THEN
                weight_BiasRange << 1 ;
END
```

F2::MapToNLQ_Adjective

(

iNRQ **,**

iMaximum$_{Speaker}$ **,**

iExpectation$_{Speaker}$ **,**

iAnticipation$_{Speaker}$ **,**

weight$_{Speaker}$ **,**

weight$_{Listener}$

)

This function is the main function in the NRQ-to-NLQ mapping process.

In this description **::UnifyValueTypes()** is called first to ensure that all values are in one format. For example, percentages may be converted to absolute values. This approach has primarily been taken here to minimise the number of needed functions.

The NRQ is returned if the neutral mapping cannot be calculated. Otherwise, the discourse mapping and its distance from the neutral mapping is calculated. This deviation determines whether a positive, neutral or negative mapping is to be used.

See **F22::DoMappingToNLQ_Adjective_Negative()** for more details.

```
BEGIN
        UnifyValueTypes( iNRQ , iMaximum_Speaker , iExpectation_Speaker ,
                         iAnticipation_Speaker ) ;

        IF ( iMaximum_Speaker == <undefined> ) THEN
                RETURN iNRQ ;
```

```
        iMapping_neutral << ( iNRQ / iMaximum_Speaker ) * 100;

        IF ( iExpectation_Speaker != <undefined> ) THEN
               iMapping_expectation <<
                      ( iExpectation_Speaker / iMaximum_Speaker ) * 100 ;
        IF (iAnticipation_Speaker != <undefined> ) THEN
               iMapping_anticipation <<
                      ( iAnticipation_Speaker / iMaximum_Speaker ) * 100 ;

        IF ( ( iExpectation_Speaker == <undefined> ) AND
             ( iAnticipation_Speaker == <undefined> ) ) THEN
               RETURN DoMappingToNLQ_Adjective_Neutral( iMapping_neutral ) ;

        IF ( ( iExpectation_Speaker != <undefined> ) AND
             ( iAnticipation_Speaker == <undefined> ) ) THEN
               iDeviation <<
                      weight_Speaker * ( iMapping_neutral - iMapping_expectation ) ;
        IF ( ( iExpectation_Speaker == <undefined> ) AND
             ( iAnticipation_Speaker != <undefined> ) ) THEN
               iDeviation <<
                      weight_Listener * ( iMapping_neutral - iMapping_anticipation ) ;
        IF ( ( iExpectation_Speaker != <undefined> ) AND
             ( iAnticipation_Speaker != <undefined> ) ) THEN
               iDeviation <<
                      ( weight_Speaker * ( iMapping_neutral - iMapping_expectation ) +
                        weight_Listener * ( iMapping_neutral - iMapping_anticipation ) )
                      / 2 ;

        iMapping_discourse <<
               iMapping_neutral + ( ( iMapping_neutral / 100 ) * iDeviation ) ;

        IF ( iDeviation >= 0 ) THEN
               RETURN DoMappingToNLQ_Adjective_Positive( iMapping_discourse ) ;
        IF ( iDeviation < 0 ) THEN
               RETURN DoMappingToNLQ_Adjective_Negative( iMapping_discourse ) ;
END
```

F22::DoMappingToNLQ_Adjective_Neutral

(

iPercentageNeutral

)

This function is called from **F2::MapToNLQ_Adjective()**.

Here, the actual NRQ-to-NLQ mapping takes place. The closest match for the incoming percentage is searched in the mapping table and its corresponding NLQ is returned.

```
BEGIN
       aisMapping << { {   0, "no" } ,
                       {  10, "a few" } ,
                       {  20, "some" } ,
                       {  30, "a number of" } ,
                       {  90, "most" } ,
                       { 100, "all" } } ;
       i << 0 ;
       WHILE ( aisMapping[i++][0] < iPercentageNeutral )
              CONTINUE ;

       iDistanceToPreviousEntry << iPercentageNeutral - aisMapping[i - 1] ;
       iDistanceToNextEntry << aisMapping[i] - iPercentageNeutral ;

       IF ( iDistanceToNextEntry <= iDistanceToPreviousEntry )
              RETURN aisMapping[i][1] ;
       RETURN aisMapping[i - 1][1] ;
END
```

F3::MapToNRQ_Adjective

(

sNLQ **,**

$iMaximum_{Listener}$ **,**

$iExpectation_{Listener}$ **,**

$iAnticipation_{Listener}$ **,**

$weight'_{Listener}$ **,**

$weight'_{Speaker}$ **,**

weight'$_{BiasRange}$

)

This function is the main function in the NLQ-to-NRQ mapping process.

In this description **::UnifyValueTypes()** is called first to ensure that all values are in one format. For example, percentages may be converted to absolute values. This approach has primarily been taken here to minimise the number of needed functions.

If the neutral mapping cannot be calculated, the mapping is aborted and an undefined value is returned. If the neutral mapping can be calculated but there is no information available for this topic, the mapping is also aborted and the NLQ is returned. Otherwise, the discourse mapping and its distance from the neutral mapping is calculated. This deviation determines whether a positive, neutral or negative mapping is to be used.

See **F33::DoMappingToNRQ_Adjective_Negative()** for more details.

```
BEGIN
        UnifyValueTypes( iMaximum_Listener , iExpectation_Listener ,
                         iAnticipation_Listener ) ;


        iMapping_neutral << DoMappingToNRQ_Adjective_Neutral(
                                    sNLQ , 0 , weight'_BiasRange ) ;
        IF ( iMapping_neutral == <undefined> ) THEN
               RETURN <undefined> ;


        IF ( iMaximum_Listener == <undefined> ) THEN
               RETURN sNLQ ;
        IF ( ( iExpectation_Speaker == <undefined> ) AND
             ( iAnticipation_Speaker == <undefined> ) ) THEN
               RETURN ( iMapping_neutral * ( iMaximum_Listener / 100 ) ) ;


        IF ( iExpectation_Listener != <undefined> ) THEN
               iMapping_expectation <<
                      ( iExpectation_Listener / iMaximum_Listener ) * 100 ;
        IF (iAnticipation_Listener != <undefined> ) THEN
               iMapping_anticipation <<
                      ( iAnticipation_Listener / iMaximum_Listener ) * 100 ;


        IF ( ( iExpectation_Listener != <undefined> ) AND
             ( iAnticipation_Listener == <undefined> ) ) THEN
               iDeviation <<
                      weight'_Listener * ( iMapping_neutral - iMapping_expectation ) ;
        IF ( ( iExpectation_Listener == <undefined> ) AND
             ( iAnticipation_Listener != <undefined> ) ) THEN
               iDeviation <<
```

```
                    weight'Speaker * ( iMappingneutral - iMappinganticipation ) ;
    IF ( ( iExpectationListener != <undefined> ) AND
         ( iAnticipationListener != <undefined> ) ) THEN
          iDeviation <<
                    ( weight'Listener * ( iMappingneutral - iMappingexpectation ) +
                      weight'Speaker * ( iMappingneutral - iMappinganticipation ) )
                    / 2 ;

    IF ( iDeviation >= 0 ) THEN
          iMappingdiscourse << DoMappingToNRQ_Adjective_Positive(
                                sNLQ , iDeviation , weight'BiasRange );
    IF ( iDeviation < 0 ) THEN
          iMappingdiscourse << DoMappingToNRQ_Adjective_Negative(
                                sNLQ , iDeviation , weight'BiasRange ) ;

    IF ( iMappingdiscourse == <undefined> ) THEN
          RETURN <undefined> ;

    RETURN ( iMappingdiscourse * ( iMaximumListener / 100 ) ) ;
END
```

F33::DoMappingToNRQ_Adjective_Negative

(

***sNLQ* ,**

***iDeviation* ,**

***weight*$_{BiasRange}$**

)

This function is called from **F3::MapToNRQ_Adjective()**.

Here, the actual NLQ-to-NRQ mapping takes place. First, the NLQ is searched in the mapping table. If this string is found, then its corresponding percentage and the incoming context deviation are used to calculate the NRQ. The influence of the deviation is limited to the value of the mapping percentage, i.e. the range of the return value is kept between zero and twice the mapping percentage.

The deviation is levelled via the bias-range weight if the NLQ cannot found in this table. This is necessary because a NLQ is likely to have different corresponding numerical values in different mapping tables. After levelling, control is transferred to the neutral mapping method . If the NLQ can also not be mapped neutrally, a final attempt is made via a positive mapping. This method is executed after levelling the deviation again.

```
BEGIN
```

```
    aisMapping << { {   0, "none" } ,
                    {  10, "few" } ,
                    {  20, "quite few" } ,
                    {  25, "not many" } ,
                    {  95, "not all" } ,
                    { 100, "all" } } ;

    i << 0 ;
    WHILE  ( ( i <= SIZEOF( aisMapping ) AND
             ( aisMapping[i++][1] != sNLQ ) )
           CONTINUE ;

    IF ( --i < SIZEOF( aisMapping ) ) THEN
           RETURN ( aisMapping[i][0] -
                    ( ( aisMapping[i][0] / 100 ) * iDeviation ) );

    iDeviation << iDeviation * weight_BiasRange ;
    iMappingNeutral << DoMappingToNRQ_Adjective_Neutral(
                               sNLQ , iDeviation , weight_BiasRange ) ;
    IF ( iMappingNeutral != <undefined> ) THEN
           RETURN ( iMappingNeutral -
                    ( ( iMappingNeutral / 100 ) * iDeviation ) );

    iDeviation << iDeviation * weight_BiasRange ;
    iMappingPositive << DoMappingToNRQ_Adjective_Positive(
                               sNLQ , iDeviation , weight_BiasRange ) ;
    IF ( iMappingPositive != <undefined> ) THEN
           RETURN ( iMappingPositive -
                    ( ( iMappingPositive / 100 ) * iDeviation ) );

    RETURN <undefined> ;
END
```

Examples

Example 1 : Speaker - No mapping

Action	Variable	Value
Initialisation	*topic*	tennis
	$weight_{Speaker}$	<*undefined*> ⇨ 0
	$weight_{Listener}$	<*undefined*> ⇨ 0
	iNRQ	**1000**
	$iMaximum_{Speaker}$	<*undefined*>
	$iExpectation_{Speaker}$	<*undefined*>
	$iAnticipation_{Speaker}$	<*undefined*>
Mapping		[too much missing information]
Utterance		"**1000** fans went to the match"

Example 2 : Speaker - Neutral mapping

Action	Variable	Value
Initialisation	*topic*	tennis
	$weight_{Speaker}$	0.8
	$weight_{Listener}$	<*undefined*> ⇨ 0
	iNRQ	**1000**
	$iMaximum_{Speaker}$	3000
	$iExpectation_{Speaker}$	<*undefined*>
	$iAnticipation_{Speaker}$	<*undefined*>
Calculation	$iMapping_{neutral}$	33.3 % << (1000 / 3000) * 100
Mapping	*sNLQ*	"a number of" << DoMappingTo...(33.3)
Utterance		"**A number of** fans went to the match"

Example 3 : Speaker - Inclusion of expectation

Action	Variable	Value
Initialisation	*topic*	tennis
	$weight_{Speaker}$	0.8
	$weight_{Listener}$	<*undefined*> ⇨ 0
	iNRQ	**1000**
	$iMaximum_{Speaker}$	3000
	$iExpectation_{Speaker}$	2000
	$iAnticipation_{Speaker}$	<*undefined*>
Calculation	$iMapping_{neutral}$	33.3 % << (1000 / 3000) * 100
	$iMapping_{expectation}$	66.6 % << (2000 / 3000) * 100
	iDeviation	-26.4 % << 0.8 * (33.3 - 66.6)
	$iMapping_{discourse}$	24.5 % << 33.3 + ((33.3 / 100) * -26.4)
Mapping	*sNLQ*	"not many" << DoMappingTo...(24.5)
Utterance		"**Not many** fans went to the match"

Example 4 : Speaker - Inclusion of anticipation

Action	Variable	Value
Initialisation	*topic*	tennis
	$weight_{Speaker}$	0.8
	$weight_{Listener}$	0.95
	iNRQ	**1000**
	$iMaximum_{Speaker}$	3000
	$iExpectation_{Speaker}$	2000
	$iAnticipation_{Speaker}$	2800
Calculation	$iMapping_{neutral}$	33.3 % << (1000 / 3000) * 100
	$iMapping_{expectation}$	66.6 % << (2000 / 3000) * 100
	$iMapping_{anticipation}$	93.3 % << (2800 / 3000) * 100
	iDeviation	-41.8 % << ((0.8 * (33.3 - 66.6)) + (0.95 * (33.3 - 93.3))) / 2
	$iMapping_{discourse}$	19.4 % << 33.3 + ((33.3 / 100) * -41.8)
Mapping	*sNLQ*	"quite few" << DoMappingTo...(19.4)
Utterance		"**Quite few** fans went to the match"

Example 5 : Listener - Inclusion of anticipation

Action	Variable	Value
Initialisation	*topic*	tennis
	$weight_{Listener}$	0.5
	$weight_{Speaker}$	0.8
	$weight_{BiasRange}$	0.7
	sNLQ	"**few**"
	$iMaximum_{Listener}$	4000
	$iExpectation_{Listener}$	80 %
	$iAnticipation_{Listener}$	95 %
Type unification	$iExpectation_{Listener}$	3200
	$iAnticipation_{Listener}$	3800
Calculation	$iMapping_{neutral}$	10 % << DoMappingTo...("few" , 0 , 0.7)
	$iMapping_{expectation}$	80 % << (3200 / 4000) * 100
	$iMapping_{anticipation}$	95 % << (3800 / 4000) * 100
	iDeviation	-51.5 % << ((0.5 * (10 - 80)) + (0.8 * (10 - 95))) / 2
	$iMapping_{discourse}$	15.15 % << 10 + ((10/100) * -51.5) << DoMappingTo...("few" , -51.5 , 0.7)
Mapping	*NRQ'*	606 << 15.15 * (4000 / 100)
Assumption		"**606** fans went to the match"

Example 6 : Listener - Levelling through bias range

Action	Variable	Value
Initialisation	*topic*	tennis
	$weight_{Listener}$	0.5
	$weight_{Speaker}$	0.8
	$weight_{BiasRange}$	0.7
	sNLQ	"**quite a lot**"
	$iMaximum_{Listener}$	4000
	$iExpectation_{Listener}$	3200
	$iAnticipation_{Listener}$	95 %
Type unification	$iAnticipation_{Listener}$	3800
Calculation	$iMapping_{neutral}$	60 % << DoMappingTo...("quite a lot" , 0 , 0.7)

	$iMapping_{expectation}$	80 % << (3200 / 4000) * 100
	$iMapping_{anticipation}$	95 % << (3800 / 4000) * 100
	iDeviation	-19 % << ((0.5 * (60 - 80)) + (0.8 * (60 - 95))) / 2
Mapping (attempt to map negatively)	*iDeviation*	-13.3 % << -19 * 0.7
Mapping (attempt to map neutrally)	*iDeviation*	-9.3 % << -13.3 * 0.7
Mapping (attempt to map positively)	$iMapping_{discourse}$	54.4 % << 60 + ((60 / 100) * -9.3)
Mapping	*NRQ'*	2176 << 54.4 * (4000 / 100)
Assumption		"**2176** fans went to the match"

Example 7 : Listener - Enable focus shifting

Action	**Variable**	**Value**
Initialisation	*topic*	tennis
	$weight_{Listener}$	0.5
	$weight_{Speaker}$	0.8
	$weight_{BiasRange}$	0.7
	sNLQ	"**an exorbitant proportion of**"
	$iMaximum_{Listener}$	4000
	$iExpectation_{Listener}$	3200
	$iAnticipation_{Listener}$	3800
Calculation	$iMapping_{neutral}$	<*undefined*> << DoMappingTo...("an exorbitant proportion of" , 0 , 0.7)
Assumption		<*undefined*>

Recapitulation

In this description, quantifier mappings are still performed through dictionary accounts. Also, quantities are still considered to be stored numerically. The extensions consist essentially of

- personalisation of focus-sensitive quantifier mapping information
- personalisation of the perception of and thus the information about the world
- determination of weights per topic-communication entity
- continuous adaptation of the information about the world

The actual mapping data in the **::DoMappingTo...()** functions is, of course, far from complete and serves only as a general indication. The function-naming should also imply the relative scope of the described functions, since different mapping procedures and data are likely to be necessary for different NLQ-categories and expectation evaluations.

Some function return values indicate special cases. For example, **::MapToNLQ...()** returns the incoming numerical value if there is no adequate information available, i.e. Speaker cannot interpret the fact. Speaker could then use the numerical value itself, perhaps because Listener might be able to help create a context for Speaker.

Also, focus shifting may be caused when functions return an undefined value. This means that Listener cannot process the data. He or she might request clarification and thus alter the focus. For example, Speaker might use a NLQ that Listener cannot resolve. If Listener wants to know with what quantity Speaker associates this NLQ and explicitly requests so, the focus will shift and Listener may adapt his or her quantifier mapping tables accordingly.

Needless to say, this description of the process of quantifier mappings in discourse is far from complete. However, the interaction between and the intertwining of conceptual and implementation levels has turned out to be quite useful in the process of producing this paper.

Marco René Spruit,

Department of Computational Linguistics,
Faculty of Arts,
University of Amsterdam,

spruit@xs4all.nl,

May 1995.