Site report:

Research Group in Digital Linguistics

at NTNU, Trondheim, Norway

http://www.ntnu.no/web/isk/digital-linguistics

Lars Hellan

DELPH-IN meeting, July 29, 2013

Saarbrücken/ St. Wendel

Members of group at NTNU

Dorothee Beermann
Tore Bruland
Lars Hellan
Mads H. Sandøy
Elias Aamot

Members of group abroad

Dan Flickinger, Stanford
Pavel Mihaylov, Sofia
Mary Esther Kropp Dakubu, Accra

Activities

Basic:

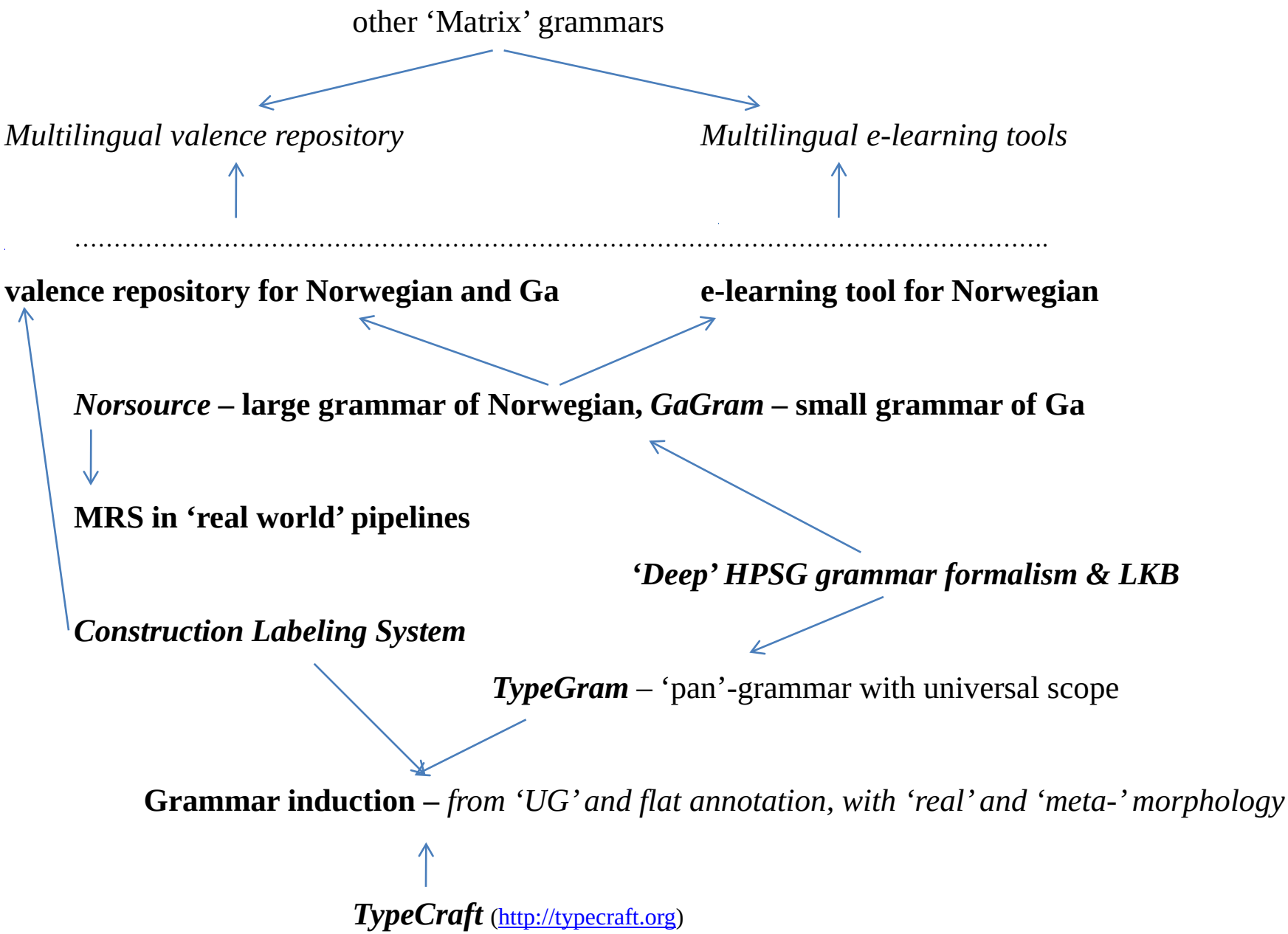
Grammar engineering, with large and small ‘Matrix’ grammars, and experimental general designs

Tool development for online annotation

Design of typological construction-classification systems

Extensions (by slide numbers of presentation):

- Deriving verb valence repositories(mono- and multi-lingual) (5-10)
- Defining semantic spaces for situation types and roles (11-15)
- Construction of pipelines from grammar-produced MRS to knowledge bases and ‘real world’ scenarios (16)
- Induction of grammars from ‘flat’ annotation and implemented ‘UG’ (17-24)
- Construction of e-learning tools based on Matrix grammars (25-30)


other ‘Matrix’ grammars
Multilingual valence repository
Multilingual e-learning tools
valence repository for Norwegian and Ga
e-learning tool for Norwegian
Norsource – large grammar of Norwegian, GaGram – small grammar of Ga
MRS in ‘real world’ pipelines
‘Deep’ HPSG grammar formalism & LKB
Construction Labeling System
TypeGram – ‘pan’-grammar with universal scope
Grammar induction – from ‘UG’ and flat annotation, with ‘real’ and ‘meta-’ morphology
TypeCraft (http://typecraft.org)

Valence profile (v-profile)

- We assume a combination of ‘formal’ and ‘functional’ characterization of verb valence, such that, for example, a notion like ‘standard transitive’ (with NP subject and NP object) can be represented from the formal side as something like ‘NP+NP’, from the functional side as ‘transitive’ (as a notion implying the existence of a subject and an object), and in a combined fashion as ‘v-tr-suN_obN’. Many more parameters will be relevant in a valence type specification, and the number of verb-valence types can be estimated to lie between 200 and 300 for a language, and the assembly of such types we may call the *valence profile* (*v-profile*) of the language.
- A multilingual valence type inventory will in principle have the same architecture, only with an additional parameter of *languages*.
- http://regdili.idi.ntnu.no:8080/multilanguage_valence_demo/multivalence

Multilanguage Valency Patterns

Version 1.0

Languages:

☑ Norwegian ☑ Ga

Search fields:

V-key	Syntactic Arguments	Semantic and Functional Properties	Type
b	NP+NP+NP		

Search | Count | Clear | Download

Search Result

ga show ba_14
ga show ba_8
no show bake_ditr
ga show baŋ_28
no show belære_ditr
no show benevne_ditr
no show berøve_ditr
no show beskjære_ditr
no show beta_ditr
no show betale_ditr
no show betro_ditr
no show bevilge_ditr
ga show bi_52
no show bibringe_ditr
no show booke_ditr
no show bringe_ditr
ga show bu_90
no show by_ditr
ga show bɔ_74
ga show bɔ_76
ga show bɔle_85
ga show bɔɔs_88

Multilanguage Valency Patterns

Version 1.0

Languages:

☑ Norwegian ☑ Ga

Search fields:

V-key	Syntactic Arguments	Semantic and Functional Properties	Type
b	NP+NP+NP		

Search | Count | Clear | Download

Lexicon Instance

Language	ga
Verb Id	bɔle_85
Syntactic Arguments	NP+NP+NP
Semantic and Functional Properties	ternaryRel
Verb Type	v-ditr
Example of type	
Orthography	<"bɔle">
Phon	<"bɔ̀lè">
Engl-gloss	<"expect">
Example	Wɔ-bɔle-ee bo nakai
Gloss	1P.AOR-go.around-NEG.IMPERF 2S that
Free-transl	we didn't expect it of you, that you would behave in that manner.

Import from a computational grammar – 1a

Strategy 1. Populating the databases based on *Lexical (valence) types*:

- One makes a correspondence list with members like (i) and (ii) below.
- To the left of each arrow is a valence type name, as employed in the grammar, and the lines to the right state 'expansions' of the type name, in the more perspicuous format entered in the database:

(i) v-intrImpers =>
SAS: "EXPL"
FS: impersonal
Sit: weatherProcess
Example of type: det regner

(ii) v-intrImpersPrtcl =>
SAS: "EXPL+adpos"
FS: impersonal
Sit: weatherProcess
Example of type: det klarner opp

…

Import from a computational grammar – 1b

- Populating the column ‘Example_of_type’ is easy, since a v-profile is quite limited. Automatic import to it is possible in the following way: a grammar of this type normally has various test-suites, one of which may be for reflecting vale All 'Examples_of_types' receive a morpheme level annotation in TypeCraft, adding a further level of annotation. nce frames in the language. If such a valence test-suite is indexed for the valence type of each sentence, the sentences might be selected for the their respective ‘Example_of_type’ occurrences through automatic selection from the test-suite.
- Since the list is limited, entering the sentences one by one, as in the slide above, is also feasible. All 'Examples_of_types' receive a morpheme level annotation in TypeCraft, adding a further level of annotation.

Import from a computational grammar - 2

Strategy 2. Using AVMs

- From the AVM of each verb as defined by the grammar (with unification and type resolution performed), SAS, FS and Sit can be assigned for each verb. Here a fixed constellation of paths of AVMs is run through for every lexical entry, delivering results defined within the same repertoire as used on Strategy 1. For instance, to induce the SAS of “snø”, the procedure would base itself on the following AVM - SAS correspondence:

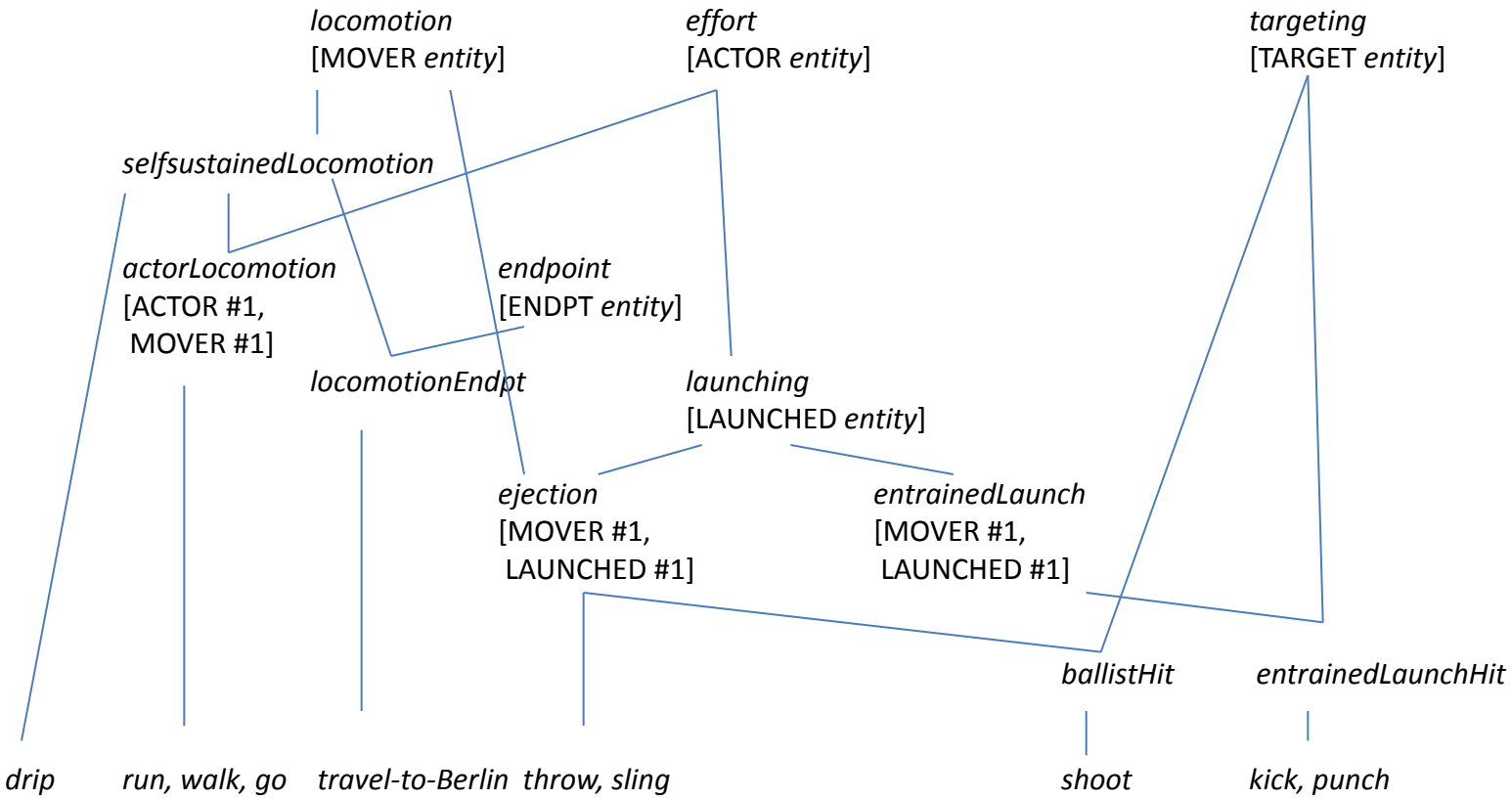
[SYNSEM.LOCAL.CAT.VAL.SUBJ.FIRST.LOCAL.CAT.HEAD expl-pron, => “EXPL”
SYNSEM.LOCAL.CAT.VAL.COMPS null,
SYNSEM.LOCAL.CAT.VAL.ICOMPS null]

This strategy ignores the verb types of the entries, so that the content of each slot is inferred directly from the feature structure of the verb. While LKB grammars largely use distinct lexical types, but have in common the basic AVM structures, strategy 2 may conceivably require less alternating scripts across LKB grammars than strategy 1 will. Also here, the slot ‘Example’ will not get filled, for the same reason as above.

’Sit-types’

- We are assuming that situation types can be included in valence description.
- While grammatical valence frames can be projected from normal ’deep’ grammars, it is less obvious that situation types can be, since they are rarely included in functioning grammars.
- We here show a possible format for representation of situation types.

Excerpt of a possible situation-type hierarchy



Modeling situation types

- When establishing correspondences between valence types and situation types, one has to avoid that the labels for situation types get too tightly linked to actual words used in one or more languages, that is, that the situation type inventory becomes circularly dependent on the inventory of linguistic constructs to be analyzed.
- Moreover, a typology of situation types will range from the very general to the very specific, and regardless of generality level, a situation type will be tied to a set of 'participant roles' characteristic of the type. Subsumption relations of situation types can be reflected in the roles, such that, e.g., situation types corresponding to *stand*, *seat*, and *lay*, subsumed by a situation type PLACEMENT, will share the roles of PLACEMENT, but with further specification in terms of posture of the 'placed' item.
- A standard way of modeling subsumption factors is by means of multiple inheritance hierarchies, where an attribute, in this case standing for a *participant role*, can be introduced only with one type, but be inherited by all the subtypes of that type. The next slide illustrates this design, for a very small segment of a possible situation type hierarchy. The architecture illustrated is that of the LKB formalism. Relative to this architecture, the figure also illustrates what replaces words in the analytic formalism, namely attributes (indicating parameters to be specified) and values, both in a 'universal' terminology independent of any specific language.

- Given the formal principles of the LKB system, there will be strict ties between situation types and roles: each role is licensed by a specific situation type, and thereby licensed for any of its subtypes, and a rich multiple inheritance system is what allows for the desired co-specifications of roles when relevant. In this respect the system is far stricter than any comparable system in this domain, with the inventory of situation types consistently balanced against (but considerably larger than) the inventory of participant roles.
- The significance of keeping situation types separate from word inventories still cannot be underestimated. A common situation illustrating the point is where one and the same situation type is cast in different valence frames (or differently 'profiled') across languages (with no indication of 'frame alternations' intervening). One example is seen in the next slide, where the situation type PLACEMENT is associated with a double object pattern in Ga, and an "NP+NP+PP" pattern in English. Clearly, the situation type PLACEMENT is here the same, independently of whether the syntactic frame is "NP+NP+NP" or "NP+NP+PP" (and independently of the role distribution among the last two constituents, and thus dissociated from word encodings such as *put* vs. *wo*).

’Placement’ construction in *Ga*

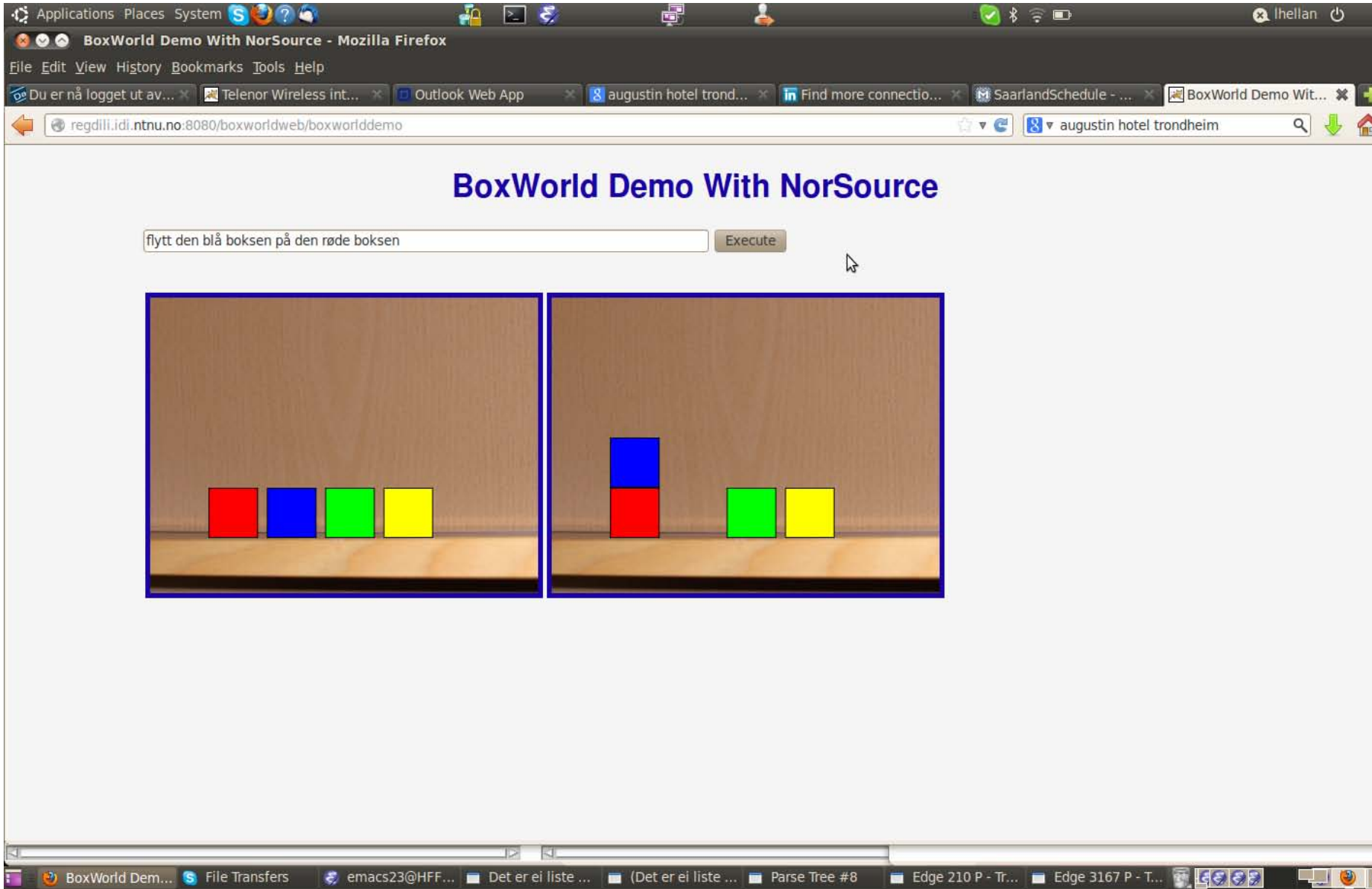
[v-ditr-obPostp-suAg_obEndpt_ob2Mover-PLACEMENT]

Amɛ-wo	***tsɔne***	***lɛ***	***mli***	***yɛlɛ***
3P.AOR-put	vehicle	DEF	inside	yam
V	N	Art	N	N

‘They put [vehicle’s inside] [yam]’

=‘They put yams in the lorry.’

Talking to a ’box world’ via MRS



Inducing grammatical types and rules from construction types, valence-types and valence-profiles of a language, given a defined classification system

From a valence-profile, or a set of valence-types of a language, one can induce ***lexical types*** for its grammar (examples for Ga):

v-ditr-suAg_obAff_ob2Instr-CUTTING

v-ditr-suAg_obLoc_ob2Res-CUTTING

v-ditr-suAg_obTh_ob2Instr-PENETRATION

v-ditr-suAg_obTrgt_ob2Endpt-COMMUNICATION

v-ditr-suAg_iobTrgt_obThmover-COMMUNICATION

vHab-ditr-suNrg_ob2DECLcmp-obSens_ob2Thsit-COGNITION

Types of syntactic constructions, such as Serial Verbs (from Akan)

svAspID-v1tr-v1obIDv2su-v1suAg_v1obEjct-v2tr-v2suTh_v2obEndpt-CONTACTEJECTION

can induce syntactic ***combinatorial rules*** for the grammar.

Classified morpho-lexical types, such as (for Citumbuka)

V-ditrCs-obCsu_ob2Cob-CAUSATION

can induce ***lexical rules*** for the grammar (here a rule of causativization in the grammar of Citumbuka).

Illustrating an algorithmic linking system in terms of *tdl*

```
v-ditr-obPostp-suAg_obEndpt_ob2Th-PLACEMENT :=
    v & ditr & obPostp & suAg & obEndpt & ob2Th & PLACEMENT.

v :=  sign & [HEAD headverb].
ditr :=  ditr-lex.
obPostp := sign & [GF.OBJ poss-sign &
    [ ACTNTS.PRED spatial-coord_rel]].
suAg := sign & [GF.SUBJ.INDX.ROLE agent].
obEndpt := sign & [GF.OBJ.INDX #1 & [ROLE endpnt],
                                ACTNTS.DIR.ACT2 #1].
ob2Th := sign & [GF.OBJ.INDX.ROLE theme-locative].
PLACEMENT := sign & [ SIT-TYPE placement_sit ].
```

Illustrating the same system in terms of *AVMs*

Ex.: **v-tr-suAg_obAffincrem-COMPLETED_MONODEVMNT**

v - - - $[\text{HEAD verb}]$

tr - - -

$$\begin{bmatrix} \text{GF} \begin{bmatrix} \text{SUBJ} \begin{bmatrix} \text{INDX } \boxed{1} \end{bmatrix} \\ \text{OBJ} \begin{bmatrix} \text{INDX } \boxed{2} \end{bmatrix} \end{bmatrix} \\ \text{ACTANTS} \begin{bmatrix} \text{ACT1 } \boxed{1} \\ \text{ACT2 } \boxed{2} \end{bmatrix} \end{bmatrix}$$

suAg - - - $\left[\text{GF} \left[\text{SUBJ} \left[\text{INDX} \left[\text{ROLE agent}\right]\right]\right]\right]$

obAffincrem - - - $\left[\text{GF} \left[\text{OBJ} \left[\text{INDX} \left[\text{ROLE aff-increm}\right]\right]\right]\right]$

COMPLETED_MONODEVMNT - - - $\begin{bmatrix} \text{ASPECT completed} \\ \text{SIT-TYPE monotonic\_development} \end{bmatrix}$

Unification result

v-tr-suAg_obAffincrem-COMPLETED-MONODEVMNT

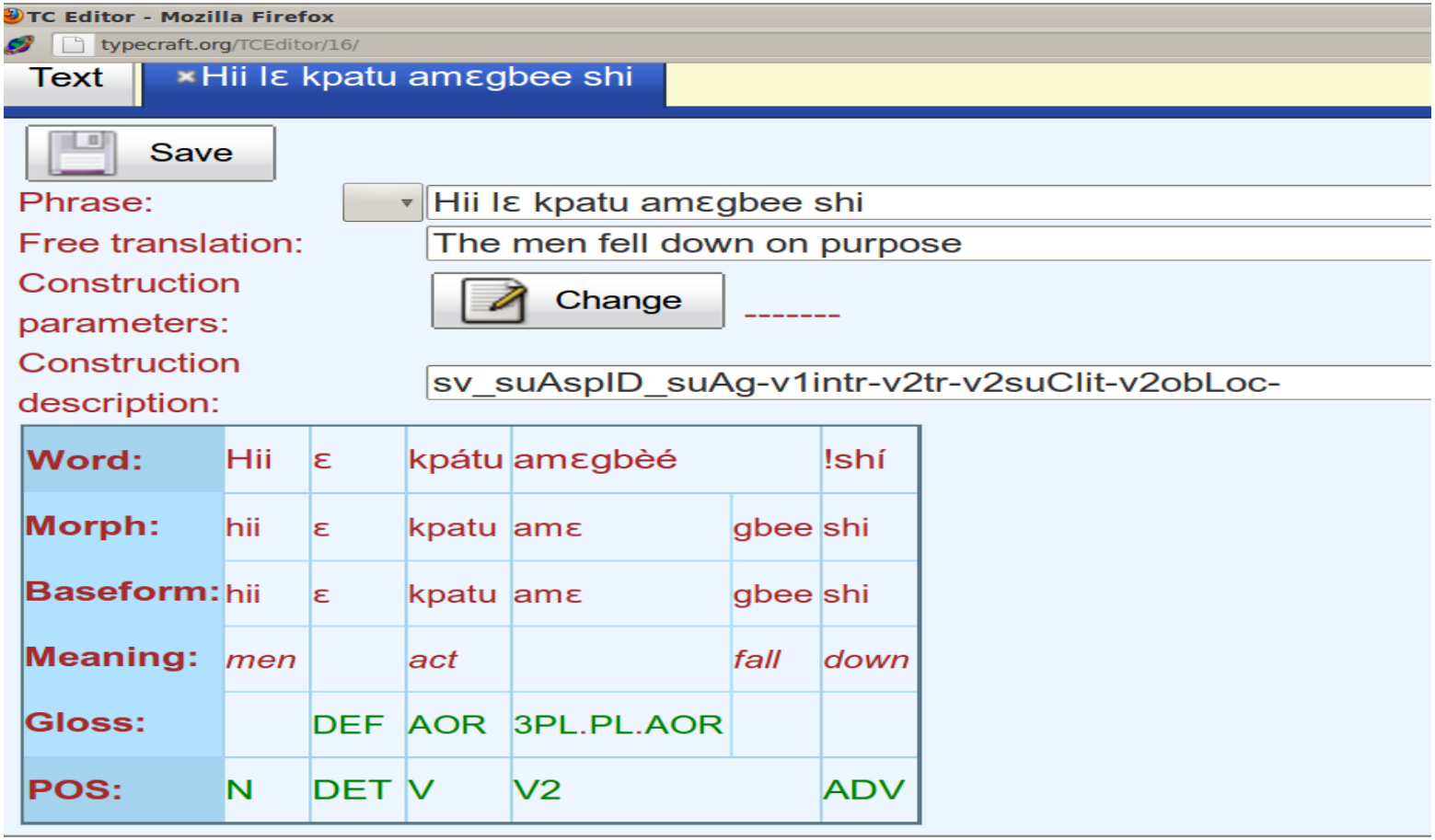
Ex.: He ate the cake

$$\begin{bmatrix} \text{HEAD verb} \\ \text{GF} \begin{bmatrix} \text{SUBJ} \left[\text{INDX } \boxed{1}\left[\text{ROLE agent}\right]\right] \\ \text{OBJ} \left[\text{INDX } \boxed{2}\left[\text{ROLE aff-increm}\right]\right] \end{bmatrix} \\ \text{ASPECT completed} \\ \text{ACTANTS} \begin{bmatrix} \text{ACT1 } \boxed{1} \\ \text{ACT2 } \boxed{2} \end{bmatrix} \\ \text{SIT-TYPE monotonic\_development} \end{bmatrix}$$

To act as a *parser*, any of these induced partial grammars will need to be supplemented by a lexicon and inflectional rules. Rather than try to define such items too in a ‘universal’ repository, we induce them from IGTs of the language in question.
Thus, from IGT, we induce
- a lexicon file for content words (open classes)
- a lexicon file for closed class words
- an inflection rules file

To illustrate, IGT annotations in TypeCraft (TC) are converted into XML format and ported to the grammar under construction. For instance, a perfective verb form like *etee* ‘gone’ with an annotation as indicated in the TC annotation snippet below is assigned a snippet of an XML as below; first a slide showing the annotation interface of TypeCraft:

TypeCraft - Annotation User Interface (Beermann and Mihaylov)



Word	etee
Morph	e | tee
Meaning	| go
Gloss	PERF |
POS	V

```
<word id="30409" text="etee" citation="etee">
    <pos>V</pos>
    <morpheme id="46593" text="eÌ">
        <gloss>PERF</gloss>
    </morpheme>
    <morpheme id="46594" text="tee" meaning="go"/>
</word>
```

tee-v := v-lxm & [STEM <"tee">, ACTNTS.PRED tee_rel].

verb-Perf_irule := %prefix (* e) word & [ASPECT perf, INPUT < v-lxm >].

Meta-strings and meta-items

At this point, we can also introduce what we may call a *meta-grammar* instantiation of these files. In such files, we enter not the actual words and morphemes of the language, but the **gloss** versions of these items, as they are reflected in the IGT. Thus, what we import from the IGTs are not actual morphs but their glosses.

go_v := v-lxm & [STEM <"go"> , ACTNTS.PRED go_rel].

verb-PFV_irule := %suffix (* PFV) word & [ASPECT perf, INPUT < v-lxm >].

Correspondingly, the strings to be parsed by this grammar are ‘meta-strings’, composed exclusively by gloss symbols. Such a string could for instance be

man DEF goPFV

Constructing an e-learning tool from an LKB grammar

The ***Norwegian Online Grammar Sparrer*** is an online language training tool developed at NTNU, with an indirect access point via

http://typecraft.org/tc2wiki/A_Norwegian_Grammar_Sparrer

,

- which provides a general setting and references to various resources on Norwegian, and as direct access point

http://129.241.111.247:8080/norsource/parseStudent .

It can also be reached via a button 'Grammar checker' on each chapter page of the web-based L2 course NoW at NTNU:

http://www.ntnu.edu/now .

TypeCraft

The Natural Language Database

typecraft editor
- New text
- Training templates
- My texts

typecraft search
- Text search
- Phrase search

typecraft corpora
- New corpus

typecraft help
- POS tags list
- Gloss tags list
- HELP Pages
- About TypeCraft
- Type IPA symbols

typecraft information
- Main Page
- Advisory Board
- Bulletin Board
- Current Events
- Research
- Languages

search
Go Search

toolbox
- What links here
- Related changes
- Upload file
- Special pages
- Printable version

A Norwegian Grammar Sparrer

The first small steps in learning a new language reside in mastering the patterns of small sentences and small constructions. The mastery grows partly from recognizing what are the admitted patterns, partly from *using* these patterns over and over again until they sit in one's backbone. When learning a language as a 'second' language, the **Grammar Sparrer** provides you with a limited 'use' environment in which you can get feedback on linguistic patterns of your choice, with repetition and variations and explorations in exactly the directions you want. On clicking on the icon below, you will come to the Sparrer:

Instructions for its use are found at Classroom:Norwegian Grammar Checking

For each expression you type into the Sparrer, you get a response as to whether the string belongs to an admitted pattern, and if not, the Sparrer may tell you what you did wrong. With the facilities of the Grammar Sparrer, you can repeat the same or partly the same material as much as you like, and you can impose ever new variations in the patterns as you like. With intensive sessions, you can both explore and drill the basic patterns of the language.

Behind this Sparrer is a computational grammar of Norwegian, with the sparring function as one of its applications. The 'advices' given are derived from the rules that the grammar applies in order to retract how a string conforms to the language (i.e., when it *parses* the string). Information about the grammar and its present application can be found at Norwegian HPSG grammar NorSource.

The phenomena for which the Sparrer provides feedback messages are found at Grammar sparring phenomena.

For an overview of current feedback messages, go to Feedback messages.

--Lars Hellan 21:15, 11 September 2011 (UTC)

NorMal

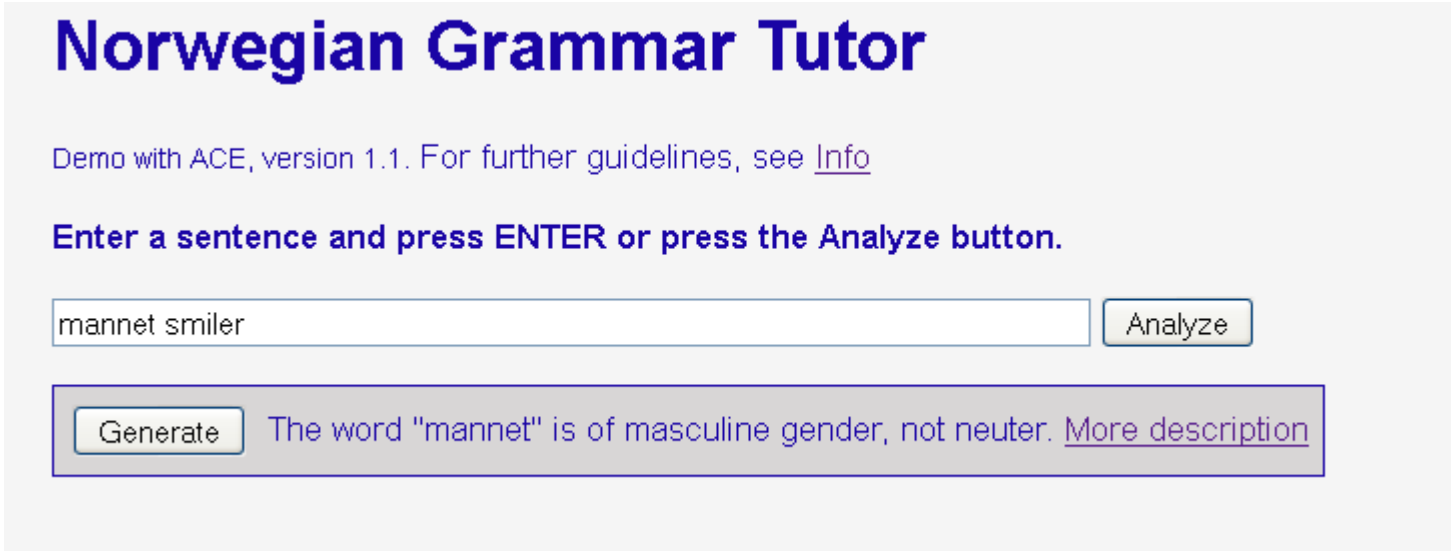
- The system has been created by *Lars Hellan*, *Tore Bruland*, *Elias Aamot* and *Mads Hustad Sandøy*, with ample assistance by Dan Flickinger, starting late in 2010, throughout 2011 and till now, and builds on the computational grammar ***NorSource*** of Norwegian, developed at NTNU since 2001 (see http://typecraft.org/tc2wiki/Norwegian_HPSG_grammar_NorSource). A 'mal-apparatus' is built onto this 'bon'-grammar, together constituting the full system 'NorMal'. (Thus, all files of Norsource are used in NorMal, while NorMal includes files not used in Norsource.)
- The sparrer is accommodated in the TypeCraft web interface.

The Procedure

- For each error sentence, a *recommendation* is generated from the MRS of the NorMal-parsed sentence.
- Both mal-rules and mal-lexical entries introduce into the MRS exactly the same EP(s) as their ’bon’-counterparts generally introduce, whereby generation can produce well-formed strings coming very close to the intended form.

The procedure - 2

- Enter an ungrammatical sentence
- Receive an error message
- Select the first MRS and classify it with Utool
- If the MRS is accepted, a button to generate is displayed



The procedure – 3: Generate to Find Option(s)