Modular HPSG

Vlado Keselj
vkeselj@cs.uwaterloo.ca

presented by Lijun Hou

NLPKE 2001

Modularity in NLP

- **NLP modularity:** dividing NL grammars into smaller modules
- improves NL engineering in the similar way as OOP improves computer programming
- advantages:
  - → managing complexity
  - → parsing efficiency
  - → context-based disambiguation

HPSG

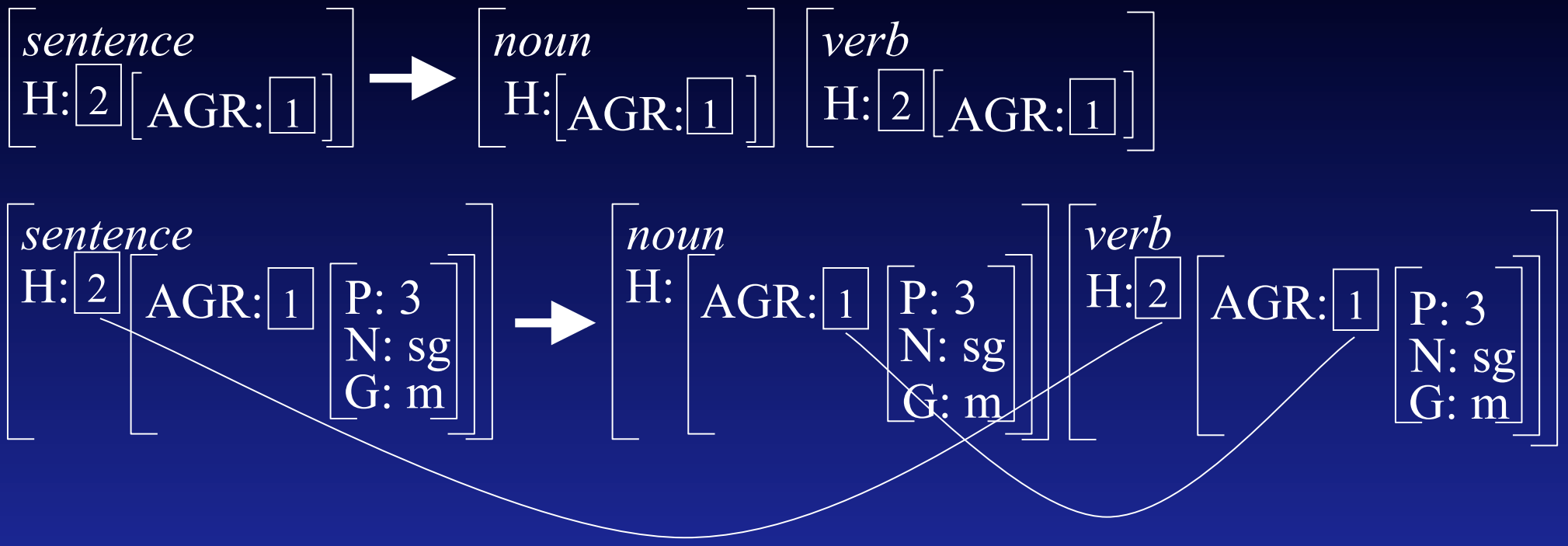
Head-driven Phrase Structure Grammar (HPSG)

tuple (*Atom, Feat, Var, Type, Init, Rule*):

- ⇒ *Atom* - set of atoms
- ⇒ *Feat* - set of features or attributes
- ⇒ *Type* = (*T,* subtype) - type hierarchy
- ⇒ *Init* - set of initial AVMs (attribute-value matrices)
- ⇒ *Rule* - set of rules

HPSG principes are defined and used to define HPSG modules

HPSG mechanism



He

writes.

HPSG Principles

Principles have the same form as HPSG rules:

Unlike HPSG rules, principles are not applied directly.

Instead, principles are unified with HPSG rules:

$P \otimes R = \{\, p \sqcup r : p \in P,\ r \in R,$ and $p \sqcup r$ exists $\} \cup$

$\{\, r : r \in R,$ for all $p \in P,\ p \sqcup r$ does not exist $\}$

$P \otimes R$ is used as the set of rules instead of R

An Approach to HPSG Modularity

Task:

define what is an HPSG module

how are two modules merged to get a resulting module

how an HPSG module defines an HPSG

Approach:

similar to OOP

define *public* and *private* information

Atoms, Variables, Principles, and Initial AVMs

Atoms, principles, and initial AVMs are always public.

When two modules are merged, we make a union of those sets; e.g.: $Atom_1 \cup Atom_2 = Atom$

We assume that all modules use the same set of variables: $Var_1 = Var_2 = Var$

(Variables could be treated in the same way as atoms.)

Features

The set of features is divided in two sets:

a *private* ($Feat^{priv}$) and a *public* ($Feat^{pub}$) set.

They are merged in the following way:

$$Feat^{pub} = Feat_1^{pub} \cup Feat_2^{pub}$$

$$Feat^{priv} = \{(f, M_1) : f \in Feat_1^{priv}\} \cup$$

$$\{(f, M_2) : f \in Feat_2^{priv}\}$$

where M_1 and M_2 are the two merged modules.

Types

Types are treated in a similar way as features:

Types are divided in two sets: T^{priv} and T^{pub}

Merge operation is done in the same way as with features.

Additionally, with types the resulting type hierarchy is defined as:

$$\underline{\text{subtype}} = (\underline{\text{subtype}}_1 \cup \underline{\text{subtype}}_2)^+$$

Two modules can be merged only if the resulting type hierarchy is valid. *(Usually it is.)*

Rules

divided in two sets: *private* (R^{priv}) and *public* (R^{pub})

merged in the following way:

$$R^{pub} = R_1^{pub} \cup R_2^{pub}$$

$$R^{priv} = R_1^{priv} \cup R_2^{priv}$$

The rules for the HPSG produced by the module are obtained in the following way:

$$(P \otimes R^{pub}) \cup R^{priv}$$

where P is the set of principles.

HPSG Modules

An HPSG module is defined to be a tuple:

$$(Atom, Feat^{pub}, Feat^{priv}, Var, T^{pub}, T^{priv}, \underline{subtype}, Init, Rule^{pub}, Rule^{priv}, Prin)$$

It is defined how two such modules are merged.

We also define what is the HPSG defined by an HPSG module:

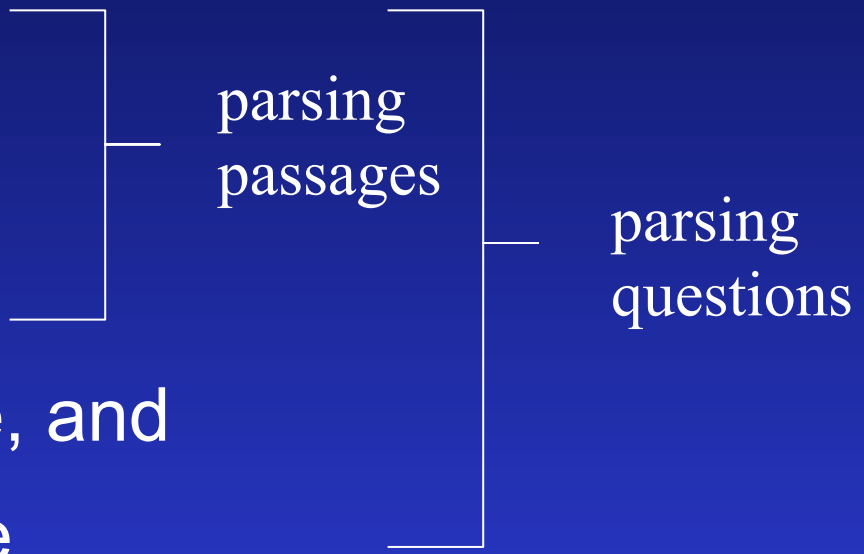
$$(Atom, Feat^{pub} \cup Feat^{priv}, Var, (T^{pub} \cup T^{priv}, \underline{subtype}), Init, (P \otimes R^{pub}) \cup R^{priv})$$

Example

The second part of paper presents an example of application of HPSG modularity.

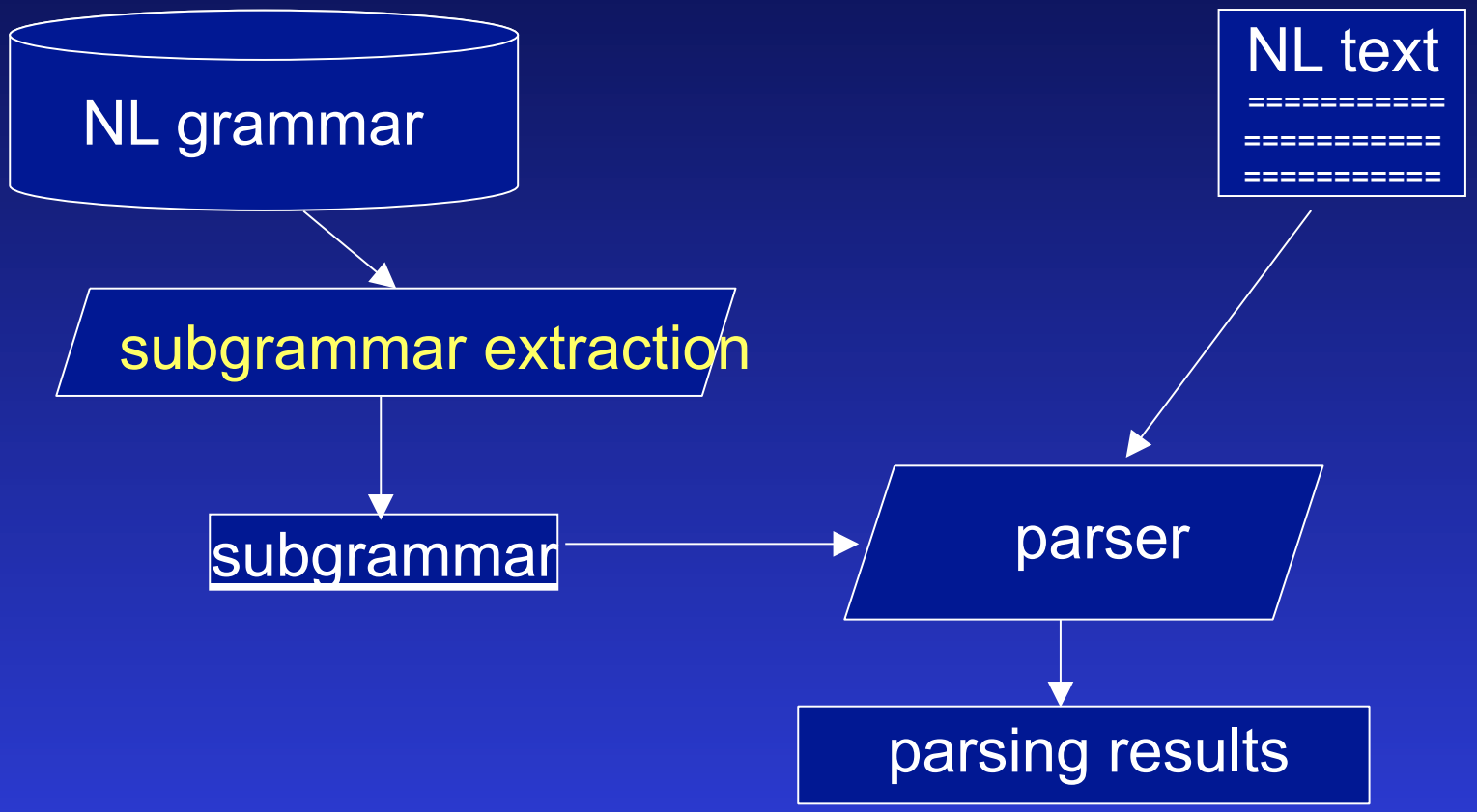
The following five modules are defined:

1. chart module
2. general syntactic module
3. general semantic module
4. question syntactic module, and
5. question semantic module



used in a question-answering system

Just-in-time Subgrammar Extraction



Modularity and Distributed NLP

Modules can be provided from independent sources over the Internet and merged at the point of use.

A parser can be implemented as a Java applet:

⇒ grammar and lexicon are too large to be sent for each access

⇒ a relevant subgrammar can be extracted and sent

⇒ if the parser is used in a dialogue, subgrammars can be sent incrementally and merged at the client side