Syntactic Theory

Typed Feature Structures (TFS)

Yi Zhang, Antske Fokkens

Department of Computational Linguistics
Saarland University

December 8th, 2009

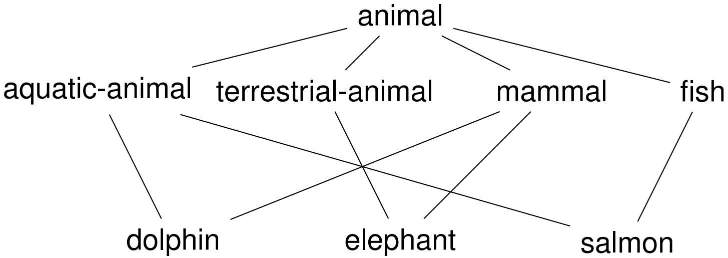
Type Hierarchy

Definition

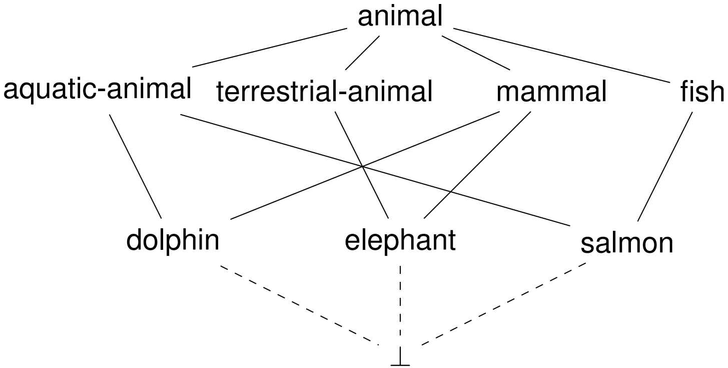
A type hierarchy is a finite bounded complete partial order $\langle \textbf{Type}, \sqsubseteq \rangle$

- A type hierarchy describes a classification of feature structures (and the corresponding linguistic objects modeled by the feature structures)
- Multiple inheritance allows classification on multiple dimensions
- Types are occasionally referred to as **sorts**

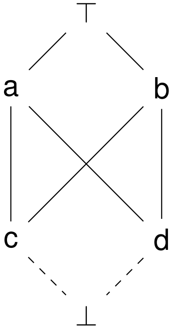
Type Hierarchy: Example



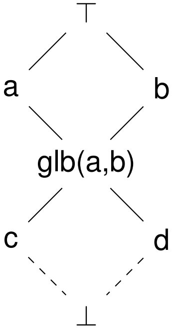
Type Hierarchy: Example



Type Hierarchy: Example (CPO $\Rightarrow$ BCPO)



Type Hierarchy: Example (CPO ⇒ BCPO)



Type Subsumption

For two types $\sigma, \tau \in$ **Type**, if $\sigma \sqsubseteq \tau$, then

- σ **subsumes** τ
- σ is more **general** than τ; τ is more **specific** than σ
- σ is a **supertype** of τ; τ is a **subtype** of σ
- One unique type that subsumes all other types: $*$*top*$*$ $\top$ []
- Types without subtype (other than itself and $\perp$) are called **maximal types** or **leaf types**

- Subsumption relation is a partial order:
  - Reflexive: $\sigma \sqsubseteq \sigma$
  - Antisymmetric: if $\sigma \sqsubseteq \tau$ and $\tau \sqsubseteq \sigma$ then $\sigma = \tau$
  - Transitive: if $\sigma \sqsubseteq \omega$ and $\omega \sqsubseteq \tau$ then $\sigma \sqsubseteq \tau$

Typed Feature Structures

Definition

A typed feature structure is defined on a finite set of features **Feat** and a type hierarchy $\langle \textbf{Type}, \sqsubseteq \rangle$ as a tuple $\langle Q, r, \delta, \theta \rangle$, where:

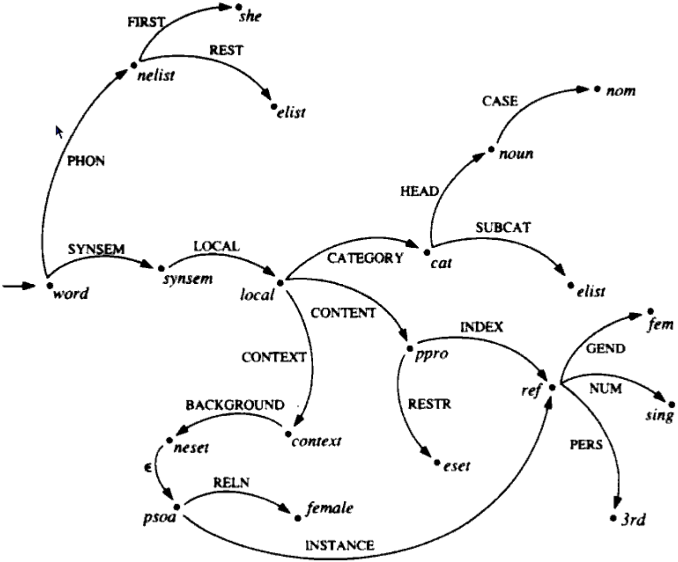
- Q is a finite set of nodes
- $r \in Q$ is the root node
- $\theta : Q \rightarrow \textbf{Type}$ is a total typing function
- $\delta : Q \times \textbf{Feat} \rightarrow Q$ is a partial feature value function

subject to the following conditions:

- r is not a $\delta-$descendant
- all members of Q except r are $\delta-$descendants of r

(*) there is no node n or path π such that $\delta(n, \pi) = n$

Typed Feature Structures: An Example



Reentrancy

- A path is understood as a sequence of features: $\pi \in \textbf{Feat}^+$
- $\delta(n, \pi)$ is the value node starting from n following path π
- If $\delta(r, \pi) = \delta(r, \pi')$ and $\pi \neq \pi'$, i.e. two paths start from the root of the feature structure and point to the same node, then it is said there is a **reentrancy** between path π and π'
- Reentrancy is also called **token identity** or **path equivalence**

Token-Identity v.s. Type-Identity

There is another kind of identity: type identity

Definition

Two nodes n and n' are type-identical when

- $\theta(n) = \theta(n')$
- For any path π, the value of $\delta(n, \pi)$ is defined if and only if the value of $\delta(n', \pi)$ is defined, such that $\theta(\delta(n, \pi)) = \theta(\delta(n', \pi))$

- The identical values in type identity are specified independently; they are two values that happened to look the same
- Token-identical values are achieved by structure sharing, i.e. different paths are pointing to the same node in the TFS

Subsumption of Typed Feature Structures

Definition

F subsumes F', written $F \sqsubseteq F'$, if and only if

- $\pi \equiv_F \pi'$ implies $\pi \equiv_{F'} \pi'$
- $\mathcal{P}_F(\pi) = t$ implies $\mathcal{P}_{F'}(\pi) = t'$ and $t \sqsubseteq t'$

- $\pi \equiv_F \pi'$ means that feature structure F contains path equivalence or reentrancy between the path π and π', i.e. $\delta(r, \pi) = \delta(r, \pi')$ where r is the root node of F
- $\mathcal{P}_F(\pi) = \sigma$ means that the type on the path π in F is σ, in other words $\theta(\delta(r, \pi)) = \sigma$

Constraint Function

Types are associated with constraints expressed as typed feature structures

Definition

Constraint function $C : \langle \mathbf{Type}, \sqsubseteq \rangle \rightarrow \mathcal{F}$ obeys the following conditions

- **Type** For a given type t, if $C(t)$ is the feature structure $\langle Q, q_0, \delta, \theta \rangle$ then $\theta(q_0) = t$
- **Monotonicity** Given type t_1 and t_2, if $t_1 \sqsubseteq t_2$ then $C(t_1) \sqsubseteq C(t_2)$
- **Compatibility of constraints** For all $q \in Q$ the feature structure $C(\theta(q)) \sqsubseteq F' = \langle Q', q, \delta, \theta \rangle$ and $Feat(q) = Appfeat(\theta(q))$
- **Maximal introduction of features** For every feature $f \in \mathbf{Feat}$ there is a unique type t such that $f \in Appfeat(t)$ and there is no type s such that $s \sqsubset t$ and $f \in Appfeat(s)$

Appropriateness of Features

Definition

If $C(t) = \langle Q, q_0, \delta, \alpha \rangle$, then the appropriate features of t are defined as $Appfeat(t) = Feat(\langle F, q_0 \rangle)$ where $Feat(\langle F, q \rangle)$ is defined to be the set of features labeling transitions from the node q in some feature structure F i.e. $f \in Feat(\langle F, q \rangle)$ such that $\delta(f, q)$ is defined

Example

$$_{shirt}\begin{bmatrix} \text{NECK} & \text{length-measure} \end{bmatrix}$$

$$_{trousers}\begin{bmatrix} \text{WAIST} & \text{length-measure} \end{bmatrix}$$

Well-formed Feature Structures

Definition

$F = \langle Q, q_0, \delta, \theta \rangle$ is a well-formed feature structure if and only if for all $q \in Q$, we have that $C(\theta(q)) \sqsubseteq F' = \langle Q', q, \delta, \theta \rangle$ and $Feat(\langle F, q \rangle) = Appfeat(\theta(q))$

Example

Typed feature structures described by the following AVMs are ill-formed

$$_{shirt}\begin{bmatrix} \text{NECK} & 65\text{kg} \end{bmatrix}$$

$$_{trousers}\begin{bmatrix} \text{NECK} & 50\text{cm} \end{bmatrix}$$

Signature

Definition

A signature consists of

- A type inheritance hierarchy $\langle \mathbf{Type}, \sqsubseteq \rangle$
- A corresponding constraint function $C : \langle \mathbf{Type}, \sqsubseteq \rangle \rightarrow \mathcal{F}$

- Linguistic theories are developed by describing the inheritance type hierarchy together with proper constraints
- A constraint-based grammar framework

Attribute-Value Matrix (AVM)

Attribute-value matrix (AVM) notation is a **description language** to describe sets of feature structures, with the following three building blocks

- **Type** descriptions selects all objects of a particular type
- **Attribute-value pairs** describe objects that have a particular property. The attribute must be appropriate for the particular type, and the value can be any kind of description
- **Tags** to specify **token identity**

$$
{}_{t1}\begin{bmatrix} \text{F1} & \mathrm{t2} \\ \text{F2} & \boxed{1}\; {}_{t3}\begin{bmatrix} \text{F4} & \mathrm{t2} \end{bmatrix} \\ \text{F3} & \boxed{1} \end{bmatrix}
$$

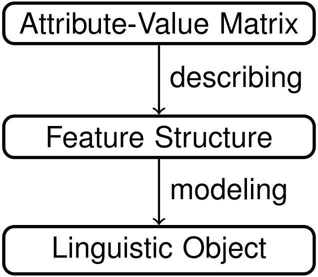
Attribute-Value Matrix (AVM) cont.

- Attribute-Value Matrix (AVM) is used to describe feature structures
- The order of the rows is not important
- Each attribute can only take one value, hence the following AVM is **improper** and does NOT describe any feature structure

$$\begin{bmatrix} \text{NAME} & \text{Sandy} \\ \text{AGE} & 29 \\ \text{AGE} & 30 \end{bmatrix}_{\textit{person}}$$

- It is common practice to refer to AVMs as “feature structures”, although strictly speaking they are **feature structure descriptions**

Feature Structure v.s. Feature Structure Description



- Linguistic objects are modeled by feature structures, they are total with respect to the ontology declared in the signature. Technically, one say that these feature structures are
  - **Totally well-formed**: every node has all the attributes appropriate for its type and each attribute has an appropriate value
  - **Type-resolved**: every node is of a maximally specific type
- Each AVM can partially describe a set of feature structures by underspecifying information

Unification of Typed Feature Structures

Definition

The unification $F \sqcup F'$ of two feature structures F and F' is the greatest lower bound of F and F' in the collection of feature structures ordered by subsumption

Definition

The well-formed unification $F \sqcup_{wf} F'$ of two feature structures F and F' is the greatest lower bound of F and F' in the collection of well-formed feature structures ordered by subsumption

- Unification is the only operation used to process TFSes
- Grammars developed in such frameworks are called unification-based grammars

Unification of Typed Feature Structures (cont.)

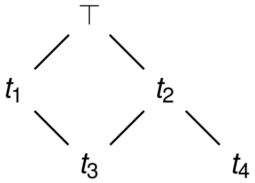
- A special symbol $\perp$ (bottom) is introduced to denote the failed unification of two incompatible feature structures
- Conceptually, $\forall \sigma \in$ **Type** $\quad \sigma \sqsubseteq \perp$
- The type hierarchy (including $\perp$) is assumed to be a **bounded complement partial order (BCOP)**, so that unification operation is deterministic (glb exists for any pair of types)
- $\sigma \sqsubseteq \tau \Leftrightarrow \sigma \bigsqcup \tau = \tau$

Unification of AVMs

- As FSD, the unification of two AVMs A_1 A_2 results in a new AVM A_3 that describes the intersecting set of typed feature structures described by both AVMs A_1 A_2

Example

1. ${}_{t_4}\begin{bmatrix} \mathrm{F}_1 & \mathrm{t}_1 \end{bmatrix} \sqcup {}_{t_4}\begin{bmatrix} \mathrm{F}_1 & \mathrm{t}_2 \\ \mathrm{F}_2 & \mathrm{t}_3 \end{bmatrix} = {}_{t_4}\begin{bmatrix} \mathrm{F}_1 & \mathrm{t}_3 \\ \mathrm{F}_2 & \mathrm{t}_3 \end{bmatrix}$
2. ${}_{t_4}\begin{bmatrix} \mathrm{F}_1 & \mathrm{t}_1 \\ \mathrm{F}_2 & \mathrm{t}_2 \end{bmatrix} \sqcup {}_{\top}\begin{bmatrix} \mathrm{F}_1 & \boxed{1} \\ \mathrm{F}_2 & \boxed{1} \end{bmatrix} = {}_{t_4}\begin{bmatrix} \mathrm{F}_1 & \boxed{1}\ \mathrm{t}_3 \\ \mathrm{F}_2 & \boxed{1} \end{bmatrix}$
3. ${}_{\top}\begin{bmatrix} \mathrm{F}_1 & \boxed{1}\ \mathrm{t}_1 \\ \mathrm{F}_2 & \boxed{1} \end{bmatrix} \sqcup {}_{\top}\begin{bmatrix} \mathrm{F}_2 & \boxed{2} \\ \mathrm{F}_3 & \boxed{2}\ \mathrm{t}_4 \end{bmatrix} = \perp$



Lists in Typed Feature Structures

- Ordered list can be described using the following type hierarchy and constraints

list(σ)

elist

nelist(σ)

$$\left[\begin{array}{ll} \text{FIRST} & \sigma \\ \text{REST} & \text{list}(\sigma) \end{array}\right]_{\textit{nelist}}$$

- For convenience, we use notation $\left\langle \text{A, B, C} \right\rangle$ to denote

$$\left[\begin{array}{ll} \text{FIRST} & \text{a} \\ \text{REST} & \left[\begin{array}{ll} \text{FIRST} & \text{b} \\ \text{REST} & \left[\begin{array}{ll} \text{FIRST} & \text{c} \\ \text{REST} & \text{elist} \end{array}\right]_{\textit{nelist}} \end{array}\right]_{\textit{nelist}} \end{array}\right]_{\textit{nelist}}$$

Sets in Typed Feature Structures

- Type *set*(σ) is used to describe **sets** of feature structures of type σ
- Notation $\{a, b, c\}$ is used to describe set membership
- Formally introducing **sets** in typed feature structures involves a fair amount of technical complications ([Carpenter, 1992])
- For our purpose, an intuitive understanding is sufficient
- In some implementations, lists are used to simulate sets

References I

Carpenter, B. (1992).
The Logic of Typed Feature Structures.
Cambridge University Press, Cambridge, UK.

Copestake, A. (2000).
Definitions of typed feature structures.
Natural Language Engineering (appendix to special issue on efficient processing with HPSG), 6(1).