Appendix B

Glossary

The purpose of this appendix is to provide a collection of definitions about the set of different concepts involved in Noos language as well as a reference to the section of the memory in which are defined.

In this glossary, terms appearing in **boldface** indicate they are defined in the glossary. Often we will use *term* as a shorthand for *feature term* (and will not appear in boldface).

Anonymous terms — A term that has no **identifier**, it can be referenced only by its position (as a subterm of other terms). For instance, in `(define (person Jack) (girlfriend (define (girl))))` the girlfriend of Jack is anonymous and can only be referenced by `(>> girlfriend of Jack)`. (§ 3.2.2).

Antiunification — A basic operation defined in Noos that given two terms constructs another term holding *which is common to both* (yielding the notion of generalization) and *all that is common to both* (the most specific generalization). Formally, the antiunification of a set of terms yields a greatest lower bound with respect to **subsumption** ordering. (§ 4.6).

Any — The highest sort in the Noos refinement hierarchy. `Any` represents the minimum information and all the other sorts are more specific than `any`. See also **single description**. (§ 3.2.2).

Built-in method — A Noos predefined method. Examples of Noos built-in methods are arithmetic operations, set operations, logic operations, operations for comparing feature terms, and other basic constructs such as conditional or sequencing. (§ 3.2.4, § D).

Domain knowledge — Specifies a set of concepts, a set of relations among concepts, and problem data that are relevant for an application. Concepts and relations define the domain ontology of an application. (§ 3.1).

Constant term — A feature term that is not a **method**–i.e. is not evaluable. (§ 5.2).

Constituent — The constituent of a term `T` is the identifier of a term `T'` from which `T` has been constructed by **refinement**. (§ 3.2.2).

Current-task — A reflective operation that allows to refer to the task in which a method is involved. (§ 3.3.5).

Default — Reflective operator that allows to refer to the **default metalevel** term of a term. § 3.3.5.

Default metalevel — A special kind of **metalevel** that applies to all the features of its **referent**. In a default metalevel we can specify a method (or a set of methods) for *any* feature of a **referent**. (§ 3.3.2).

Description — The syntax Noos uses for constructing **feature terms**. The description syntax is based on lists (like Lisp) starting with token `define`. For instance, `(define (person man))`. (§ 3.2.1).

Episodic knowledge — The reification of part of the behavior of the system represented in Noos. Episodic knowledge is is organized in **episodic models** and stored in the **episodic memory**. (§ 3.1, § 4.1).

Episodic memory — Episodic memory is the (accessible and retrievable) collection of the episodic models of the problems that a system has solved. (§ 4.1).

Episodic model — The *explanation* of the inference process engaged by Noos in solving a specific **problem task**. An episodic model holds the set of knowledge pieces used for solving a specific problem task, how and where they were used, and the decisions taken for solving that problem. (§ 4.1).

Ephemeral term — A term that is not memorized—and thus, it is not amenable to **retrieval**. (§ A).

Evaluable term — A kind of feature term that models Noos methods. Evaluable feature terms are interpreted as functions. (§ 5.9).

Failure — In a specific subtask, a failure occurs when no value for that task can be inferred—and causes backtracking at that point. When a global failure occurs, the token `fail` is returned. (§ 3.5).

Feature term — The basic data structure of Noos. They can be seen as a generalization of first order terms and lambda terms. They are extendable records organized in a **subsumption** hierarchy. Feature terms are constructed by means of **descriptions** and the **refinement** constructor. (§ 5.2, § 5.6).

Identifier — An identifier is a symbol denoting a term. Terms that have an identifier are **named terms**, otherwise they are **anonymous terms**. (§ 3.2.2).

Introspection — In Noos, the capability of accessing to and reasoning about the episodic memory.

Labeled graph representation — A term can be represented as a labeled directed graph that has, for each variable $X : s$, a node q labeled with sort s, and has an arc from q to another node q' labeled by f, for each feature f defined in q with feature value q'. (§ 5.7).

Matching — See **subsumption**.

Memorization — The property of **permanent terms** to be stored in **episodic memory** and be amenable to **retrieval**. (§ 4.1).

Meta — Reflective operator that allows to access to the metalevel term of a term. (§ 3.3.5).

Metalevel — A feature term in a metalevel relationship with a **referent** term `B`. The features of the referent B have a corresponding feature with the same name on the metalevel. A feature f of the metalevel has as feature value the set of methods methods that are *applicable* to the feature f of the referent B. (§ 3.3.1).

Metalevel knowledge — Knowledge about **domain knowledge**, **problem solving knowledge**, and **episodic knowledge**. Metalevel knowledge is formed by metalevel concepts, metalevel relations, metalevel tasks, and metalevel methods. Moreover, metalevel knowledge includes **preferences**. (§ 3.1, § 3.3).

Method — An evaluable feature term. Formally, a function that receives parameters by feature names. A method term is *closed* when it possess all required parameters. Methods are defined by refinement. (§ 3.2.4).

Named Term — A term with an **identifier**. A named term can be referenced by its identifier using `(>> of NamedTerm)` although syntactic sugar allows in some places to write only `NamedTerm`. (§ 3.2.2).

Node — See **labeled graph representation**.

Path reference — A list that starts with the `>>` token. There are two kinds of path references: absolute and relative path references. An example of an absolute path reference is `(>> symptom of car)`. An example of a relative path reference is `(>> price model)`. (§ 3.2.3).

Permanent terms — **Feature terms** that are memorized (see **memorization**) and, thus, amenable to **retrieval**. (§ A).

Perspective — A mechanism for describing declarative biases for Noos retrieval. (§ 4.3).

Preferences — Model decision making about sets of alternatives present in **domain knowledge** and **problem solving knowledge**. Furthermore, preferences are used in Noos as a symbolic representation of relevance in comparing a given current problem with problems previously solved by the system. (§ 3.4).

Problem solving knowledge — Problem solving knowledge specifies a set of tasks and methods that construct a model of a problem (solve a problem). For a given subtask there may be multiple alternative methods that may be capable of solving that subtask in different situations. A method can be decomposed into subtasks that may be achieved by other methods. (§ 3.1).

Problem task — A task engaged by a **query expression**. A problem task `F(D)` engages the inference to determine the feature value for feature `F` of feature term `D`. (§ 3.2.5).

Query expression — A question that is posed to the system. Usually, a query expression is asking for a feature value, as in `(>> diagnosis of patient-33)`. Since evaluation in Noos is lazy, no feature values are inferred until a query is performed—and only those feature values needed will be computed. (§ 3.2.5).

Reference — See **identifier** and **path reference**.

Referent — The referent of a **metalevel** is that term it is metalevel of. The referent of a **task** is the result value of that task. Furthermore, the referent reflective operator allows to access to the referent—if it exists—of a term. (§ 3.3.1, § 3.3.5).

Refinement — A constructor operation that builds a term from another (already defined) term. Refinement involves two distinct aspects: (1) code reuse (the construction of a term by reusing another term) and, (2) subtyping (the definition of a domain-specific **sort** hierarchy). (§ 3.2.2).

Reflective operator — An operation that allows to access and to inspect the **metalevel knowledge** (see operations namely **meta**, **default**, **task**, **current-task**, and **referent**). (§ 3.3.5).

Reification — The process by which a Noos expression is converted into an object (a feature term). Reification is performed by the **reify** construct. (§ 3.3.6).

Reify — A construct that takes a path reference or a compact description and builds an **method** that reifies it.

Retrieval — A mechanism for content-based access to the episodic memory. Noos provide a set of basic retrieval methods. Retrieval methods allow to retrieve previous relevant episodes from the **episodic memory** using relevance criteria. (§ 4.2).

Root — The root **node** of a feature term is the outmost **node** in a **description**. For instance, the root in the following description `(define (person :id Jack) (girlfriend (define (girl))))` is the node of sort `person` with identifier `Jack`. (§ 3.2.2).

Single description — Compact syntax for defining feature terms by refinement of the top sort **any**. A description such as `(define foo)` is just a short syntax equivalent to `(define (any foo))`. (§ 3.2.2).

Sort — A symbol that denotes a set of the individuals of a domain. Sorts form a collection of partially ordered symbols. A set of predefined sorts are defined in Noos. New sorts are defined using **refinement**. In Noos the top sort is called **any**. (§ 3.2.2, § 5.2.2, § A).

Subsumption — Informational ordering among feature terms. We say that a feature term ψ subsumes another feature term ψ' when all information in ψ is also contained in ψ'—or in other words, ψ is more general than ψ'. (§ 5.6).

Symbol — See **identifier**.

Task — Tasks reify the status of the inference in the language. A given task reifies the inference status for a feature of a term. Tasks embody episodic knowledge such as the method that has succeeded in achieving that task (the method used to infer the feature value of the feature) and the result of the evaluation of the method (the feature value). (§ 3.3.4).

Transparent methods — Noos methods are *transparent*. The transparent capability of Noos methods allows to perform forms of inference that need to inspect and reason about methods and how they have been used to solve particular tasks. This capability of Noos methods is used, for instance, in analytical learning methods. (§ 4.1).