Annex D: Signature-Free and Sorted Dialects

This annex specifies the syntax and semantics of *signature-free* and *sorted* dialects of Common Logic. All such dialects are specified in terms of a subset of Common Logic called SCL. Every text *t* in a sorted dialect has a *sort structure* that restricts the set of admissible interpretations of *t*. A signature-free dialect, however, does not impose any restrictions on the interpretations of any text.

D.1 Abstract syntax of SCL

The abstract syntax for SCL has a single universe, and it does not specify methods for importing or titling texts. Section D.3 specifies methods for importing and combining texts in various dialects, but a semantic interpretation is defined only on the text that results from a successful combination.

D.1.1 *Names, sentences,* and *texts* are well-formed expressions.

D.1.2 A text is a set of zero or more *sentences* and *comments*.

D.1.3 A sentence is an *atom*, a *Boolean sentence*, or a *quantified sentence*.

D.1.4 An atom is either an equation containing two arguments, which are *terms*, or an atomic sentence, which consists of a term, called the predicate, and a *term sequence* called the argument sequence, the elements of which are called arguments of the atom.

D.1.5 A Boolean sentence has a type, called a connective, and a number of sentences called the components of the Boolean sentence. There are five types of Boolean sentences: conjunctions and disjunctions have zero or more components; implications and biconditionals have two components; and negations have one component.

For every interpretation, the semantics of Section D.2 evaluates a conjunction of zero sentences as true. It evaluates a disjunction of zero sentences as false.

D.1.6 A quantified sentence has (i) a type, called a quantifier, (ii) a finite, nonrepeating sequence of names and sequence markers called the binding sequence, each element of which is called a binding of the quantified sentence, and (iii) a sentence called the body of the quantified sentence. The two types of quantified sentence are the universal and the existential. A name or sequence marker which occurs in the binding sequence is said to be bound in the body. Any name or sequence marker which is not bound in the body is said to be free in the body.

D.1.7 A term is either a name or a *functional term*.

D.1.8 A functional term consists of a term, called the *operator*, and a term sequence called the argument sequence, the elements of which are called arguments of the functional term.

D.1.9 A term sequence is a finite sequence of terms or sequence markers.

Note: Term sequences may be empty. An atomic sentence with zero arguments may be used to represent a proposition. A functional term with zero arguments may be used to represent a constant.

D.1.10 A lexicon is a union ($V \cup S$) of two disjoint sets: V of names and S of sequence markers.

D.1.11 A comment is data that shall be designated as a comment and may be attached to other comments or to well-formed expressions. There are no restrictions on the data of a comment.

D.2 Semantics of the SCL abstract syntax

The semantics of the SCL abstract syntax is defined in terms of a satisfaction relation between a text and mathematical structures called *interpretations*.

D.2.1 An interpretation *I* of the abstract syntax with lexicon V∪S is a set U_I, called the universe of *I*, and mappings for the categories of relations, functions, names, and sequence markers:

- For relations, rel_I maps U_I to the set U_I^* of finite sequences of elements of U_I. The set U_I^* includes the empty sequence < >, which contains no elements of U_I.
- For functions, fun_I maps U_I to total functions from U_I^* to U_I.
- For names, int_I maps V to U_I.
- For sequence markers, seq_I maps S to U_I^*.

Note: In some dialects, free names and bound names may have the same spelling. In SCL, a name in any binding shall be distinct from any free name or any name in a different binding that may have a similar spelling. Any translation from a dialect to SCL shall preserve that distinction.

D.2.2 For every interpretation *I* and every name *n* in V, the element $int_I(n)$ in U_I is called the denotation of *n*. For every interpretation *I* and every element *x* in the universe U_I, the set $rel_I(x)$ is called the relational component of *x*, and the function $fun_I(x)$ is called the functional component of *x*. The relational component of *x* is called vestigial iff $rel_I(x)$ is the empty set; the functional component of *x* is called vestigial iff $fun_I(x)$ is $int_I(x)$ for any sequence of zero or more arguments.

D.2.3 For any interpretation *I* and expression E, the semantic value I(E) is determined by the rules in Table 1.

Table 1 requires two auxiliary definitions: *concatenated sequences* are used in rows E3 and E4, and *N-variants* of interpretations are used in rows E13 and E14.

D.2.4 If $s = \langle s_1, ..., s_n \rangle$ and $t = \langle t_1, ..., t_m \rangle$ are finite sequences, then $s;t$ is the concatenated sequence $\langle s_1, ..., s_n, t_1, ..., t_m \rangle$. A concatenation of any sequence *s* with < > is identical to *s*: $s;\langle\rangle = s$.

D.2.5 Let N be a subset of a lexicon V∪S. An interpretation *J* of V∪S is an N-variant of *I* if *J* is exactly like *I* except that int_J and seq_J may differ from int_I and seq_I in their assignments to elements of N. Formally, *J* is an N-variant of *I* if $U_J = U_I$, $rel_J = rel_I$, $fun_J = fun_I$, $int_J(n) = int_I(n)$ for any name *n* not in N, and $seq_J(s) = seq_I(s)$ for any sequence marker *s* not in N.

Table 1 does not specify an interpretation for comments. The interpretation of any expression with an attached comment is the same as the interpretation of that expression without the comment. Adding or deleting comments cannot change the truth-conditions of any sentence or text. Nevertheless, comments are part of the formal syntax. Applications should preserve comments and their attachments when importing or translating texts in any dialect.

The logic that conforms to the abstract syntax as specified in Section D.1 and to the interpretations of Table 1 shall be called SCL.

Table 1 – Interpretation of expressions in the abstract syntax

	If E is an expression of the form	**Then *I*(E) =**
E1	name **N**	$int_I(\mathbf{N})$
E2	sequence marker **S**	$seq_I(\mathbf{S})$
E3	term sequence $\mathbf{T}_1$ ... $\mathbf{T}_n$ with $\mathbf{T}_1$ a term	$\langle I(\mathbf{T}_1)\rangle ; I(\langle \mathbf{T}_2 \ \ldots \ \mathbf{T}_n \rangle)$
E4	term sequence $\mathbf{T}_1$ ... $\mathbf{T}_n$ with $\mathbf{T}_1$ a sequence marker	$I(\mathbf{T}_1) ; I(\langle \mathbf{T}_2 \ \ldots \ \mathbf{T}_n \rangle)$
E5	term with operator **O** and argument sequence **S**	$fun_I(I(\mathbf{O}))(I(\mathbf{S}))$
E6	Atom which is an equation containing terms $\mathbf{T}_1$, $\mathbf{T}_2$	true if $I(\mathbf{T}_1) = I(\mathbf{T}_2)$, otherwise false
E7	Atomic sentence with predicate **P** and argument sequence **S**	true if $I(\mathbf{S})$ is in $rel_I(I(\mathbf{P}))$, otherwise false
E8	Boolean sentence of type negation and component **C**	true if $I(\mathbf{C})$ = false, otherwise false
E9	Boolean sentence of type conjunction and components $\mathbf{C}_1$ ... $\mathbf{C}_n$	true if n=0 or if $I(\mathbf{C}_1) = ... = I(\mathbf{C}_n)$ = true, otherwise false
E10	Boolean sentence of type disjunction and components $\mathbf{C}_1$ ... $\mathbf{C}_n$	false if n=0 or if $I(\mathbf{C}_1) = ... = I(\mathbf{C}_n)$ = false, otherwise true
E11	Boolean sentence of type implication and components $\mathbf{C}_1$, $\mathbf{C}_2$	false if $I(\mathbf{C}_1)$ = true and $I(\mathbf{C}_2)$ = false, otherwise true
E12	Boolean sentence of type biconditional and components $\mathbf{C}_1$, $\mathbf{C}_2$	true if $I(\mathbf{C}_1) = I(\mathbf{C}_2)$, otherwise false
E13	quantified sentence of type universal with bindings **N** and body **B**	true if for every **N**-variant J of I, $J(\mathbf{B})$ is true, otherwise false
E14	quantified sentence of type existential with bindings **N** and body **B**	true if for some **N**-variant J of I, $J(\mathbf{B})$ is true, otherwise false
E15	sentence **S**	$I(\mathbf{S})$
E16	A text containing the sentences $\mathbf{S}_1$ ... $\mathbf{S}_n$	true if n=0 or if $I(\mathbf{S}_1) = ... = I(\mathbf{S}_n)$ = true, otherwise false

D.3 Common Logic dialects

Syntactically, dialects of Common Logic may have an open-ended variety of notations, linear or graphic. Semantically, Common Logic supports two kinds of dialects: *signature free* and *sorted*.

D.3.1 Signature-free dialects of Common Logic

As a signature-free logic, the SCL abstract syntax imposes no restrictions on names. The semantics of SCL allows any name, free or bound, to refer to anything in the universe U. The syntax of SCL allows any name to occur in relation position, function position, or argument position. For example, the sentence $x(x(x))$ uses x in all three ways.

D.3.1.1 A logic L shall be called *well formed* if it has the following properties:

- The syntax of L shall designate four kinds of well-formed expressions (WFEs): names, sentences, texts, and comments.
- Every syntactic feature of L shall be a WFE or a component of a WFE. Some syntactic categories of L may be designated as more than one kind of WFE.
- For every interpretation I with universe U, $I(n) \in U$ for any name n; $I(s) \in \{\text{true,false}\}$ for any sentence s; $I(t) \in \{\text{true,false}\}$ for any text t.
- Comments may be attached to WFEs. But they shall have no interpretation, and they shall not affect the interpretation of other WFEs.

Definition D.3.1.1 accommodates a wide range of notations. Some predicate logics, for example, don't have functions; they may represent a function as a kind of relation. Some equational logics don't have relations; they use functions and equations to represent the equivalent of relations. A propositional logic L has no functions, relations, or equations. For any proposition p in L, a translation to SCL would be an atomic sentence with a relation named p that has no arguments.

If L is a graph logic, a graph in L may be translated to or from a text, a sentence, or a conjunction of sentences in a linear notation. An equation between two variables in a linear notation may be represented in a graph by mapping both variables to the same node. A combinatorial logic may have no variables at all.

D.3.1.2 For any well-formed logics L and L′, a translation function f from L to L′ shall have the following properties:

- For every WFE e of L of kind k, $f(e)$ shall be a WFE of L′ of kind k.
- For every interpretation J of every WFE e of L, there exists an interpretation I of $f(e)$ such that $I(f(e)) = J(e)$.
- For every WFE e of L and every interpretation I of $f(e)$ there exists an interpretation J of e such that $I(f(e)) = J(e)$.

D.3.1.3 A well-formed logic with a translation function to SCL is called a signature-free dialect.

D.3.1.4 A signature-free dialect L is equivalent to SCL if there is a translation function from SCL to L. If some proper subset of L is equivalent to SCL, then L is a syntactic extension of SCL.

As an example, consider a Boolean operator such as *xor* (exclusive-or). With a definition of *xor* in terms of *and*, *or*, and *not*, an equivalent SCL dialect would become a syntactic extension of SCL.

D.3.2 Sorted dialects of Common Logic

A *sorted logic* is a well-formed logic with a sort structure that restricts the *admissible* interpretations. With an added sort structure, any signature-free SCL dialect may become a *sorted* SCL dialect.

D.3.2.1 A sorted logic L is a well-formed logic with a sort structure (S,C) and a signature function *sgn*:

- S shall be a set of sorts. For each sort *s*, there shall be a sort relation *r*, such that $s = \{x \mid r(x)\}$.
- C shall be a set of sentences, called sort constraints, expressed in a well-formed logic distinct from L. For every name *n* that occurs free or bound in any text T of L, *sgn*(*n*) shall be a sort constraint called the signature of *n*. C may contain sentences that are not signatures of any name.
- For every text T of L, an interpretation *I* of T shall be admissible iff *I*(C) is true. For any *I* that is not admissible, *I*(T) shall be undefined.
- If S has N sorts, L is called an N-sorted logic. If N≥2, L is called a many-sorted logic (MSL). If S is partially ordered, L is called an order-sorted logic (OSL).

A sorted logic is a hybrid: a text C in one logic restricts the admissible interpretations of a text T in another logic. By using the equivalent of sort relations, a signature-free logic may enforce some, but not all, the restrictions of a sorted logic. As an example, any interpretation that makes the sorted sentence A true would also make the signature-free sentence B true:

A: $(\forall x{:}\text{Elephant})(\exists y{:}\text{Trunk})\text{HasPart}(x,y)$

B: $(\forall x)(\exists y)(\text{Elephant}(x) \supset (\text{Trunk}(y) \wedge \text{HasPart}(x,y)))$

But sentence B may have interpretations that are not admissible for A. For example, the assignment of an apple to *x* and a banana to *y* would make B true, but A false. In a sorted logic, the signatures of the bound variables *x* and *y* would block those unwanted assignments.

For some sort constraints, a syntax checker may detect possible violations. For a name *f*, the signature *sgn*(*f*) may specify its sort as Function and restrict the sorts of its arguments and value. In mathematics, such a signature is typically expressed in the following notation:

$f\colon \text{A}_1 \times \ldots \times \text{A}_\text{N} \rightarrow \text{A}_{\text{N+1}}$.

Each A names a sort. The first N sorts restrict the arguments of *f*, and the last sort restricts the value. If the sorts are disjoint, a syntax checker may use that signature to verify that every occurrence of *f* shall satisfy the constraints. For an OSL, however, the sort constraints usually require semantic tests.

D.3.2.2 A sorted SCL dialect L shall be an SCL dialect with a sort structure (S,C) and a pair of translation functions (f,f'). For any text T in L,

- For every name *n* in T, the signature *sgn*(*n*) in C shall be an *SCL signature*.
- Both f(C) and f'(T) shall be texts in SCL.
- For every text T of L, an interpretation *I* shall be admissible iff *I*(f(C)) = true. If *I* is admissible, *I*(T) shall be defined as *I*(f'(T)); otherwise, *I*(T) is undefined.

For any name *n*, an SCL signature may restrict the sort, the relational component, and the functional component of *I*(*n*). For example, the following CLIF sentence may be used as a *function restriction* that requires any argument or value of a function named *plus* to be of sort Real:

```
(forall (x1 x2 x3)
  (if (= (plus x1 x2) x3) (and (Real x1) (Real x2) (Real x3))))
```

A *relation restriction* for plus may be defined in terms of its function restriction:

(forall (x1 x2 x3)
  (if (plus x1 x2 x3) (= (plus x1 x2) x3)))

The name *divide* may have a function restriction that prohibits any assignment of 0 to argument x2:

(forall (x1 x2 x3)
  (if (= (divide x1 x2) x3) (and (Real x1) (Real x2) (Real x3) (not (= x2 0)))))

For any text, a syntax checker can verify the restrictions on the name *plus*. But the restriction x2≠0 is a semantic restriction. A syntax checker can verify some, but not all semantic restrictions.

D.3.2.3 For any name *n*, an SCL signature for *n* is a conjunction of three sentences: a *sort restriction sr*, a *relation restriction rr*, and a *function restriction fr*. The unrestricted signature for any name *n* shall be be a conjunction of the unrestricted *sr*, *rr*, and *fr*.

- A sort restriction *sr* for *n* by sort *s* shall be the atomic sentence $r(n)$, where *r* is the sort relation for *s*. If *sr* is unrestricted, $r(x)$ shall be true for every *x* in the universe of any interpretation.
- A relation restriction *rr* for *n* shall be a Boolean sentence whose non-Boolean components shall have the form $(\forall ...x)(n(...x) \supset ac)$, where *...x* represents a sequence of zero or more names used as the arguments of $n(...x)$ and *ac* is a sentence called the argument constraints. The unrestricted relation restriction shall be a conjunction of zero sentences, which is true in all interpretations.
- A function restriction *fr* for *n* shall be a Boolean sentence whose non-Boolean components shall have the form $(\forall v, ...x)((v = n(...x)) \supset avc)$, where *...x* represents a sequence of zero or more names used as the arguments of $n(...x)$, *v* is a name that represents the value of the function *n*, and *avc* is a sentence called the argument and value constraints. The unrestricted function restriction shall be a conjunction of zero sentences, which is true in all interpretations.
- If the signature of *n* states that the relational (functional) component of *n* is vestigial, any use of *n* in relation (function) position may be eliminated by a truth-preserving substitution.

D.3.2.4 For any sorted logic L with sort structure (S,C) and any sorted logic L′ with sort structure (S′,C′), a sorted translation function f from L to L′ shall have three components: $f = (f_1,f_2,f_3)$.

- The mapping f_1 shall be an isomorphism from S to some subset of S′.
- The mapping f_2 shall be a translation function from C to C′.
- The mapping f_3 shall be a translation function from any text in L to a text in L'.

D.2.3.5 If f is a sorted translation function f from L to L′, Definition D.2.3.4 implies that

- For every text T in L and every admissible interpretation *I* of f_3(T) there shall exist an admissible interpretation *J* of T such that $I(f_3(T)) = J(T)$.
- f shall be faithful if for every admissible interpretation *J* of every text T in L, there exists an admissible interpretation *I* of f_3(T) such that $I(f_3(T)) = J(T)$.

By Definition D.3.2.4, all the sort constraints of C are preserved by C′, but C′ may have additional constraints. Therefore, any interpretation that satisfies the stronger constraints in C′ would also satisfy C. The function f is faithful only if all the interpretations that satisfy C also satisfy C′. For signature-free logics, all translation functions are faithful.

D.3.3 Importing, translating, and combining texts

To support knowledge sharing and reuse, SCL allows multiple texts in different logics with different notations to be imported, translated, and combined in a single text of some SCL dialect. But not all logics can be combined. A set of SCL dialects is a *family* if each dialect has a faithful translation function to a member of the family called the *head* dialect. If all the texts belong to the same family, they can be combined by mapping them to a single text in the head dialect.

Since all translation functions for signature-free logics are faithful, any set of signature-free SCL dialects form a family with any dialect that is equivalent to SCL as the head. In a family of sorted SCL dialects, any signature-free dialect may be treated as a 1-sorted dialect.

D.3.3.1 Every signature-free SCL dialect L is also a 1-sorted SCL dialect with a sort structure (S,C):

- The only sort in S shall be the universe of any interpretation. For any text T in L, C shall consist of an unrestricted signature for each name *n* in T.
- If L and L′ are signature-free SCL dialects with a translation function *f* from L to L′, a faithful sorted translation function from L to L′ shall be the identity function in mapping the sort structure of L to L′, and it shall map any text *t* in L to a text *f*(*t*) in L′.

D.3.3.2 A set F of sorted SCL dialects shall be called a family if every dialect L in F has a faithful sorted translation function to some dialect H in F, which shall be called the *head* of F.

D.3.3.3 An SCL dialect text T, called a D-text, shall have a dialect name *d*, an optional text name *n*, an optional sort structure (S,C), a set *t* of sentences, a set *dt* of D-texts, and a set *im* of import statements. The sets *t*, *dt*, and *im* may be empty. Any element of *dt* or *im* is said to be nested in T.

A D-text is a structure that may contain D-texts and import statements nested arbitrarily deep. Before the semantics of a D-text is evaluated, it shall be transformed to an equivalent text in a single SCL dialect with no nested D-texts or import statements. Errors may occur that block some steps of that transformation. If the transformation of a D-text T is blocked, the semantics of T shall be undefined.

D.3.3.4 An *import statement* shall designate a D-text by some dialect-dependent naming convention. Any D-text *d* with a nested import statement *s* shall be equivalent to a D-text *d*′ that is identical to *d* with *s* replaced by the D-text designated by *s*.

D.3.3.5 If a D-text *d* has no import statements nested directly or indirectly in *d* and all the dialects of *d* and every D-text nested directly or indirectly in *d* are members of some family F, then *d* shall be equivalent to a D-text *e* derived by the following steps:

1. Let H be the head of F.
2. Translate the sentences of *d* and every D-text nested in *d* to sentences in the dialect H.
3. Let *e* be the D-text whose dialect is H and whose text *t* shall be the union of all the sets of sentences derived in step 2.
4. For every interpretation *I*, *I*(*d*) shall be defined as *I*(*e*).

Any combination of texts in signature-free dialects may be translated to the same head. But proving that an arbitrary set of sorted dialects belong to the same family may be undecidable. A simpler strategy is to start with a general sort structure for the head dialect. Then design a family of dialects that have faithful translation functions to the head.