E. Annex (informative): Conformance of UML class and object diagrams with DOL

This informative annex demonstrates conformance of UML class and object diagrams with DOL by defining an institution for both. We concentrate on the static aspects of class diagrams; that is, change of state is ignored. This means that all operations are query operations.

The institution of UML class and object diagrams is defined using a translation of UML class diagrams to Common Logic, following the fUML specification and [49].

From the fUML specification, section 10.3.1, we inherit the axioms for primitive types: Booleans, numbers, sequences and strings. These axiomatize (among others) predicates corresponding to primitive types, e.g. `buml:Boolean`, `form:Number`, `form:NaturalNumber`, `buml:Integer`, `form:Sequence`, `form:Character`, and `buml:String`.

We additionally need to axiomatize a number of predicates in Common Logic (note that enumerations are not axiomatized in fUML):

```
(distinct)        // the empty sequence is distinct
(distinct x)    // singleton sequences are distinct
(iff (distinct x y ...) // recursion for length > 1
     (and (not (= x y))       // the first two elements must be different
          (distinct x ...)    // and each of them distinct
          (distinct y ...) )) // to the rest

(iff (exhaustive c ...) (forall (x) (if (c x) (oneof x ...))))
     // does ... exhaust the extension of c?
(not (oneof x)) // is x among the remaining arguments?
(iff (oneof x y ...) (or (= x y) (oneof x ...)))
(iff (enumeration c ...) (and (exhaustive c ...) (distinct ...)))
   // c is an enumeration type with values ...

// fuml:sequence - membership of an element in a sequence
(forall (x s)
    (if (member x s)
        (form:Sequence s)))

(forall (x s)
    (iff (member x s)
         (exists (pt)
             (and (form:in-sequence s pt)
                  (form:in-position pt x)) )))
```

Using this infrastructure, we obtain an institution for UML class diagrams as follows:[73] Note(73)

Classifier hierarchies. A *classifier hierarchy* $(C, \leq_C)$ is given by a partial order where the set C contains the *classifier names*, which are closed w.r.t. the *built-in types* Boolean, UnlimitedNatural, Integer, Real, and String, i.e., $\{\mathsf{Boolean}, \mathsf{UnlimitedNatural}, \mathsf{Integer}, \mathsf{Real}, \mathsf{String}\} \subseteq C$;[74] and the partial ordering relation $\leq_C$ represents a *generalisation relation* on C, where we say that c_1 is a *sub-classifier* of c_2 if $c_1 \leq_C c_2$. Note(74)

A *classifier hierarchy map* $\gamma : (C, \leq_C) \to (D, \leq_D)$ is given by a monotone map from $(C, \leq_C)$ to $(D, \leq_D)$, i.e., $\gamma(c) \leq_D \gamma(c')$ if $c \leq_C c'$, such that $\gamma(c) = c$ for all built-in types $c \in \{\mathsf{Boolean}, \mathsf{UnlimitedNatural}, \mathsf{Integer}, \mathsf{Real}, \mathsf{String}\}$.

For each classifier $c \in C$ of a classifier hierarchy $(C, \leq_C)$ we use the *classifier annotations* OrderedSet, Set, Sequence, and Bag representing the meta-properties "ordered" and "unique" according to the following table:

	ordered	not ordered
unique	OrderedSet	Set
not unique	Sequence	Bag

We write $\tau[c]$ for an annotated classifier for $\tau \in \{\mathsf{OrderedSet}, \mathsf{Set}, \mathsf{Sequence}, \mathsf{Bag}\}$. The default is "not ordered" and "unique" (UML Superstructure Specification 2.4.1, p. 96).

Classifier nets (Signatures). A *classifier net* $\Sigma = ((C, \leq_C), K, P, M, A)$ comprises

- a *classifier hierarchy* $(C, \leq_C)$;
- a set K of *instance specifications declarations* of the form $k : c$ with k an *instance specification name* and $c \in C$;
- a set P of *property declarations* of the form $c.p : \tau[c']$ with $c, c' \in C$, τ a classifier annotation, and p a *property name*;
- a set M of *composition declarations* of the form $c\blacklozenge r : \tau[c']$ with $c, c' \in C$, τ a classifier annotation, and r a *composition role name*;
- and a set A of *association declarations* of the form $a\{r_1 : c_1, \ldots, r_n : c_n\}$ with $n \geq 2$, $c_1, \ldots, c_n \in C$, a an *association name*, and $r_1, \ldots, r_n$ *association role names*,

such that the following properties are satisfied:

- instance specification names are unique: if $k_1 : c_1$ and $k_2 : c_2$ are different instance specification declarations in K, then $k_1 \neq k_2$;
- property names are unique along the generalisation relation: if $c_1.p_1 : \tau_1[c_1']$ and $c_2.p_2 : \tau_2[c_2']$ are different property declararations in P and $c_1 \leq c_2$, then $p_1 \neq p_2$;
- composition role names are unique along the generalisation relation: if $c_1\blacklozenge r_1 : \tau_1[c_1']$ and $c_2\blacklozenge r_2 : \tau_2[c_2']$ are different composition declararations in M and $c_1 \leq c_2$, then $r_1 \neq r_2$;
- property and composition role names are unique along the generalisation relation: if $c_1.p_1 : \tau_1[c_1']$ is a property declaration in P and $c_2\blacklozenge r_2 : \tau_2[c_2']$ is a composition declaration in M and $c_1 \leq_C c_2$, then $p_1 \neq r_2$; and if $c_1\blacklozenge r_1 : \tau_1[c_1']$ is a composition declaration in M and $c_2.p_2 : \tau_2[c_2']$ is a property declaration in P and $c_1 \leq_C c_2$, then $r_1 \neq p_2$;
- association role names are unique: if $a\{r_1 : c_1, \ldots, r_n : c_n\}$ is an association declaration in A and $1 \leq i \neq j \leq n$, then $r_i \neq r_j$;

[73] NOTE: AK: Added the following up to the next ed-note.

[74] NOTE: what about enumeration types? Should we assume that some classifiers are marked as enumeration types and equipped with their set of constants?

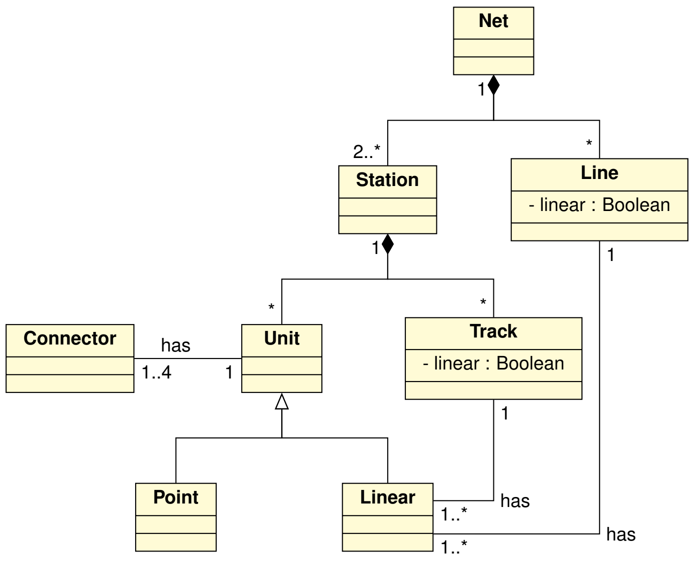

Figure E.1.: Sample UML class diagram.

– composition declarations are cycle-free: if $c_1◆r_1 : \tau_2[c_2], \dots, c_n◆r_n : \tau_{n+1}[c_{n+1}] \in M$, then $c_{n+1} \neq c_1$.

A *classifier net morphism* $\sigma = (\gamma, \kappa, \pi, \mu, \alpha) : \Sigma = ((C, \leq_C), K, P, M, A) \to \mathrm{T} = ((D, \leq_D), L, Q, N, B)$ is given by

– a classifier hierarchy map $\gamma : (C, \leq_C) \to (D, \leq_D)$;

– an instance specification map $\kappa : K \to L$ such that if $\kappa(k : c) = l : d \in L$, then $d = \gamma(c)$;

– a property declaration map $\pi : P \to Q$ such that if $\pi(c.p : \tau[c']) = d.q : \tau'[d'] \in Q$, then $d = \gamma(c)$, $d' = \gamma(d')$, and $\tau = \tau'$;

– a composition declaration map $\mu : M \to N$ such that if $\mu(c◆r : \tau[c']) = d◆s : \tau'[d'] \in M$, then $d = \gamma(c)$, $d' = \gamma(d')$, and $\tau = \tau'$;

– an association declaration map $\alpha : A \to B$ such that if $\alpha(a\{r_1 : \tau_1[c_1], \dots, r_n : \tau_n[c_n]\}) = b\{s_1 : \tau'_1[d_1], \dots, s_m : \tau'_m[d_m]\} \in B$, then there is a bijective map $\rho : \{r_1, \dots, r_n\} \to \{s_1, \dots, s_m\}$ with $d_j = \gamma(c_i)$ and $\tau_j = \tau'_i$ if $\rho(r_i) = s_j$.

Classifier nets as objects and classifier net morphisms as morphisms form the category of *classifier nets*, denoted by Cl.

Example For the class diagram in Fig. E.1 we have

Classifiers: Net, Station, Line, Connector, Unit, Track, Point, Linear

Generalisations: Point $\leq$ Unit, Linear $\leq$ Unit

Properties: Line.linear : Set[Boolean], Track.linear : Set[Boolean]

Compositions: Station◆unit : Set[Unit], Station◆track : Set[Track],
Net◆station : Set[Station], Net◆line : Set[Line]

$$\begin{aligned}\text{Associations: }&\mathsf{has}\{\mathsf{line} : \mathsf{Set}[\mathsf{Line}], \mathsf{linear} : \mathsf{Set}[\mathsf{Linear}]\},\\ &\mathsf{has}\{\mathsf{track} : \mathsf{Set}[\mathsf{Track}], \mathsf{linear} : \mathsf{Set}[\mathsf{Linear}]\},\\ &\mathsf{has}\{\mathsf{connector} : \mathsf{Set}[\mathsf{Connector}], \mathsf{unit} : \mathsf{Set}[\mathsf{Unit}]\}\end{aligned}$$

Multiplicity constraints. The set of *multiplicity formulae Frm* is given by the following grammar:

$$\begin{aligned}
\mathit{Frm} &::= \mathit{NumLiteral} \leq \mathit{FunExpr} \mid \mathit{FunExpr} \leq \mathit{NumLiteral} \mid \mathit{Composition}\ !\\
\mathit{FunExpr} &::= \#\ \mathit{Composition} \mid \#\ \mathit{Association}\ [\ \mathit{Role}\ (,\ \mathit{Role})^*\ ]\\
\mathit{Composition} &::= \mathit{Classifier}{\blacklozenge}\mathit{Role} : \mathit{Annot}\ [\ \mathit{Classifier}\ ]\\
\mathit{Association} &::= \mathit{Name}\{\mathit{Role} : \mathit{Annot}\ [\ \mathit{Classifier}\ ](,\ \mathit{Role} : \mathit{Annot}\ [\ \mathit{Classifier}\ ])^*\}\\
\mathit{Classifier} &::= \mathit{Name}\\
\mathit{Role} &::= \mathit{Name}\\
\mathit{Annot} &::= \mathsf{OrderedSet} \mid \mathsf{Set} \mid \mathsf{Sequence} \mid \mathsf{Bag}\\
\mathit{NumLiteral} &::= \mathsf{0} \mid \mathsf{1} \mid \cdots
\end{aligned}$$

where *Name* is a set of names.

The set of Σ-*multiplicity constraints* $\mathit{Mult}(\Sigma)$ for a classifier net Σ is given by the multiplicity formulae in *Frm* such that all mentioned elements of *Composition* and *Association* correspond to composition declarations and association declarations of Σ, respectively, and the role names mentioned in the last clause of *FunExpr* occur in the mentioned association. The *translation* of a formula $\varphi \in \mathit{Mult}(\Sigma)$ along a classifier net morphism σ, written as $\sigma(\varphi)$, is given by applying σ to compositions, associations, and role names.

Example For the class diagram in Fig. E.1 we have

$$\begin{aligned}
&2 \leq \#\mathsf{Net}{\blacklozenge}\mathsf{station} : \mathsf{Set}[\mathsf{Station}]\\
&\mathsf{Net}{\blacklozenge}\mathsf{station} : \mathsf{Set}[\mathsf{Station}]\,!\\
&\mathsf{Net}{\blacklozenge}\mathsf{line} : \mathsf{Set}[\mathsf{Line}]\,!\\
&\mathsf{Station}{\blacklozenge}\mathsf{unit} : \mathsf{Set}[\mathsf{Unit}]\,!\\
&\mathsf{Station}{\blacklozenge}\mathsf{track} : \mathsf{Set}[\mathsf{Track}]\,!\\
&1 \leq \#\mathsf{has}\{\mathsf{connector} : \mathsf{Set}[\mathsf{Connector}], \mathsf{unit} : \mathsf{Set}[\mathsf{Unit}]\}[\mathsf{unit}]\\
&\#\mathsf{has}\{\mathsf{connector} : \mathsf{Set}[\mathsf{Connector}], \mathsf{unit} : \mathsf{Set}[\mathsf{Unit}]\}[\mathsf{unit}] \leq 4\\
&\#\mathsf{has}\{\mathsf{connector} : \mathsf{Set}[\mathsf{Connector}], \mathsf{unit} : \mathsf{Set}[\mathsf{Unit}]\}[\mathsf{connector}] = 1\\
&1 \leq \#\mathsf{has}\{\mathsf{track} : \mathsf{Set}[\mathsf{Track}], \mathsf{linear} : \mathsf{Set}[\mathsf{Linear}]\}[\mathsf{track}]\\
&\#\mathsf{has}\{\mathsf{track} : \mathsf{Set}[\mathsf{Track}], \mathsf{linear} : \mathsf{Set}[\mathsf{Linear}]\}[\mathsf{linear}] = 1\\
&1 \leq \#\mathsf{has}\{\mathsf{line} : \mathsf{Set}[\mathsf{Line}], \mathsf{linear} : \mathsf{Set}[\mathsf{Linear}]\}[\mathsf{line}]\\
&\#\mathsf{has}\{\mathsf{line} : \mathsf{Set}[\mathsf{Line}], \mathsf{linear} : \mathsf{Set}[\mathsf{Linear}]\}[\mathsf{linear}] = 1
\end{aligned}$$

where we write "=" as an abbreviation for two inequations using "$\leq$".

For a classifier net $\Sigma = ((C, \leq_C), K, P, M, A)$, we define a Common Logic theory $CL(\Sigma)$ consisting of:

- for $c \in C$, a predicate[1] $\mathsf{CL}(c)$, such that[75] Note(75)

[1] Strictly speaking, this is just a name.
[75] Note: class predicates should be restricted to be unary

  - CL(Boolean) = `buml:Boolean`,
  - CL(String) = `buml:String`,
  - CL(Integer) = `buml:Integer`,
  - CL(UnlimitedNatural) = `form:NaturalNumber`,
  - CL(Real) = `buml:Real`,
  - CL(c) = c, if c is an enumeration type with values $k_1, \dots, k_n$. Additionally, the Common Logic theory is augmented by `(enumeration` c $k_1 \cdots k_n$`)`,[76] Note(76)
  - CL(List[c]) = `form:Sequence`[77], Note(77)
  - CL(Set[c]) =???[78], Note(78)
  - CL(OrderedSet[c]) =???[79], Note(79)
  - CL(Bag[c]) =???[80], Note(80)

- for each relation $c_1 \leq_C c_2$, an axiom `(forall (x) (if (C1 x) (C2 x)))`, where C1= CL(c_1), C2= CL(c_2),
- CL maps each instance specification declaration $k : c \in K$ to constant CL(k) and an axiom `(c k)`, where by abuse of notation, we identify c with CL(c), and k with (CL(k)) (this abuse of notation will also be used in the sequel);
- for the set of instance specifications $k_1 : c_1, \dots, k_n : c_n$ an axiom `(different` $k_1 \cdots k_n$`)` (the unique name assumption);
- CL maps each property declaration $c.p(x_1 : c_1, \dots, x_n : c_n) : \tau[c'] \in P$ to a predicate CL($c.p$) and axioms stating type-correctness and functionality:
  - `(forall (x` x_1 x_2 $\cdots$ x_n `y) (if (c.p x` x_1 x_2 $\cdots$ x_n `y) (`c x`)))`
  - `(forall (x` x_1 x_2 $\cdots$ x_n `y) (if (c.p x` x_1 x_2 $\cdots$ x_n `y) (`c_i x_i`)))` for each $i = 1 \dots n$,[2]
  - `(forall (x` x_1 x_2 $\cdots$ x_n `y) (if (c.p x` x_1 x_2 $\cdots$ x_n `y) (`$\tau[c']$ y`)))`
  - `(forall (x` x_1 x_2 $\cdots$ x_n `y m)`
    `(if (and (c.p x` x_1 x_2 $\cdots$ x_n `y) (member m y)) (c′ m)))`
  - `(forall (x` x_1 x_2 $\cdots$ x_n`)`
    `(exists (y) (c.p x` x_1 x_2 $\cdots$ x_n `y)))`
  - `(forall (x` x_1 x_2 $\cdots$ x_n `y z)`
    `(if (and (c.p x` x_1 x_2 $\cdots$ x_n `y) (c.p x` x_1 x_2 $\cdots$ x_n `z))`
    `(= y z)))`
- CL maps each composition declaration $c \blacklozenge r : \tau[c'] \in M$ to a predicate CL(r) and axioms
  `(forall (x) (if (c x) (exists (y) (and (r x y) (`τ`[c′] y)))))`
  `(forall (x y) (if (r x y) (and (c x) (`τ`[c′] y))))`
  `(forall (x y z) (if (and (r x y) (r x z)) (= y z)))`

[76] NOTE: Ed Seidewitz: enumerations are specializable, but this is type-unsafe, so maybe omit it
[77] NOTE: UML has typed sequences, but fUML appaerantly not. How do we solve this problem?
[78] NOTE: UML has sets, but fUML appaerantly not. Do we want to provide our own specification of sets in Common Logic?
[79] NOTE: UML has ordered sets, but fUML appaerantly not. Do we want to provide our own specification of ordered sets in Common Logic?
[80] NOTE: UML has bags, but fUML appaerantly not. Do we want to provide our own specification of bags in Common Logic?
[2] Note that the $\cdots$ here is meta notation, not a sequence marker!

- for any pair of composition declarations $c_1 \blacklozenge r_1 : c'_1$ and $c_2 \blacklozenge r_2 : c'_2$, an axiom stating "each instance has at most one owner":
  `(forall (`x_1 x_2 y_1 y_2`) (if (and (`r_1 x_1 y_1`) (`r_2 x_2 y_2`))`
  `(exists (z) (and (member z `x_1`) (member z `x_2`)))))`
- CL maps each association declaration $a(r_1 : c_1, \ldots, r_n : c_n) \in A$ to a predicate $\mathsf{CL}(a)$ and an axiom
  `(forall (`x_1 x_2 $\cdots$ x_n`) (if (a `x_1 x_2 $\cdots$ x_n`) (and (`c_1 x_1`) `$\cdots$` (`c_n x_n`)))))`[3]

It is straightforward to extend CL from signatures to signature morphisms.

Models. A Σ-model of the UML class diagram institution is just a $\mathsf{CL}(\Sigma)$-model in Common Logic. That is, the UML class diagram institution inherits models from Common Logic. Moreover, model reducts are inherited as well, using the action of CL on signature morphisms.

Multiplicity formulae — Sentences. [81] [82] [83] For the sentences of the UML class diagram institution we use *multiplicity formulae* defined by the following grammar: Note(81) Note(82) Note(83)

$$\begin{aligned}
\mathit{Frm} &::= \mathit{NumLit} \leq \mathit{FunExpr} \mid \mathit{FunExpr} \leq \mathit{NumLit} \mid \mathit{Composition}\ !\\
\mathit{FunExpr} &::= \#\ \mathit{Composition} \mid \#\ \mathit{Association}\ [\ \mathit{Role}(,\ \mathit{Role})^*\ ]\\
\mathit{Composition} &::= \mathit{Class}\blacklozenge\mathit{Role} : \mathit{Class}\\
\mathit{Association} &::= \mathit{Name}\{\mathit{Role} : \mathit{Class}(,\ \mathit{Role} : \mathit{Class})^*\}\\
\mathit{Class} &::= \mathit{Name}\\
\mathit{Role} &::= \mathit{Name}\\
\mathit{NumLit} &::= \mathsf{0} \mid \mathsf{1} \mid \cdots
\end{aligned}$$

where *Name* is a set of strings. The $\leq$-formulae express constraints on the cardinalities of composition and association declarations, i.e., how many instances are allowed to be in a certain relation with others. The #-expressions return the number of links in an association when some roles are fixed. The !-formulae for compositions express that the owning end must not be empty. The set of sentences or Σ-*multiplicity constraints* $\mathit{Mult}(\Sigma)$ for a class net Σ is given by the multiplicity formulae in *Frm* such that all mentioned elements of *Composition* and *Association* correspond to composition declarations and association declarations of Σ respectively, and the *Role* names mentioned in the last clause of *FunExpr* occur in the mentioned association.

EXAMPLE Some of the cardinality constraints of the running example in Fig. E.1 can be expressed with multiplicity formulae as follows:

$\mathsf{Station}\blacklozenge\mathsf{has} : \mathsf{Track}\,!$
— each Track is owned by a Station via has,

$2 \leq \#\mathsf{has}(\mathsf{connector} : \mathsf{Connector}, \mathsf{unit} : \mathsf{Unit})[\mathsf{unit}]$
— association has links each Unit to at least two Connectors,

$\#\mathsf{state}(\mathsf{unitState} : \mathsf{UnitState}, \mathsf{unit} : \mathsf{Unit})[\mathsf{unit}] = 1$
— association state links each Unit to exactly one UnitState.

[3] Ignoring the annotations τ_i in the interpretation of an association is intentional, see the semantics of associations in the UML Superstructure Specification 2.4.1, p. 37.

[81] NOTE: Ed Seidewitz: Multiplicities are different for associations and for properties, this is a bit tricky. Associations are predicates of single values, whereas properties can be over sets etc. Aexander: we need to correct the representation for compositions used in associations.

[82] NOTE: missing: relations between properties, e.g. subsets, unions

[83] NOTE: derived associations: one association is the composition of two others

Note we have used formulae with $=$ instead of two formulae with $\leq$ where the left hand side and the right hand side are switched.□

The *translation* of a formula $\varphi \in \mathit{Mult}(\Sigma)$ along a class net morphism $\sigma = (\gamma, \kappa, \pi, \mu, \alpha) : \Sigma \to \mathrm{T}$, written as $\sigma(\varphi)$, is given by applying σ to compositions, associations, and role names.

Satisfaction relation. The satisfaction relation is inherited from Common Logic, using a translation $\mathsf{CL}(\_)$ of multiplicity formulas to Common Logic. That is, given a UML class and object diagram Σ, a multiplicity formula φ and a Σ-model M (the latter amounts to a $\mathsf{CL}(\Sigma)$-model M in Common Logic), we define

$$M \models_\Sigma \varphi \text{ iff } M \models_{\mathsf{CL}(\Sigma)} \mathsf{CL}(\varphi)$$

The translation of multiplicity formulas to Common Logic is as follows:

- $\mathsf{CL}(\ell \leq \#c\blacklozenge r : c') =$
  (forall (x y n)
      (if (and (r x y) (form:sequence-length y n)) (leq $[\![\ell]\!]$ n))
- $\mathsf{CL}(\ell \leq \#c\blacklozenge r : c') =$
  (forall (x y n)
      (if (and (r x y) (form:sequence-length y n)) (geq $[\![\ell]\!]$ n))
- $\mathsf{CL}(c\blacklozenge r : c'!) =$ (forall (y) (if (c′ y) (exists (x) (and (c x) (r x y)))))
- $\mathsf{CL}(\ell \leq \#a(r_1 : c_1, \ldots, r_n : c_n)[r_{i_1}, \ldots, r_{i_m}] =$
  (forall (x_{i_1} $\cdots$ x_{i_m})
      (if (and (c_{i_1} x_{i_1}) $\cdots$ (c_{i_m} x_{i_m}))
          ((min-card-tuple a x_{i_1} $\cdots$ x_{i_m}) sel_{i_1} $\cdots$ sel_{i_n})))
- $\mathsf{CL}(\#a(r_1 : c_1, \ldots, r_n : c_n)[r_{i_1}, \ldots, r_{i_m}] \leq \ell =$
  (forall (x_{i_1} $\cdots$ x_{i_m})
      (if (and (c_{i_1} x_{i_1}) $\cdots$ (c_{i_m} x_{i_m}))
          ((max-card-tuple a x_{i_1} $\cdots$ x_{i_m}) sel_{i_1} $\cdots$ sel_{i_n})))

where $[\![-]\!] : \mathit{NumLit} \to \mathbb{Z}$ maps a numerical literal to an integer.[84] Note(84)

[84] Note: the last two items need adaption