Towards a Type System for Semantic Streams[⋆]

(Extended abstract on work in progress)

Michael Mendler and Stephan Scheele

Informatics Theory Group
University of Bamberg
{michael.mendler,stephan.scheele}@uni-bamberg.de

Abstract. This paper proposes an approach to use constructive description logics as a typing system for streaming data in the domain of auditing. We introduce the constructive description logic $c\mathcal{ALC}$ and show how it can serve as a semantic type system and knowledge representation formalism for data streams and give a direct interpretation of proofs as computations following the Curry-Howard-Isomorphism.

1 Introduction

Description Logics(DLs) [1] are a family of knowledge representation languages that can be used to represent the terminological knowledge of a specific application domain in a structured and formally well-understood way. DLs have become important in the Semantic Web as a formal base for the W3C-endorsed Web Ontology Language (OWL).

In some domains like auditing, complications with the standard arise and one has to deal with partial knowledge that is evolving and incomplete. Auditors need to understand the flow of information (streaming data) and their related control activities to be able to ensure the validity and reliability of business transactions. Classical DLs, however, are insufficiently expressive to deal with streaming information (auditing knowledge) which is highly evolving and incomplete. The entities building up the domain may not be determined and concrete but are abstractions of real individuals whose properties are evolving and defined only up to construction. Classical DLs assume that a concept is static and at the outset either includes a given entity or not. However, in auditing either option may be inconsistent, if the entity or the concept is not fully defined until a later stage of the auditing process where lower levels of detail become available. Because the semantical meaning is context-dependent and possibly involves many levels of explication, DLs for auditing must support a constructive notion of abstraction which permits that concepts and entities evolve. Audit statements about the validity of the streaming information, e.g., accounting data, absence of fraud or conformance to financial process standards must constructively take account of many dimensions of abstraction and refinement [4].

Towards this goal we have developed a new constructive description logic [4], called $c\mathcal{ALC}$, which is meant to form the core of a family of semantic type systems for the static analysis of audit processes but also to support the automated generation of executable audit processes themselves. $c\mathcal{ALC}$ is based on DLs but its semantics are refined to a constructive notion of truth which is capturing the uncertainty aspects for dealing with data streams. The idea is that $c\mathcal{ALC}$ would be extended to represent semantic ontologies for streaming data in the domain of auditing and operationalised in terms of a type-theoretic specification language for a data-flow execution model of audit streams. In the following we recall the definition of $c\mathcal{ALC}$ [4] and we suggest its application as a type system showing a direct interpretation of proofs as computations following the Curry-Howard-Isomorphism. This is inspired by the work of Bozatto et al. [2].

2 Syntax and Semantics of Constructive Description Logic $c\mathcal{ALC}$

Concept descriptions in $c\mathcal{ALC}$ are based on sets of *role names* N_R and atomic *concept names* N_C and formed as follows, where $A \in N_C$ and $R \in N_R$:

$$C, D \to A \mid \bot \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid C \sqsubseteq D \mid \exists R.C \mid \forall R.C.$$

[⋆] Research project funded by the German Research Foundation (DFG) under ME 1427/4-1.

In contrast to standard $\mathcal{ALC}$ [1] this syntax includes subsumption $\sqsubseteq$ as a concept–forming operator. Being a first class operator, subsumption can be nested arbitrarily as in $((D \sqsubseteq C) \sqsubseteq B) \sqsubseteq A$. Negation $\neg$ can be represented as $\neg C = C \sqsubseteq \bot$. Concepts describe a "class" of objects or individuals, a role represents a binary relation on these classes.

Example 1. To explain what we have in mind let Δ be a class of database tables over attributes N_R. These tables are collections of records $r = [R_1 = v_1, R_2 = v_2, \ldots, R_n = v_n]$ where $R_i \in N_R$ and each v_i is either a basic value $v_i \in \mathbb{D}$ or a (reference, key) to another table. The domain $\mathbb{D}$ of basic data might comprise, e.g., Booleans $\mathbb{B} = \{\mathtt{T}, \mathtt{F}\}$, naturals $\mathbb{N}$ and strings $\mathbb{S}$, i.e., $\mathbb{D} = \mathbb{N} \uplus \mathbb{B} \uplus \mathbb{S}$. The record r has *type* $type(r) = \{R_1{:}\mathbb{D}, R_2{:}\mathbb{D}, \ldots, R_n{:}\mathbb{D}\}$.

E.g. assume a record $r_w = [hasName = Petit\ Chablis, hasOrigin = Alsace, isVintage = 1999, awarded = \mathtt{F}]$ describing a single wine which is of the *type* $type(r_w) = \{\ hasName{:}\mathbb{S}, hasOrigin{:}\mathbb{S}, isVintage{:}\mathbb{N}, awarded{:}\mathbb{B}\ \}$. This record is an instance of the concept description $WINE_r = \exists hasName.\mathbb{S} \sqcap \exists hasOrigin.\mathbb{S} \sqcap \exists isVintage.\mathbb{N} \sqcap \forall awarded.\mathbb{B}$. Therefore, this concept description can be thought of as a weak type specification of r_w. Note, that there is the possibility to represent null-values by use of $\forall$ instead of $\exists$. □

Constructive interpretations $\mathcal{I}$ of concept descriptions extend the classical models for $\mathcal{ALC}$ by a pre–ordering $\preceq^{\mathcal{I}}$ for expressing refinement between individuals and by a notion of fallible entities $\bot^{\mathcal{I}}$ for interpreting contradiction. As explained in [4], entities of the domain $\Delta^{\mathcal{I}}$ are partial descriptions representing incomplete information about individuals. The relationship $a \preceq b$ says that a *"is an abstraction of"* b, i.e., that it has *"contains no more information"* than b. Fallible elements $b \in \bot^{\mathcal{I}}$ may be thought of as over–constrained tokens of information, self-contradictory objects of evidence or undefined computations. E.g., they may be used to model the situation in which computing a role–filler for an abstract individual a fails, i.e., $\forall b.\, R(a, b) \Rightarrow b \in \bot^{\mathcal{I}}$, yet when a is refined to a' then a non-fallible role-filler $b' \in \Delta^{\mathcal{I}}$ exists with $R(a', b')$. In [4] several examples are given illustrating this.

The refinement relation $\preceq$ accommodates a range of different applications [3], one important interpretation is the stream interpretation. In this way a database table $t \in \Delta^{\mathcal{I}}$ can be presented as a stream of records as follows. Let t be a *stream* of records. In this case $t = t_1 \cdot t_2 \cdot \ldots$ is a finite or infinite stream where $\preceq$ is the suffix ordering, which is the least relation closed under the rule

$$\frac{v \in \mathbb{D}}{v \cdot s \preceq s}$$

where $v \cdot s$ is the stream $s \in (\mathbb{D}^* \cup \mathbb{D}^\infty)$ prefixed by value $v \in \mathbb{D}$. For instance,

$$1 \cdot (2, \mathtt{T}) \cdot \mathtt{T} \cdot \mathtt{F} \ \preceq^{\mathcal{I}}\ (2, \mathtt{T}) \cdot \mathtt{T} \cdot \mathtt{F} \ \preceq^{\mathcal{I}}\ \mathtt{T} \cdot \mathtt{F} \ \preceq^{\mathcal{I}}\ \mathtt{F} \ \preceq^{\mathcal{I}}\ \epsilon,$$

is a stream of naturals, booleans and their pairings, where ϵ denotes the empty stream. Under this interpretation, concepts $C^{\mathcal{I}}$, which must be closed under $\preceq$ express *future projected behaviour* of streams. The empty stream is a fallible entity and has no future behaviour, it represents a computational deadlock, i.e., $\bot^{\mathcal{I}} = \{\epsilon\}$. Fallible elements $\bot^{\mathcal{I}}$ correspond to divergent or dead-locked reactions which do not produce any value. Viewed as stream computations these produce ϵ which is universally polymorphic, i.e., naturally contained in any type. This corresponds to the fact that fallible entities in $c\mathcal{ALC}$ are information-wise maximal elements and therefore included in every concept, i.e., $\bot^{\mathcal{I}} \subseteq C^{\mathcal{I}}$ for all C.

Definition 1. A *constructive interpretation* of $c\mathcal{ALC}$ is a structure $\mathcal{I} = (\Delta^{\mathcal{I}}, \preceq^{\mathcal{I}}, \bot^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consisting of a non-empty set $\Delta^{\mathcal{I}}$ of *entities*, the universe of discourse; a *refinement* pre-ordering $\preceq^{\mathcal{I}}$ on $\Delta^{\mathcal{I}}$, i.e., a reflexive and transitive relation; a subset $\bot^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ of *fallible* entities, and an interpretation function $\cdot^{\mathcal{I}}$ mapping each role name $R \in N_R$ to a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and each atomic concept $A \in N_C$ to a set $\bot^{\mathcal{I}} \subseteq A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, which is closed under refinement. □

The crucial point about the constructive semantics is that each entity implicitly subsumes all its refinements and truth is inherited. Specifically, if entity x satisfies a concept description C and $x \preceq^{\mathcal{I}} y$ then this implies that y satisfies C as well, for all concepts C. Robustness under refinement is a natural prerequisite of type systems. Suppose p is a program of type τ, where

τ expresses some static properties of the computations performed by p. The statement that "p guarantees τ" is usually written $p{:}\tau$. Now, if $p \xrightarrow{v} p'$ is a computation step in which p produces a response value v and then transforms to the continuation program p', then soundness of typing requires that p', too, satisfies the type specification τ. This is known as the *subject reduction* property. Thus, if we wish to view $c\mathcal{ALC}$ concepts C as type specifications then it is natural to consider computation steps $p \xrightarrow{v} p'$ as a form of refinement $\preceq^{\mathcal{I}}$. The typing statement $p{:}C$ then corresponds to the condition $p \in C^{\mathcal{I}}$ and subject reduction is for free. Deadlocked programs are represented by fallible entities and do not produce any value, i.e., $p{:}\bot$ iff $p \not\longrightarrow$.

In other words, if streams are considered as abstract entities then $c\mathcal{ALC}$ concepts can act as a typing system to specify the static semantics of streams of business data such as as linearised database tables or time–series of financial market transactions. Note that in this model database tables are first class values. In standard relational databases tables do not usually contain references to tables but merely references (keys) to *entries* in other tables. This is a first-order model of data. Here we will permit that attribute values inside an abstract entity or concept may be described by its own database table. In this way we obtain a second-order semantics in which the data to be manipulated are tables (i.e., collections of records) rather than records of atomic data. This is in line with the idea of an hierarchical data model in which objects are composite and arising by abstraction of lower level parameters.

3 A Constructive Term Assignment System

With each concept C we associate a set of *realisers* or *information terms* $\mathrm{IT}(C)$. Information terms [2] can be thought of as constructive explanation of logical connectives, i.e. they are mathematical objects explaining the truth of a formula by providing a witness in the form of a construction or program. Latter can be obtained by use of a calculus as has been demonstrated in [3].

Example 2. To illustrate information terms, let us reconsider the Food-Wine knowledge-base by Brachman et.al. (1991) as reported by [2]. First a knowledge-base consists of the terminological knowledge and is called in short TBox. In our case this is given by the atomic concepts $FOOD, WINE, COLOR$, roles $isColorOf, goesWith$ and the following axioms: $\Theta = \{Ax_1, Ax_2\}$ where

$$Ax_1 =_{df} \text{FOOD} \sqsubseteq \exists goesWith.\text{COLOR}$$
$$Ax_2 =_{df} \text{COLOR} \sqsubseteq \exists isColorOf.\text{WINE}.$$

Intuitionally AX_1 specifies that for all individuals of the concept FOOD exists a role–filler over relation *goesWith* that belongs to the concept COLOR, AX_2 can be explained similarly. Instances of TBox concept descriptions are specified as assertional knowledge and form the so-called ABox: For the Food-Wine ontology we give the following ABox, which satisfies the axioms AX_1 and AX_2:

$fish : FOOD$	$red : COLOR$	$(red, barolo) : isColorOf$
$meat : FOOD$	$white : COLOR$	$(white, chardonnay) : isColorOf$
$chardonnay : WINE$		$(fish, white) : goesWith$
$barolo : WINE$		$(meat, red) : goesWith.$

The Curry–Howard–Isomorphism can be customised to realise any Hilbert-proof of AX_1 and AX_2 as a program construction. For instance, AX_1 can be read as a function transforming FOOD-entities u into COLOR-entities x such that $goesWith(u, x)$, similarly AX_2 is a function from COLORs c to WINEs w so that $isColorOf(c, w)$. □

These realisers are then taken as extra ABox parameters so that instead of $\mathcal{I} \models x{:}C$ we declare what it means that $\mathcal{I} \models x{:}\langle\alpha\rangle C$, relative to a given interpretation $\mathcal{I}$, for a particular realiser $\alpha \in \mathrm{IT}(C)$. This so-called *realisability predicate* gives additional constructive semantics to our concepts in the sense that $\mathcal{I} \models x{:}\langle\alpha\rangle C$ implies $\mathcal{I} \models x{:}C$.

The sets $\mathrm{IT}(C)$ and refined concepts $\langle\alpha\rangle C$ are defined by induction on C. For our example we need the following information terms:

- $\mathrm{IT}(A) =_{df} 1 = \{0\}$ for atomic concepts;
- $\mathrm{IT}(\bot) =_{df} 1 = \{0\}$;
- $\mathrm{IT}(C \sqcap D) =_{df} \mathrm{IT}(C) \times \mathrm{IT}(D)$;
- $\mathrm{IT}(C \sqcup D) =_{df} \mathrm{IT}(C) + \mathrm{IT}(D)$;
- $\mathrm{IT}(C \sqsubseteq D) =_{df} \mathrm{IT}(C) \to \mathrm{IT}(D)$;
- $\mathrm{IT}(\exists R.C) =_{df} (\Delta^{\mathcal{I}} \to \Delta^{\mathcal{I}}) \times (\Delta^{\mathcal{I}} \to \mathrm{IT}(C))$;
- $\mathrm{IT}(\forall R.C) =_{df} \Delta^{\mathcal{I}} \to \mathrm{IT}(C)$.

Definition 2 (Realisability). Realisability is such that

- $\mathcal{I} \models x{:}\langle 0\rangle A$ iff $x \in A^{\mathcal{I}}$ for $A = \bot$ or atomic;
- $\mathcal{I} \models x{:}\langle \alpha, \beta\rangle (C \sqcap D)$ iff $\mathcal{I} \models x{:}\langle \alpha\rangle C$ and $\mathcal{I} \models x{:}\langle \beta\rangle C$;
- $\mathcal{I} \models x{:}\langle \alpha\rangle (C \sqcup D)$ iff $\alpha = \iota_0\beta$ and $\mathcal{I} \models x{:}\langle \beta\rangle C$ or $\alpha = \iota_1\beta$ and $\mathcal{I} \models x{:}\langle \beta\rangle C$;
- $\mathcal{I} \models x{:}\langle f\rangle (C \sqsubseteq D)$ iff $\forall y \succeq x. \forall \alpha \in \mathrm{IT}(C).\ \mathcal{I} \models y{:}\langle \alpha\rangle C \Rightarrow \mathcal{I} \models y{:}\langle f\alpha\rangle D$;
- $\mathcal{I} \models x{:}\langle a, \alpha\rangle (\exists R.C)$ iff $\forall y \succeq x.\ (y, a\, y) \in R^{\mathcal{I}}\ \ \&\ \ \mathcal{I} \models a\, y{:}\langle \alpha\, (a\, y)\rangle C$;
- $\mathcal{I} \models x{:}\langle \alpha\rangle (\forall R.C)$ iff $\forall y \succeq x. \forall a \in \Delta^{\mathcal{I}}.\ (y, a) \in R^{\mathcal{I}} \Rightarrow \mathcal{I} \models a{:}\langle \alpha\, a\rangle C$.

□

Under the Curry-Howard Isomorphism (propositions-as-types) [5, 6] the Cartesian product $C \times D$ is the constructive interpretation of conjunction $C \sqcap D$, function spaces $C \to D$ are the constructive reading of subsumptions $C \sqsubseteq D$ and Disjunction $C \sqcup D$ can be interpreted by disjoint union. The information term for $\exists R.C$ can be written equivalently as $\Delta^{\mathcal{I}} \to (\Delta^{\mathcal{I}} \times \mathrm{IT}(C))$ that can be interpreted as a function from $\Delta^{\mathcal{I}}$ to a pair, which consists of a witness for the role-filler and the information term for C. $\forall R.C$ is interpreted as function assigning every $x \in \Delta^{\mathcal{I}}$ an information term for C.

Example 3. Information terms for the axioms AX_1, AX_2 of Ex. 2 arise either from proof terms [3] or they are created from a particular ABox. Based on the ABox of Ex. 2 the information terms ax_1, ax_2 can be chosen such that

$$ax_1 =_{df} \lambda u.\lambda x.\mathtt{case}\, u\, \mathtt{of}\, [\mathtt{meat} \to (\mathtt{red}, 0) \mid \mathtt{fish} \to (\mathtt{white}, 0)]$$
$$ax_2 =_{df} \lambda u.\lambda x.\mathtt{case}\, u\, \mathtt{of}\, [\mathtt{red} \to (\mathtt{barolo}, 0) \mid \mathtt{white} \to (\mathtt{chardonnay}, 0)]$$

where $ax_1 \in \Delta^{\mathcal{I}} \to \mathrm{IT}(\textsc{food} \sqsubseteq \exists \mathit{goesWith}.\textsc{color})$ and $ax_2 \in \Delta^{\mathcal{I}} \to \mathrm{IT}(\textsc{color} \sqsubseteq \exists \mathit{isColorOf}.\textsc{wine})$. These express the constructive content of Ax_1, Ax_2 in $\mathcal{I}$. ax_1 can be read as a function which returns a pair of COLOR and 0 dependent on case analysis on input u to decide between *meat* and *fish*, 0 is taken here as second component since it is the information term for the atomic concept COLOR. □

Proposition 1 (Subject Reduction). *Let* $\mathcal{I} \models x{:}\langle \alpha\rangle C$ *and* $x \preceq y$. *Then,* $\mathcal{I} \models y{:}\langle \alpha\rangle C$.

Proof. By induction on concepts C. For atomic concepts $\langle 0\rangle A$ the statement follows by invariance under refinement. The cases $\langle \alpha, \beta\rangle (C \sqcap D)$, $\langle \alpha\rangle (C \sqcup D)$ are by induction. For $\langle f\rangle (C \sqsubseteq D)$, $\langle a, \alpha\rangle (\exists R.C)$ and $\langle \alpha\rangle (\forall R.C)$ we exploit transitivity of refinement. □

Example 4. An information term for $A \sqcup B$ is an element $\alpha \in \mathrm{IT}(A \sqcup B) = 1 + 1 = 2$ which is represented canonically as the set $2 = \{0, 1\}$. If $\alpha = 0$ then $\mathcal{I} \models x{:}\langle \alpha\rangle (A \sqcup B)$ iff $x \in A^{\mathcal{I}}$ and if $\alpha = 1$ then $\mathcal{I} \models x{:}\langle \alpha\rangle (A \sqcup B)$ iff $x \in B^{\mathcal{I}}$. Thus, the realiser gives explicit information about which side of the disjunction holds. If we abstract from α we get standard validity: $\exists \alpha. \mathcal{I} \models x{:}\langle \alpha\rangle (A \sqcup B)$ is the same as $x \in (A \sqcup B)$. □

Example 5. Assume that $\Delta^{\mathcal{I}}$ contains the particular stable stream $\mathsf{data} = \mathsf{data}_0\mathsf{data}_1\mathsf{data}_2 \ldots$ where each data_i is a non-deterministic record with every possible data value as a *val*-filler, i.e., $(\mathsf{data}_i, v) \in \mathit{val}^{\mathcal{I}}$ for all $i \geq 0$ and $v \in \mathbb{D} = \mathbb{N} \uplus \mathbb{B}$. Such a stream data may be viewed as the universal abstraction of all entities that have a *val*-filler in $\mathbb{D}$. Think of data as a concept name for the time-invariant set $\mathbb{D}$ and *val* as a role name for the set-theoretic element relationship $\in$.

Under interpretation $\mathcal{I}$ let NAT and BOOL be the atomic concepts of natural numbers and booleans, i.e., such that $\textsc{nat}^{\mathcal{I}} = \mathbb{N}$ and $\textsc{bool}^{\mathcal{I}} = \mathbb{B}$. Then $\mathsf{data} \in \exists \mathit{val}.\textsc{nat}$ and $\mathsf{data} \in \exists \mathit{val}.\textsc{bool}$. A realiser of $\exists \mathit{val}.\textsc{nat}$ on data is just a stream of naturals (as *val*-fillers). Similarly, a realiser of $\exists \mathit{val}.\textsc{bool}$ on data is an arbitrary stream of Booleans. Every stream of records with *val*-fillers in

$\mathbb{D}$ can be seen as a realiser of $\exists val.(\textsc{bool} \sqcup \textsc{nat})$ on data while a realiser of $\exists val.\textsc{bool} \sqcup \exists val.\textsc{nat}$ must select either a stream of Booleans or a stream of naturals.

Now, what is an information term realising the concept description $\exists val.\textsc{nat} \sqsubseteq \exists val.\textsc{bool}$ for entity data, i.e., f such that

$$\mathcal{I} \models \mathsf{data}{:}\langle f\rangle(\exists val.\textsc{nat} \sqsubseteq \exists val.\textsc{bool})?$$

According to the definition f is a function from realisers α with $\mathcal{I} \models y{:}\langle\alpha\rangle(\exists val.\textsc{nat})$ to realisers $f\,\alpha$ such that $\mathcal{I} \models y{:}\langle f\,\alpha\rangle(\exists val.\textsc{bool})$ for all $y \succeq \mathsf{data}$. Since data is a constant stream every realiser for a refinement of data is a realiser for data and vice versa, i.e., $\mathcal{I} \models \mathsf{data}{:}\langle\alpha\rangle C$ iff $\mathcal{I} \models y{:}\langle\alpha\rangle C$ for $y \succeq \mathsf{data}$. Thus, f maps realisers α of $\exists val.\textsc{nat}$ (on data) to realisers $f\,\alpha$ of $\exists val.\textsc{bool}$. Given what has been said above, this means that f is a function from streams of naturals to streams of Booleans. □

One can then show that every Hilbert proof $\vdash_H C$ generates, for any interpretation $\mathcal{I}$, a function $f{:}\Delta^{\mathcal{I}} \to \mathrm{IT}(C)$ such that $\forall u \in \Delta^{\mathcal{I}}.\mathcal{I} \models u{:}\langle fu\rangle C$. Specifically, each of the following Hilbert axioms is realised by a λ-term, for instance:

$$\begin{aligned}
&\mathrm{IPL1}: C \sqsubseteq (D \sqsubseteq C) \rightsquigarrow \lambda u.\lambda x.\lambda y.x\\
&\mathrm{IPL2}: ((C \sqsubseteq (D \sqsubseteq E)) \sqsubseteq (C \sqsubseteq D) \sqsubseteq (C \sqsubseteq E)) \rightsquigarrow \lambda u.\lambda x.\lambda y.\lambda z.\ (xz)(yz)\\
&\mathrm{IPL3}: C \sqsubseteq (D \sqsubseteq (C \sqcap D)) \rightsquigarrow \lambda u.\lambda x.\lambda y.\ (x,y)
\end{aligned}$$

Axiom $\exists K$ is the function $\exists K =_{df} \lambda u.\lambda x.\lambda y.(\pi_1 y, x(\pi_1 y)(\pi_2 y))$. The rule of MP and Nec are refined to

If $\langle\alpha\rangle C$ and $\langle\beta\rangle(C \sqsubseteq D)$ then $\langle\lambda u.(\beta\,u)(\alpha\,u)\rangle D$
If $\langle\alpha\rangle C$ then $\langle\lambda u.\lambda x.\,\alpha\,x\rangle(\forall R.C)$.

Example 6. In this way, using the above mentioned axioms Ax_1 and Ax_2, the derivation of the concept description $\textsc{food} \sqsubseteq \exists goesWith.(COLOR \sqcap \exists isColorOf.\textsc{wine})$, up to reductions in the λ-calculus, corresponds to

$$prf = \lambda u.\lambda x.(\pi_1(ax_1\,x), (\pi_2(ax_1\,x), (\pi_1(ax_2(\pi_2(ax_1\,x))), \pi_2(ax_2(\pi_2(ax_1\,x))))))$$

which is an information term such that $\forall u.\mathcal{I} \models u{:}\langle prf\;u\rangle(\textsc{food} \sqsubseteq \exists goesWith.(\textsc{color} \sqcap \exists isColorOf.\textsc{wine}))$ assuming that $\forall u.\mathcal{I} \models u{:}\langle ax_1\,u\rangle Ax_1$ and $\forall u.\mathcal{I} \models u{:}\langle ax_2\,u\rangle Ax_2$. Such realisers ax_1, ax_2 can be obtained from a concrete ABox as has been shown in Ex. 3. □

4 Outlook

Auditors need reliable tools to draw the right consequences, thus they have strong requirements on correctness of software (certified code) so that formal, mathematical methods are needed. Extensions of $c\mathcal{ALC}$ can serve as ontological specification languages to characterise the semantics of financial information flows as strongly typed data streams. Logic-based reasoning algorithms (type-checking, model-checking) will then be available to raise the trustworthiness of next-generation auditing tools. We aim to develop a component–based data flow programming environment in which $c\mathcal{ALC}$ types can be used as type specification of stream processing functions and for the extraction of auditing processes from proof terms by use of a calculus.

References

1. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider. *The description logic handbook: theory, implementation, and applications.* Cambridge University Press, 2003.
2. L. Botazzo, M. Ferrari, C. Fiorentini, and G. Fiorino. A constructive semantics for ALC. In *Int'l Workshop on Description Logics (DL 2007)*, pages 219–226, 2007.
3. M. Mendler and S. Scheele. Towards constructive description logics for abstraction and refinement. Technical Report 77(2008), University of Bamberg, September 2008.
4. M. Mendler and S. Scheele. Towards constructive dl for abstraction and refinement. In Franz Baader, Carsten Lutz, and Boris Motik, editors, *21st International Workshop on Description Logics (DL2008), Dresden, Germany, May 13-16, 2008*, volume 353 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.
5. A. S. Troelstra and D. van Dalen. *Constructivism in Mathematics*, volume II. North-Holland, 1988.
6. D. van Dalen. Intuitionistic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume III, chapter 4, pages 225–339. Reidel, 1986.