Towards a Conceptual Structure based on Type theory.

Richard Dapoigny and Patrick Barlatier

Université de Savoie, Polytech'Savoie Laboratoire d'Informatique, Sytèmes,
Traitement de l'Information et de la Connaissance Po.Box 80439,
F-74944 ANNECY-Le-Vieux Cedex, France
Phone: +33 450 096529 Fax: +33 450 096559 richard.dapoigny@univ-savoie.fr

Abstract. Since a conceptual structure is a typed system it is worthwhile to investigate how a type theory can serve as a basis to reason about concepts and relations. In this article, we look at this issue from a proof-theoretical perspective using the constructive (or intuitionistic) logic and the Curry-Howard correspondence. The resulting constructive type theory introduces Dependent Record Types (DRT) which offers a conceptual structure with a simple and natural representation. The crucial aspect of the proposed typed system is its decidability while maintaining a high level of expressivity.

1 Introduction

In most domains including the semantic Web, ontology and rules are the core components for formal knowledge representation. As a result, there is a need for an expressive formalism (e.g., Description Logics) able to reason about knowledge extracted from ontologies. Recently, Description Logics have attempted to represent action formalisms as fragments of Situation Calculus (or Fluent Calculus) [1] but they reveal some decidability problems. In this paper we propose a decidable alternative which focusses on the theoretical aspects of conceptual structures with Type Theory and which shows how this structure is able to reason about knowledge. This theory exploits a representation of knowledge that is extracted from domain ontologies. The reasoning process is a typing (and subtyping) mechanism which allows one to infer implicitly some knowledge from the knowledge that is explicitly present in the ontology. Already used to solve difficult problems in Natural Language Processing (NLP) [4, 18, 7], Type Theory has proved to be a natural candidate for formalizing linguistic statements as well as real world situations. The logical formalism adopted here is a fragment of the Constructive Type Theory (CTT) [15, 14]. In the second section we summarize the basic mechanisms of the type-theoretic approach centered on the Dependent record Types (DRT) structures (for further details, see for instance [10]). In the third section, we describe the data structures that are at the basis of the reasoning process and in the fourth section, we illustrate the approach by revisiting a context-aware scenario with the type-theoretical approach.

2 Type Theory

2.1 The basis of the Type Theory

In the Curry-Howard correspondence [12], propositions in some logical system are translated into types in the type theory such that derivable propositions give rise to inhabited types. For instance, we can interpret certain types as propositions whereas their inhabitants are representations of proofs for those propositions. As a result, propositions are types and proofs are programs [2]. Under this correspondence, connectives $\top$, $\wedge$ and $\supset$ in propositional logic are respectively expressed by type formers 1, $\times$ and $\rightarrow$ in simple type theory, whereas universal quantifiers $\forall$ and $\exists$ in predicate logic are translated into Π-types and Σ-types in CTT.

Within this knowledge representation formalism, proofs can be checked automatically. A major benefit is the computability of any judgement: constructive theory of types is functionally decidable [19]. The building blocks of CTT are terms and the basic relation is the typing relation. The expression $a : T$ itself is called a judgment. The fundamental notion of typing judgement $a : T$ classifies an object a as being of type T. We call a an inhabitant of T, and we call T the type of a. The context Γ in a judgement $\Gamma \vdash a : T$ contains the prerequisites necessary for establishing the statement $a : T$. Some types are always considered wellformed and are introduced by means of axioms (sorts). We will use two sorts here, $Type$ and $Prop$, which denote respectively 'the sort of types' and 'the sort of propositions'. Dependant types are a way i) of expressing subsets and ii) to enhance the expressive power of the language. The two basic constructors for dependent types are the Π-types and the Σ-types.

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.M : \Pi x : A.B}\ \Pi - \mathsf{intro} \qquad \frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B[M/x]}{\Gamma \vdash < M, N >: \Sigma x : A.B}\ \Sigma - \mathsf{intro}$$

For instance, one may define the following Π-type in order to represent the fact that a bird referred as $titi$ has wings: $has\_wings : (\Pi x : bird.P(x))$ in which $P(x)$ stands for a proposition that depends on x. An instance of the Π-type would be $has\_wings(titi) : P(x)$. Π-types also express the universal quantification $\forall$ and generalize function spaces. Similarly, Σ-types model pairs in which the second component depends on the first. Let us consider the pair $\sigma_1 : \Sigma x : bird.flies(x))$. A proof for the Σ-type σ_1 is given for example by the instance $< titi, q_1 >$ indicating that for an individual $titi$, the proposition is proved (q_1 is a proof of $flies(titi)$).

$$\frac{\Gamma \vdash \sigma : \Sigma x : A.B}{\Gamma \vdash \pi_1(\sigma) : A}\ \pi_1 - \mathsf{elim} \qquad \frac{\Gamma \vdash \sigma : \Sigma x : A.B}{\Gamma \vdash \pi_2(\sigma) : B[\pi_1(\sigma)/x]}\ \pi_2 - \mathsf{elim}$$

The projection rules introduce π_1 and π_2 as elimination rules. A proof $s : \Sigma x : T.p$ in a sum is a pair $s =< \pi_1 s, \pi_2 s >$ that consists of an element $\pi_1 s : T$ of the domain type T together with a proof $\pi_2 s : p[\pi_1 s/x]$ stating that the proposition p is true for this element $\pi_1 s$.

Records are introduced first with the purpose of replacing bound variables (e.g., x) with labels in order to get a more readable and more compact structure, and second to gather within a single structure all the knowledge related to a semantic concept. The basic idea of the present work is to apply the formalism of dependent types to ontological knowledge in order to get a better expressivity than first-order and classical logic formalisms. For that purpose, Dependent Record Types (DRTs) [3, 13] are an extension of Π-types and Σ-types in which types are expressed in terms of data. Dependent record types are much more flexible than simple dependent types such as Π-types and Σ-types [16]. They realize a continuum of precision from the basic assertions we are used to expect from types, up to a complete specification of a representation (e.g., a context).

Definition 1 *A dependent record type is a sequence of fields in which labels l_i correspond to certain types T_i, that is, each successive field type can depend on the values of the preceding fields:*

$$< l_1 : T_1, l_2 : T_2(l_1) \ldots, l_n : T_n(l_1 \ldots l_{n-1}) > \tag{1}$$

where the type T_i may depend on the preceding labels $l_1, ..., l_{i-1}$.

A similar definition holds for record tokens where a sequence of values is such that a value v_i can depend on the values of the preceding fields $l_1, ..., l_{i-1}$:

$$< l_1 = v_1, ..., l_n = v_n > \tag{2}$$

Notice that a dependent record with additional fields not mentioned in the type is still of that type. Another important aspect of the modelling with DRT is that a record can have any number of fields (there is no upper limit). The introduction rule for record types constructs inductively records by adding a new label l[1] and its type T to the previous one provided that the new type is consistent with the logical context Γ ($\rightarrow$ denotes the usual function symbol).

$$\frac{\Gamma \vdash R : record - type \;\; \Gamma \vdash T : R \rightarrow type}{\Gamma \vdash < R, l : T > : record - type} \; record - type - \mathsf{intro} \tag{3}$$

Since contexts are part of situations, the concept of context can be expressed as a Dependent Record Type including individuals as well as propositions[2] [9, 10]. Context types (resulting from an ontology) are distinguished from context objects (resulting from observation). Let us consider the initial situation in which

[1] not already occurring in R.
[2] Propositions are able to represent properties as well as constraints.

an incoming call is processed within an intelligent phone.

$$\underbrace{\begin{bmatrix} x & : person \\ r & : room \\ p_1 & : locatedIn(x,r) \\ b & : building \\ p_2 & : part\_of(r,b) \end{bmatrix}}_{C_1 : Context\ type} \quad \underbrace{\begin{bmatrix} \dots & \\ x & = John \\ r & = ECS210I \\ p_1 & = q_1 \\ b & = ECS \\ p_2 & = q_2 \\ \dots & \end{bmatrix}}_{c_1 : Context\ token}$$

In the record instance, q_1 is a proof of $locatedIn(John, ECS210I)$, and q_2 is a proof that $part\_of(ECS210I, ECS)$.

Pre-defined values can be introduced with manifest types [8].

Definition 2 *Given x of type T, $x : T$, a singleton type T_x is such that:*

$$y : T_x \quad \texttt{iff} \quad y = x \tag{4}$$

Given a record, a manifest field is a field whose type is a singleton type:

$$r : \begin{bmatrix} \dots \\ l = x : T \\ \dots \end{bmatrix} \quad \text{for example :} \quad r : \begin{bmatrix} \dots \\ t_{min} = 11PM : time \\ \dots \end{bmatrix} \tag{5}$$

which means that t_{min} is a label of type *time* having a fixed value of $11PM$.

2.2 Sub-typing

The question of sub-typing requires the knowledge of all possible coercions used for a given term and their precise effect, which is untractable in practice. This problem can be avoided by imposing semantic constraints on coercions [3]: this is the case in record-based subtyping that we shall adopt here.

Definition 3 *Given two record types R and R', if R' contains at least every Σ-type occurring in R and if the types of these common Σ-types are in the subsumption relation then R' is a subtype of R which is written:*

$$R' \sqsubseteq R \tag{6}$$

Every record token of type R' is also a token of type R, since it contains components of appropriate types for all the fields specified in R. Since in type theory the analogue of a proposition is the judgement, we can conclude that the judgement in R is lifted to the judgement in R'. Type inclusion and corresponding proof rules generalize record type inclusion to DRTs.

3 Reasoning with Ontological Knowledge in Type Theory

3.1 Representation of Intentional Concepts

The concept of *context* has no meaning by itself [5] and must be related to an intentional concept: it is ontologically speaking considered as a *moment universal* [11]. Therefore, an intentional concept such as an action, a process, a diagnostic or a project will be functionally added to the context and we speak in that case, of the *context-of* resp. the action, the process, the diagnostic or the project. Using dependent types, an intentional concept is functionally deduced from its context since the basic *function* concept is the typed version of the *entailment* relation in classical logic. With π_1 and π_2 denoting respectively the Σ projection operators resulting from elimination rules, the association between a context type and an intentional concept can be represented by a Σ-type.

Definition 4 *Given a Context Record Type C, an intentional concept is described by a Σ-type such that $\phi : \Sigma\, c : C.IC(c)$ in which c is a valid context, IC is a proposition reflecting the intention and witnessing a proof of the intention achievement.*

It denotes a pair $\phi =< \pi_1\phi, \pi_2\phi >$ that consists in an element $\pi_1\phi : C$ of the domain type of quantification together with a proof $\pi_2\phi : IC[\pi_1\phi/c]$ showing that the intentional proposition IC is proved for this element. In other words, it says that the intentional proposition IC holds within this context. With the example above, the following diagnostic could be proved: $\Sigma c_1 : C_1.locatedIn(c_1.x, c_1.b)$ it relates a record c_1 to a diagnostic that consists of a localization process. We can see the association context + intentional concept as a package from the outside.

3.2 Data structures

A correspondence between CTT and an ontology is established which in turn switches the theory into an internal logic. However, constructing such an ontology requires an appropriate language and we have selected the RDF language (W3C) to take in account the future extension to distributed systems. RDF is able to express labelled graphs with triples $< subject, predicate, object >$ where the subject and object may represent resources (e.g., URIs). This ontology can represent simple types with subjects whose instances are objects related to their types by the predicate "is-of-type". Type constructors, are objects related to the subject "Type" with the same predicate "is-of-type" as above. In such a way, we get a single relation for both types and meta-types (i.e., sorts). The Σ-types are mapped into XML descriptions which themselves describe RDF resources in order to support sharing and reuse. The XML Schema structures are isomorphic to Lisp expressions and allow type inferences within the Theorem Prover. The relations referred to as "has-part-of" predicate arrange Σ-types and DRTs into a hierarchical structure which model easily sub-typing relations (see figure 1).

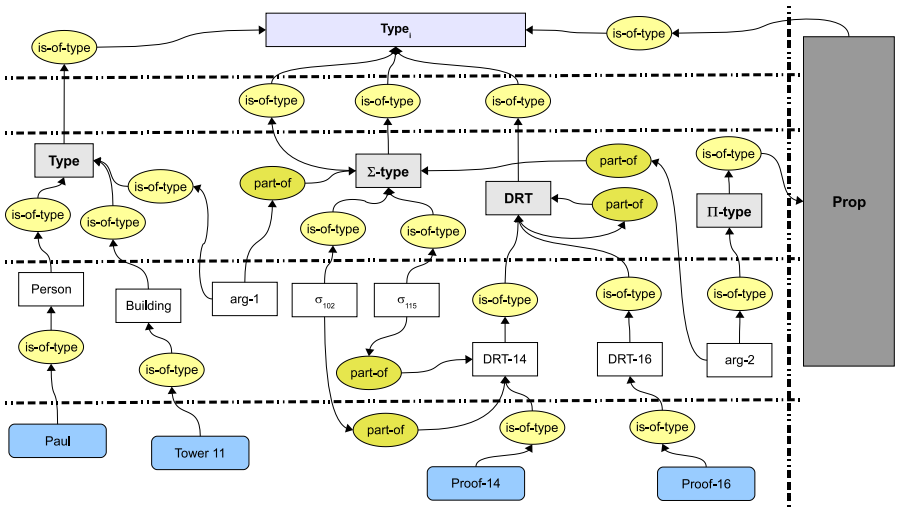

Fig. 1. Typing of basic concepts and relations.

4 Case Study

A key feature of the type-based reasoning is the ability to reason with simple ontological concepts without unnecessary typing and to express compound relations with Σ-types that can be aggregated into DRTs. Let us consider a user named $Harry$ which attends a meeting located in room $ECS210I$ within the ECS building (scenario extracted from [6]. We have to derive from the ontological knowledge that $Harry$ is inside the ECS building. In [17], the authors underline that for such a scenario, OWL offers a mechanism that is not straightforward to cope with composite relationships. Instead of classes and properties as in the classical scheme, we introduce basic concepts with atomic types, simple relations with Σ-types and complex relations with nested Σ-types or DRTs. The natural sub-typing relation between types is the well-known is_a relation. Domain rules can propagate the sub-typing relation to more complex relations such as the $part\_of$ relation.

$$\sigma_1 : \Sigma x : person.\Sigma y : room\ .\ locatedIn(x, y)$$

The Σ-type σ_1 has a proof term $< Harry, < ECS210I, p_1 >>$ where p_1 is the type of proof $locatedIn(Harry, ECS210I)$. Then, assuming the coercion:

$$\frac{\Gamma, x : person, y : room, z : building \vdash locatedIn(x, y)\ \ part\_of(y, z)}{\Gamma \vdash y \sqsubseteq z}$$

Applying this coercion to σ_1, any argument inhabitant of the type $room$ can take inhabitants of the type $building$ as well and therefore, we can derive a proof for $locatedIn(Harry, ECS)$.

The user is located at 16 : 10 in the meeting room (current time) and the meeting is scheduled to be held in the meeting room between 16 : 00 and 17 : 00. We have to deduce that *John* is in a meeting. For that purpose, a DRT including all the required pre-conditions can be designed. Notice that constant values are introduced through manifest fields, yielding complex constraints to be described very simply. Then the DRT is related to a diagnostic (intentional field) as described in section 3.1 and results in a pair σ_1. In other words if the DRT is proved, then the intentional type is proved as well.

$$\sigma_1 : \Sigma c_1 : \begin{bmatrix} t & : time \\ t_1 & = \text{"16 : 00"} : time \\ p_1 & : greaterThan(t, t_1) \\ t_2 & = \text{"17 : 00"} : time \\ p_2 & : lessThan(t, t_2) \\ y & : person \\ z & : meetingRoom \\ p_3 & : locatedInAt(y, z, t) \\ m & : meeting \\ p_4 & : holdIn(m, z) \end{bmatrix} . \; participatesIn(c_1.y, c_1.m)$$

5 Conclusion

On the one hand DRTs depict knowledge based on a support which encode the ontological knowledge via the dependent types. Their high level of expressiveness is obvious due to their wide use in NLP for solving linguistic subtleties. On the other hand, Type-theory is free from both paradoxes and from unnecessary or artificial formalization and it is more appropriate for automatic verification. The theory is able to exploit as much domain knowledge as possible by providing a mechanism by which this knowledge can be acquired, represented through dependent types. One advantage claimed for this approach is that the ontology can be checked for errors in the type-checking system. This approach also seems a good candidate to bridge the gap between a logic formalism for reasoning about actions and the ontological representation of knowledge. As for future work we plan to investigate an intelligent graphical user interface to construct more easily the reasoner.

References

1. F. Baader, C. Lutz, M. Milicic, U. Sattler and F. Wolter. Integrating Description Logics and action formalisms: First results. *Procs. of AAAI'05*, AAAI Press, 572–577, 2005.
2. H. Barendregt. *Handbook of Logic in Computer Science*, volume 2, chapter Lambda Calculi with Types, pages 117–309. Oxford University Press, 1992.
3. G. Betarte. Type checking dependent (record) types and subtyping. *Journal of Functional and Logic Programming*, 10(2):137–166, 2000.
4. P. Boldini. Formalizing context in intuitionistic type theory. *Fundamenta Informaticae*, 42(2):1–23, 2000.
5. P. Brézillon and S. Abu-Hakima. Using Knowledge in Its Context: Report on the IJCAI-93 Workshop. *AI Magazine*, 16(1):87–91, 1995.
6. H. Chen, T. Finin, and Anupam Joshi, Using OWL in a Pervasive Computing Broker, *In Workshop on Ontologies in Open Agent Systems (OAS)*, 9–16, 2003.
7. R. Cooper. Records and record types in semantic theory. *J. Log. Comput.*, 15(2):99–112, 2005.
8. T. Coquand, R. Pollack, and T. M. A logical framework with dependently typed records. *Fundamenta Informaticae*, 20:1–22, 2005.
9. R. Dapoigny and P. Barlatier. Towards a context theory for context-aware systems. In *Procs. of the 2nd IJCAI Workshop on Artificial Intelligence Techniques for Ambient Intelligence*, 2007.
10. R. Dapoigny and P. Barlatier. Goal Reasoning with Context Record Types. In *Procs. of CONTEXT'07*, 164–177, 2007.
11. P. Dockhorn-Costa, J. Paulo A. Almeida, L. F. Pires, G. Guizzardi and M. van Sinderen. Towards Conceptual Foundations for Context-Aware Applications. *Procs. of the AAAI'06 Workshop on Modeling and Retrieval of Context*, 54–58, AAAI Press, 2006.
12. W. A. Howard. *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, chapter The formulae-as-types notion of construction, pages 479–490. Academic Press, 1980.
13. A. Kopylov. Dependent Intersection: A New Way of Defining Records in Type Theory. *Procs. of the 18th Annual IEEE Symposium on Logic in Computer Science*, 86–95, IEEE Computer Society Press, 2003.
14. Z. Luo. A Unifying Theory of Dependent Types : The Schematic Approach. *LFCS*, p. 293–304, 1992.
15. P. Martin-Löf. Constructive mathematics and computer programming. *Logic, Methodology and Philosophy of Sciences*, 6:153–175, 1982.
16. J. McKinna. Why dependent types matter. *SIGPLAN Not.*, 41(1), 2006.
17. L. Ferreira Pires, M. van Sinderen, E. Munthe-Kaas, S. Prokaev, M. Hutschemaekers and D.-J. Plas (editor), *Techniques for describing and manipulating context information*, Freeband/A_MUSE /D3.5v2.0, 2005.
18. A. Ranta. Type-Theoretical Grammar *Oxford University Press*, 1995.
19. S. Valentini. Decidability in intuitionistic type theory is functionally decidable. *Math. Logic*, 42:300–304, 1996.