Dynamic Syntax in Type Theory with Records

Robin Cooper and Staffan Larsson
Centre for Linguistic Theory and Studies in Probability (CLASP)
Dept. of Philosophy, Linguistics and Theory of Science
University of Gothenburg

This abstract sketches how part of Dynamic Syntax (DS, Kempson *et al.*, 2001) can be rendered in TTR, a Type Theory with Records (Cooper, 2012, in prep; Cooper and Ginzburg, 2015) . We explore this as an alternative to DS-TTR (Eshghi, 2015) which adds TTR interpretations to DS but does not code the whole of DS in TTR. DS tree nodes are modelled as records of the type:

(1) $\begin{bmatrix} \text{type} & : & \textit{Type} \\ \text{cont} & : & \text{type} \end{bmatrix}$

However, tree nodes may also have daughters. Therefore we define a (basic, recursive) type *Tree* such that

(2) a:*Tree* iff a:$\begin{bmatrix} \text{type} & : & \textit{Type} \\ \text{cont} & : & \text{type} \\ \text{daughters} & : & \textit{Tree}* \end{bmatrix}$

Here ‘*Tree**’ denotes the type of strings of trees. The ‘*’ is related to the Kleene-*. We will use ‘ϵ’ to represent the empty string.

The TTR type for the tree for *John arrived* is shown in (3).

(3) $\begin{bmatrix} \text{type}=t & : & \textit{Type} \\ \text{cont} & : & \text{type} \\ \text{daughters} & : & \begin{bmatrix} \text{type}=e & : & \textit{Type} \\ \text{cont}=john' & : & \text{type} \\ \text{daughters}=\epsilon & : & \textit{Tree}* \end{bmatrix} ^\frown \begin{bmatrix} \text{type}=e \rightarrow t & : & \textit{Type} \\ \text{cont}=\lambda x\!:\!e\;.\;arrive(x) & : & \text{type} \\ \text{daughters}=\epsilon & : & \textit{Tree}* \end{bmatrix} \end{bmatrix}$

This can be rendered more diagrammatically as in (4).

(4)

$$\begin{bmatrix} \text{type}=t & : & \textit{Type} \\ \text{cont} & : & \text{type} \end{bmatrix}$$

$$\begin{bmatrix} \text{type}=e & : & \textit{Type} \\ \text{cont}=john' & : & \text{type} \end{bmatrix} \quad \begin{bmatrix} \text{type}=e \rightarrow t & : & \textit{Type} \\ \text{cont}=\lambda x\!:\!e\;.\;arrive(x) & : & \text{type} \end{bmatrix}$$

Note that this represents a tree *type*, not a tree (cf. underspecified trees in DS). A record type is *fully specified* iff all its fields are manifest (in TTR notation, a manifest field in a record is written $\ell = v : T$ where ℓ is a label, v is a value of type T). It is *underspecified* otherwise. For example, the type in (4) is underspecified with respect to the path ‘cont’.

Consider the DS lexical entry in (5).

(5)

john:

IF	$?Ty(e)$
THEN	`put`($Ty(e)$)
	`put`($Fo(john')$)
ELSE	`abort`

We might think of (5) as a kind of update rule which refines a type. It might be expressed as something like (6).

(6) If $T_i = \left[\begin{array}{lll} \text{type=e} & : & \textit{Type} \\ \text{cont} & : & \text{type} \end{array}\right]$,

then set T_{i+1} to be $\left[\begin{array}{lll} \text{type=e} & : & \textit{Type} \\ \text{cont=}\textit{john}' & : & \text{type} \end{array}\right]$

We could think of (6) as a *type rewrite rule* and express it in symbols as in (7).

(7)
$$\left[\begin{array}{lll} \text{type=e} & : & \textit{Type} \\ \text{cont} & : & \text{type} \end{array}\right] \Rightarrow \left[\begin{array}{lll} \text{type=e} & : & \textit{Type} \\ \text{cont=}\textit{john}' & : & \text{type} \end{array}\right]$$

This rewrite rule is a *refinement* since the type to the right of the arrow is a subtype of the type to the left. Any monotonic update would be a type refinement of this kind.

An alternative would be to introduce TTR-style content to the lexical entry for *John* as in (8).

(8)
$$\left[\begin{array}{lll} \text{type=}\left[\text{x} : \textit{Ind}\right] & : & \textit{Type} \\ \text{cont} & : & \text{type} \end{array}\right] \Rightarrow$$
$$\left[\begin{array}{lll} \text{type=}\left[\text{x} : \textit{Ind}\right] & : & \textit{Type} \\ \text{cont=}\left[\text{x=john}\right] & : & \text{type} \end{array}\right]$$

In (9), we see the tree type for *John arrived* with TTR content added.

(9)
$$\left[\begin{array}{lll} \text{type=}\textit{RecType} & : & \textit{Type} \\ \text{cont=}\left[\begin{array}{lll} \text{x=john} & : & \textit{Ind} \\ \text{p} & : & \text{arrive(x)} \end{array}\right] & : & \text{type} \end{array}\right]$$

$$\left[\begin{array}{lll} \text{type=}\left[\text{x} : \textit{Ind}\right] & : & \textit{Type} \\ \text{cont=}\left[\text{x=john}\right] & : & \text{type} \end{array}\right] \quad \left[\begin{array}{lll} \text{type=(}\left[\text{x} : \textit{Ind}\right]\rightarrow\textit{RecType}\text{)} & : & \textit{Type} \\ \text{cont=}\lambda r{:}\left[\text{x} : \textit{Ind}\right] . \left[\begin{array}{lll} \text{x=}r\text{.x} & : & \textit{Ind} \\ \text{p} & : & \text{arrive(x)} \end{array}\right] & : & \text{type} \end{array}\right]$$

In the presentation, we will go into some more detail about the above, and also show how contexts and indexical pronouns can be accounted for in TTR-DS. In particular we will explore how the use of tree types will enable us to model DS trees with unfixed nodes using a notion of *component* in a record which may be embedded in a record to any arbitrary depth. We will also give a basic introduction to TTR.

References

Cooper, Robin and Ginzburg, Jonathan 2015. Type theory with records for natural language semantics. In Lappin, Shalom and Fox, Chris, editors 2015, *The Handbook of Contemporary Semantic Theory*. Wiley-Blackwell, second edition. 375–407.

Cooper, Robin 2012. Type theory and semantics in flux. In Kempson, Ruth; Asher, Nicholas; and Fernando, Tim, editors 2012, *Handbook of the Philosophy of Science*, volume 14: Philosophy of Linguistics. Elsevier BV. General editors: Dov M. Gabbay, Paul Thagard and John Woods.

Cooper, Robin prep. Type theory and language: from perception to linguistic communication. Draft of book chapters available from `https://sites.google.com/site/typetheorywithrecords/drafts`.

Eshghi, Arash 2015. DS-TTR: An incremental, semantic, contextual parser for dialogue. *SEMDIAL 2015 goDIAL* 172.

Kempson, Ruth; Meyer-Viol, Wilfried; and Gabbay, Dov 2001. *Dynamic syntax: the flow of language understanding*. Blackwell.