Type raising, continuations, and classical logic

Philippe de Groote
Inria-Lorraine

Abstract. There is a striking analogy between type raising, as introduced by Montague (1973), and the notion of continuation that has been developped in programming language theory in order to give compositional semantics to control operators (Stratchey and Wadsworth, 1974). In fact, this analogy is such that it is possible to see Montague's semantics as a continuation based semantics.

On the other hand, the notion of continuation allows classical logic to be given a Curry-Howard interpretation (Griffin 1990). In particular, the double negation law $((A \to \bot) \to \bot) \to A$ is provided with a computational content, which may be used to give a type logical interpretation of type lowering.

Putting the pieces of the picture together, it is possible to use "classical extensions" of the λ-calculus in order to express the semantic components of the lexical entries of Morrill's (1994) type logical grammars. This solution offers the advantage of not burdening the syntax by enforcing type raising to the worst case.

1 Type raising and continuations

Montague (1973) introduced *type raising* as a way of providing a compositional semantics to constructs that may give rise to scope ambiguities. Such constructs (typically, quantifiers) have semantic scopes that may be wider than their apparent syntactic scopes. Around the same time, computer scientists were trying to provide a compositional semantics to full jumps (i.e., 'goto' statements), which led to the discovery of *continuations* (Stratchey and Wadsworth, 1974).

Both problems are similar, and both solutions present striking similitudes. Montague's type raising is based on Leibniz's principle, which consists of identifying an entity with the set of its properties. Consequently, the type of entities e is replaced by $(e \to t) \to t$, where t is the type of propositions. In programming language theory, a *continuation semantics* (as opposed to a *direct semantics*) consists in providing the semantic function with the continuation of the program as an explicit parameter. Let P be a program, let $[\![-]\!]$ be the semantic function, and let s be some initial state. If we consider programs as state transformers, a direct semantics is such that $[\![P]\!]\, s \in \textbf{State}$. On the other hand, a continuation semantics gives $[\![P]\!]\, s \in (\textbf{State} \to \textbf{State}) \to \textbf{State}$. In fact, in both cases (type raising and continuation semantics), a type A is replaced by a type $(A \to O) \to O$, where O is the type of observable entities or facts.

2 Negative translations and classical logic

In the realm of the λ-calculus, the notion of continuation gave rise to the so-called CPS-transformations (Plotkin 1975). These are continuation-based syntactic transformations of the λ-terms that allow given evaluation strategies (typically, call-by-name or call-by-value) to be simulated.

For instance, Plotkin's call-by-value CPS-transformation is as follows:

$$\begin{aligned}
\overline{c} &= \lambda k.\, k\, c;\\
\overline{x} &= \lambda k.\, k\, x;\\
\overline{\lambda x.\, M} &= \lambda k.\, k\, (\lambda x.\, \overline{M});\\
\overline{M\, N} &= \lambda k.\, \overline{M}\, (\lambda m.\, \overline{N}\, (\lambda n.\, m\, n\, k))
\end{aligned}$$

Now, compare the following naive type logical grammar, where the lexical items are assigned a direct interpretation:

John	–	J	:	NP
Mary	–	M	:	NP
loves	–	$\lambda x.\, \lambda y.\, \text{LOVE}\, y\, x$	:	$(NP \setminus S) / NP$

together with the grammar, where the lexical items are assigned a Montague-like interpretation:

John	–	$\lambda k.\, k\, \text{J}$	:	NP
Mary	–	$\lambda k.\, k\, \text{M}$	:	NP
loves	–	$\lambda f.\, \lambda g.\, f\, (\lambda x.\, g\, (\lambda y.\, \text{LOVE}\, y\, x))$	:	$(NP \setminus S) / NP$

Again, the analogy between continuation and type raising is striking. The Montague-like interpretation may almost be seen as the call-by-value CPS-transform of the direct interpretation. This opens a new line of research that has been advocated in Barker's recent work (2000, 2001).

When applying a CPS-transformation to a typed λ-term, it induced another transformation at the type level (Meyer and Wand, 1985). For instance, the above CPS-transformation induces the following type transformation:

$\overline{\alpha} = (\alpha^* \to \bot) \to \bot$, where:

$$\begin{aligned}
a^* &= a, \quad \text{for } a \text{ atomic};\\
(\alpha \to \beta)^* &= \alpha^* \to \overline{\beta}.
\end{aligned}$$

Griffin (1990) observed that these type transformations amount to double negative translations of classical logic into minimalist logic, and that it allows classical logic to be provided with a formulae-as-type interpretation. In this setting, the double negation law $((A \to \bot) \to \bot) \to A$, which corresponds to type lowering, is given a computational content by considering the absurd type $\bot$ to be the type of observable entities (this is radically different from the usual interpretation of $\bot$ as the empty type).

3 The $\lambda\mu$-calculus

Griffin's discovery gave rise to several extensions of the λ-calculus, which aim at adapting the Curry-Howard isomorphism to the case of classical logic. The $\lambda\mu$-calculus (Parigot 1992) is such a system.

The $\lambda\mu$-calculus is a strict extension of the λ-calculus. Its syntax is provided with a second alphabet of variables ($\alpha, \beta, \gamma, \ldots$ — called the μ-variables), and two additional constructs: μ-abatraction ($\mu\alpha.\, t$), and naming ($\alpha\, t$).
These constructs obey the following typing rules:

$$\frac{\alpha : \neg A \qquad t : A}{\alpha\, t : \bot} \qquad\qquad \frac{\begin{matrix}[\alpha : \neg A]\\ t : \bot\end{matrix}}{\mu\alpha.\, t : A}$$

Besides β-reduction:

$(\beta)\;\; (\lambda x.\, t)\, t \;\longrightarrow\; t[x := u]$

a notion of μ-reduction is defined:

$(\mu)\;\; (\mu\alpha.\, u)\, v \;\longrightarrow\; \mu\beta.\, u[\alpha\, t_i := \beta\,(t_i\, v)]$

where $u[\alpha\, t_i := \beta\,(t_i\, v)]$ stands for the term u where each subterm of the form $\alpha\, t_i$ has been replaced by $\beta\,(t_i\, v)$. It corresponds to the following proof-theoretic reduction:

$$\dfrac{\dfrac{\begin{array}{c}\dfrac{\dfrac{}{\alpha : \neg(A\to B)}\,1 \qquad \begin{array}{c}\vdots\\ t_i : A\to B\end{array}}{\alpha\, t_i : \bot}\\ \vdots \\ u : \bot\end{array}}{\mu\alpha.\, u : A\to B}\,1 \qquad \begin{array}{c}\vdots\\ v : A\end{array}}{(\mu\alpha.\,u)\, v : B} \quad\longrightarrow\quad \dfrac{\begin{array}{c}\dfrac{\dfrac{}{\beta : \neg B}\,1 \quad \dfrac{\begin{array}{c}\vdots\\ t_i : A\to B\end{array}\quad \begin{array}{c}\vdots\\ v : A\end{array}}{t_i\, v : B}}{\beta\,(t_i\, v) : \bot}\\ \vdots\\ u[\alpha\, t_i := \beta\,(t_i\, v)] : \bot\end{array}}{\mu\beta.\, u[\alpha\, t_i := \beta\,(t_i\, v)] : B}\,1$$

As well-known, classical logic is not naturally confluent. Consequently, there exist variants of the $\lambda\mu$-calculus that do not satisfy the Church-Rosser property (Parigot 2000). This is the case if we also consider the symmetric of the μ-reduction rule:

$(\mu')\;\; v\,(\mu\alpha.\, u) \;\longrightarrow\; \mu\beta.\, u[\alpha\, t_i := \beta\,(v\, t_i)]$

Finally, for the purpose of the example given in the next section, we also add the following simplification rules:

$(\sigma)\;\; \mu\alpha.\, u \;\longrightarrow\; u[\alpha\, t_i := t_i]$

which may be applied only to terms of type $\bot$.

4 Semantic recipes as $\lambda\mu$-terms

Dealing with a calculus that do not satisfy the Church-Rosser property is not a defect in the case of natural language semantics. Indeed, the fact that a same term may have several different normal forms allows one to deal with semantic ambiguities.

If we consider the sentential category S (or, semantically, Montague's type t) to be our domain of observable facts, the following typing judgement is derivable:

$$\dfrac{\dfrac{\dfrac{\dfrac{\textsc{Person} : e\to t \quad \dfrac{}{x : e}\,0}{(\textsc{Person}\, x) : t} \qquad \dfrac{\dfrac{}{\alpha : e\to t}\,1 \quad \dfrac{}{x : e}\,0}{(\alpha\, x) : t}}{(\textsc{Person}\, x)\supset(\alpha\, x) : t}}{\forall x.(\textsc{Person}\, x)\supset(\alpha\, x) : t}\,0}{\mu\alpha.\,\forall x.(\textsc{Person}\, x)\supset(\alpha\, x) : e}\,1$$

This allows the following type logical lexical entries to be defined:

everybody	–	$\mu\alpha.\,\forall x.(\textsc{person}\,x) \supset (\alpha\,x)$	:	NP
somebody	–	$\mu\alpha.\,\exists x.(\textsc{person}\,x) \wedge (\alpha\,x)$	:	NP
loves	–	$\lambda x.\,\lambda y.\,\textsc{love}\,y\,x$	:	$(NP \setminus S) \,/\, NP$

Then, the sentence

everybody loves somebody

has only one parsing, to which is associated the following semantic reading:

$$(\lambda x.\,\lambda y.\,\textsc{love}\,y\,x)\,(\mu\alpha.\,\exists x.(\textsc{person}\,x) \wedge (\alpha\,x))\,(\mu\alpha.\,\forall x.(\textsc{person}\,x) \supset (\alpha\,x)).$$

This $\lambda\mu$-term may be considered as an underspecified representation. Indeed, its possible reductions yield two different normal forms:

$$\begin{array}{lll}
\multicolumn{2}{l}{(\lambda x.\,\lambda y.\,\textsc{love}\,y\,x)\,(\mu\alpha.\,\exists x.(\textsc{person}\,x) \wedge (\alpha\,x))\,(\mu\alpha.\,\forall x.(\textsc{person}\,x) \supset (\alpha\,x))} & \\
\longrightarrow & (\lambda y.\,\textsc{love}\,y\,(\mu\alpha.\,\exists x.(\textsc{person}\,x) \wedge (\alpha\,x)))\,(\mu\alpha.\,\forall x.(\textsc{person}\,x) \supset (\alpha\,x)) & (\beta) \\
\longrightarrow & \textsc{love}\,(\mu\alpha.\,\forall x.(\textsc{person}\,x) \supset (\alpha\,x))\,(\mu\alpha.\,\exists x.(\textsc{person}\,x) \wedge (\alpha\,x)) & (\beta) \\
\longrightarrow & (\mu\beta.\,\forall x.(\textsc{person}\,x) \supset (\beta\,(\textsc{love}\,x)))\,(\mu\alpha.\,\exists x.(\textsc{person}\,x) \wedge (\alpha\,x)) & (\mu') \\
\longrightarrow & \mu\beta.\,\forall x.(\textsc{person}\,x) \supset (\beta\,(\textsc{love}\,x\,(\mu\alpha.\,\exists y.(\textsc{person}\,y) \wedge (\alpha\,y)))) & (\mu) \\
\longrightarrow & \forall x.(\textsc{person}\,x) \supset (\textsc{love}\,x\,(\mu\alpha.\,\exists y.(\textsc{person}\,y) \wedge (\alpha\,y))) & (\sigma) \\
\longrightarrow & \forall x.(\textsc{person}\,x) \supset (\mu\alpha.\,\exists y.(\textsc{person}\,y) \wedge (\alpha\,(\textsc{love}\,x\,y))) & (\mu') \\
\longrightarrow & \forall x.(\textsc{person}\,x) \supset (\exists y.(\textsc{person}\,y) \wedge (\textsc{love}\,x\,y)) & (\sigma)
\end{array}$$

$$\begin{array}{lll}
\multicolumn{2}{l}{(\lambda x.\,\lambda y.\,\textsc{love}\,y\,x)\,(\mu\alpha.\,\exists x.(\textsc{person}\,x) \wedge (\alpha\,x))\,(\mu\alpha.\,\forall x.(\textsc{person}\,x) \supset (\alpha\,x))} & \\
\longrightarrow & (\lambda y.\,\textsc{love}\,y\,(\mu\alpha.\,\exists x.(\textsc{person}\,x) \wedge (\alpha\,x)))\,(\mu\alpha.\,\forall x.(\textsc{person}\,x) \supset (\alpha\,x)) & (\beta) \\
\longrightarrow & \textsc{love}\,(\mu\alpha.\,\forall x.(\textsc{person}\,x) \supset (\alpha\,x))\,(\mu\alpha.\,\exists x.(\textsc{person}\,x) \wedge (\alpha\,x)) & (\beta) \\
\longrightarrow & (\mu\beta.\,\forall x.(\textsc{person}\,x) \supset (\beta\,(\textsc{love}\,x)))\,(\mu\alpha.\,\exists x.(\textsc{person}\,x) \wedge (\alpha\,x)) & (\mu') \\
\longrightarrow & \mu\alpha.\,\exists y.(\textsc{person}\,y) \wedge (\alpha\,((\mu\beta.\,\forall x.(\textsc{person}\,x) \supset (\beta\,(\textsc{love}\,x)))\,y)) & (\mu') \\
\longrightarrow & \exists y.(\textsc{person}\,y) \wedge ((\mu\beta.\,\forall x.(\textsc{person}\,x) \supset (\beta\,(\textsc{love}\,x)))\,y) & (\sigma) \\
\longrightarrow & \exists y.(\textsc{person}\,y) \wedge (\mu\beta.\,\forall x.(\textsc{person}\,x) \supset (\beta\,(\textsc{love}\,x\,y))) & (\mu) \\
\longrightarrow & \exists y.(\textsc{person}\,y) \wedge (\forall x.(\textsc{person}\,x) \supset (\textsc{love}\,x\,y)) & (\sigma)
\end{array}$$

These correspond to subject and object wide scope readings, respectively.

5 conclusions

We have argued that Montague's type raising is a particular case of continuation. Consequently, continuation based formalisms, which have been developped in the context of programming language theory, may be used to deal with the sort of semantic ambiguities for which Montague invented type raising. Parigot's $\lambda\mu$-calculus is such a formalism, and we have shown how it may be used to cope with quantifier scope ambiguities. We claim that the $\lambda\mu$-calculus is particularly suitable for expressing compositional semantics of natural languages. For instance, it allows Cooper's (1983) storage to be given a type logical foundation. In fact, it allows a lot of dynamic constructs to be defined, which is of particular interest for discourse representation.

references

Barker, C. (2000) *Continuations and the nature of quantification* working paper, submitted for publication.

Barker, C. (2001) Introducing Continuations. In R. Hastings, B. Jackson, and Z. Zvolenszky, editors, *Proceedings of SALT 11*, CLC Publications, Ithaca, New York.

Cooper, R. (1983). *Quantification and Syntactic Theory.* Dordrecht: Reidel.

Griffin, T. G. (1990). A formulae-as-types notion of control. In *Conference record of the seventeenth annual ACM symposium on Principles of Programming Languages*, pages 47–58.

Meyer, A. and Wand, M. (1985). Continuation semantics in typed lambda-calculi (summary). In R. Parikh, editor, *Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 219–224. Spinger Verlag.

Montague, R. (1973). The proper treatment of quantification in ordinary english. In J. Hintikka, J. Moravcsik, and P. Suppes, editors, *Approaches to natural language: proceedings of the 1970 Stanford workshop on Grammar and Semantics*, Dordrecht, Reidel. Reprinted: Montague, (1974, pages 247–270).

Montague, R. (1974). *Formal Philosophy: selected papers of Richard Montague, edited and with an introduction by Richmond Thomason.* Yale University Press.

Morrill, G. (1994). *Type Logical Grammar: Categorial Logic of Signs.* Kluwer Academic Publishers, Dordrecht.

Parigot, M. (1992). $\lambda\mu$-Calculus: an algorithmic interpretation of classical natural deduction. In A. Voronkov, editor, *Proceedings of the International Conference on Logic Programming and Automated Reasoning*, volume 624 of *Lecture Notes in Artificial Intelligence*, pages 190–201. Springer Verlag.

Parigot, M. (2000). On the computational interpretation of negation. In P.G. Clote and H. Schwichtenberg, editors, *Computer Science Logic*, volume 1862 of *Lecture Notes in Computer Science*, Springer Verlag.

Plotkin, G. D. (1975). Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159.

Stratchey, C. and Wadsworth, C. (1974). Continuations a mathematical semantics for handling full jumps. Technical Report PRG-11, Oxford University, Computing Laboratory.