Intensions as Computable Functions

Shalom Lappin[1]

Classical intensional semantic frameworks, like Montague's Intensional Logic (IL), identify intensional identity with logical equivalence. This criterion of co-intensionality is excessively coarse-grained, and it gives rise to several well known difficulties. Theories of fine-grained intensionality have been been proposed to avoid this problem. Several of these provide a formal solution to the problem, but they do not ground this solution in a substantive account of intensional difference. Applying the distinction between operational and denotational meaning, developed for the semantics of programming languages, to the interpretation of natural language expressions, offers the basis for such an account. It permits us to escape some of the complications generated by the traditional modal characterization of intensions.

1 Introduction

Classical intensional semantic representation languages, like Montague (1974)'s Intensional Logic (IL) do not accommodate fine-grained intensionality. Montague, following Carnap (1947), characterizes intensions as functions from worlds (indices of worlds and times) to denotations, and so reduces intensional identity to equivalence of denotation across possible worlds. Logically equivalent expressions are semantically indistinguishable. This is too course a criterion for semantic identity. Logical equivalence is not a sufficient condition for intersubstitutability in all contexts.

(1) a. Every prime number is divisible only by itself and 1. <=>

b. If $A \subseteq B$ and $B \subseteq A$, then $A = B$.

[1]King's College London

LiLT Volume 9
Perspectives on Semantic Representations for Textual Inference.

(2) a. John believes that every prime number is divisible only by itself and 1. <≠>

b. John believes that if $A \subseteq B$ and $B \subseteq A$, then $A = B$.

To avoid this difficulty a fine-grained theory of intensionality must be able to distinguish between provable equivalence and intensional identity.

2 Intensional Identity

Fox and Lappin (2005, 2010) propose Property Theory with Curry Typing (PTCT) as an alternative intensional semantic representation framework. It is a first-order system that consists of three components: (i) an untyped λ-calculus, which generates the language of terms, (ii) a rich Curry typing system for assigning types to terms, (iii) and a first-order language of well-formed formulas for reasoning about the truth of propositional terms, where these are term representations of propositions. A tableaux proof theory constrains the interpretation of each component of this federative representation language, and it relates the expressions of the different components. Restrictions on each component prevent semantic paradoxes. A model theory allows us to prove the soundness and completeness of the proof theory.

The terms of the untyped λ-calculus encode computable functions. These correspond to the intensions of the representation language. Identity in the λ-calculus is defined in terms of the α, β, and η conditions for substitution.

PTCT uses two notions of equality: intensional identity and extensional equivalence. $t \cong_T s$ states that the terms t, s are extensionally equivalent in type T. In the case where two terms t, s are propositions ($t, s \in \mathsf{Prop}$), then $t \cong_{\mathsf{Prop}} s$ corresponds to $t \leftrightarrow s$. If two predicates of T are extensionally equivalent ($t \cong_{(T \Longrightarrow \mathsf{Prop})} s$), then t, s each hold of the same elements of T. Therefore $\forall x(x \in T \rightarrow ({}^{\mathsf{T}}t(x) \leftrightarrow {}^{\mathsf{T}}s(x)))$, where ${}^{\mathsf{T}}t(x)$ asserts that the proposition represented by the term $t(x)$ is true.

$t =_T s$ states that two terms are intensionally identical in type T. As noted, the rules for intensional identity are essentially those of the $\lambda\alpha\beta\eta$-calculus. We are able to derive $t =_T s \rightarrow t \cong_T s$ for all types inhabited by t, s, but not $t \cong_T s \rightarrow t =_T s$. Therefore PTCT avoids the reduction of provable equivalence to intensional identity. Two terms can be provably equivalent by the proof theory, but not identical. In this case, they remain intensionally distinct.

PTCT allows us to sustain both the logical equivalence of (1)a and (1)b, and the non-equivalence of (2)a and (2)b. The former are provably

equivalent, but they correspond to non-identical propositional terms in PTCT.

The proof theory of PTCT induces a prelattice on the terms in Prop. In this prelattice the members of an equivalence class of mutually entailing propositional terms (terms that encode mutually entailing propositions) are non-identical and so correspond to distinct propositions.[2] While this result achieves the formal property of fine-grained intensionality, it does not, in itself, explain what intensional non-identity consists in, beyond the fact that two distinct expressions in the language of terms are identified with different intensions. This leaves us with what we can describe as a problem of ineffability. Intensional difference is posited as (a certain kind of) inscriptional distinctness in the λ-calculus of terms, but this reduction does not offer a substantive explanation of the semantic properties that ground the distinction. Intensional difference remains ineffable.

This is an instance of a general problem with inscriptional treatments of fine-grained intensionality.[3] They identify differences of meaning with distinctions among terms in a semantic representation language. But without an account of how difference in terms generates intensional distinction, the inscriptional view leaves intensional non-identity unexplained. Inscriptionalist theories avoid problems created by the characterisation of intensions as functions on possible worlds at the risk of rendering intensions primitive to the point of inscrutability.

3 Expressing Intensional Difference Operationally

We can characterize the distinction between intensional identity and provable equivalence computationally by invoking the contrast between operational and denotational semantics in programming language. Two simple examples illustrate this contrast.

For the first example take the function $predecessorSet(x)$, which maps an object in an ordered set into the set of its predecessors. So, for example, if $x \in \{0, 1, 2, 3, 4, 5\}$, $predecessorSet(x) = PredSet_x \subset \{0, 1, 2, 3, 4, 5\}$ such that $\forall y \in PredSet_x(y < x)$. It follows that $predecessorSet(0) = \emptyset$.

It is possible to define (at least) two variants of this function, $predecessorSet_a$ and $predecessorSet_b$, that are denotationally equivalent but operationally distinct. $predecessorSet_a$ is specified directly

[2]Fox et al. (2002); Fox and Lappin (2005); Pollard (2008) construct higher-order hyperintensional semantic systems using an extended version of Church's SST and a prelattice of propositions in which the entailment relation is a preorder.

[3]See Fox and Lappin (2005) for a discussion of inscriptionalist theories of intentionality.

in terms of an immediate predecessor relation, while $predecessorSet_b$ depends upon a successor relation.

(3) a. $predecessorSet_a(x) = PredSet_x$, if
$\forall y(y \in PredSet_x \rightarrow predecessor(y, x))$.
b. $predecessor(y, x)$ if
$predecessor_{immediate}(y, x)$; else
(i) $predecessor(y, x)$ if
$predecessor_{immediate}(y, z)$, and
$predecessor(z, x)$.

(4) a. $predecessorSet_b(x) = PredSet_x$, if
$\forall y(y \in PredSet_x \rightarrow successor(x, y))$.
b. $successor(x, y)$ if
$successor_{immediate}(x, y)$; else
(i) $successor(x, y)$ if
$successor_{immediate}(x, z)$, and
$successor(z, y)$.

The second example involves functions $g : \Sigma^* \rightarrow \{1, 0\}$ from Σ^*, the set of strings formed from the alphabet of a language, to the Boolean values 1 and 0, where $g(s) = 1$ if $s \in L$, and 0 otherwise. Let g_{csg1} be defined by the Definite Clause Grammar (DCG) in (5), and g_{csg2} by the DCG in (6).[4]

(5) $S \rightarrow [a],\ S(i)$.
$S(I) \rightarrow [a],\ S(i(I))$.
$S(I) \rightarrow Bn(I),\ Cn(I)$.
$Bn(i(I)) \rightarrow [b],\ Bn(I)$.
$Bn(i) \rightarrow [b]$.
$Cn(i(I)) \rightarrow [c],\ Cn(I)$.
$Cn(i) \rightarrow [c]$.

(6) $S \rightarrow A(I),\ B(I),\ C(I)$.
$A(i) \rightarrow [a]$.
$A(i(I)) \rightarrow [a],\ A(I)$.
$B(i) \rightarrow [b]$.
$B(i(I)) \rightarrow [b],\ B(I)$.
$C(i) \rightarrow [c]$.
$C(i(I)) \rightarrow [c],\ C(I)$.

[4]See Pereira and Shieber (1987) for an explanation of Definite Clause Grammars. The DCG in (5) is from Gazdar and Mellish (1989). Matthew Purver and I constructed the DCG in (6) as a Prolog programming exercise for a computational linguistics course that I gave in the Computer Science Department at King's College London in 2002.

Both these DCGs define the same context-sensitive language

$$\{a^n b^n c^n |\ 1 \leq n\},$$

the language whose strings consist of n occurrences of a, followed by n bs, and then n cs. The number of as, bs, and cs match in all strings. Each DCG uses a counting argument I for a non-terminal symbol to build up a stack of indices i that gives the successive number of occurrences of as, bs, and cs in a string. But the grammar in (5) counts from the bottom up, adding an i for each non-terminal that the recognizer encounters. By contrast the grammar in (6) imposes the requirement that the three stacks for the non-terminals A, B, and C be identical, and then it computes the indices top down. The two grammars are computationally distinct, and using each of them to recognize a string can produce different sequences of operations, of different lengths and relative efficiency. Therefore, g_{csg1} and g_{csg2} are operationally distinct, but denotationally equivalent. They compute the same string set through different sets of procedures.

4 Computable Functions and Natural Language Expressions

Recall that the terms of PTCT are λ-expressions that encode computable functions. We have identified these with the intensions of words and phrases in a natural language. Given the distinction between denotational and operational meaning we can now interpret the non-identity of terms in the representation language as an operational difference in the functions that these terms express. But a class of such terms can still be provably equivalent in the sense that they yield the same values for the same arguments by virtue of the specifications of the functions that they correspond to. This provides a straightforward account of fine-grained intensionality in PTCT which avoids taking intensional difference as ineffable.

It is reasonable to ask what it could mean to characterise the interpretation of a natural language expression as a computable function. Rich type theories, like PTCT and Type Theory with Records (TTR, Cooper (2012)), are based on the type systems used in programming languages. In some of these systems propositions are identified with proofs, where the proof of a proposition is the procedure applied to establish that it is assertable.[5] This can be a formal procedure, like the application of the rules of a proof theory, but it need not be. It could also be the sequence of operations involved in making the observations

[5]See Martin-Löf (1984) for a type theory in which propositions are characterised as proofs in a formal system.

that support the application of a classifier predicate to an object or an event.

On the view proposed here we are taking the semantic content of terms in a natural language to be the functions that we use to compute the denotations of these expressions. If such a term is a predicate, then the function that corresponds to its meaning encodes the procedure through which we determine the values that it returns for its domain of arguments (n-tuples of arguments for relational predicates).

Two predicates may correspond to distinct functions which happen to yield the same values for each argument in a given domain, but they would diverge if defined for an alternative domain. This would be the situation for predicates that are contingently co-extensive. However, as we have seen in Section 3, it is possible for two (or more) distinct computable functions to be provably equivalent. In this case they will generate the same range of values for all domains for which they are defined, through different sequences of operations, by virtue of the way in which these sequences are specified.

5 Two Alternative Operational Approaches

Muskens (2005) suggests a similar approach to hyperintensionality. He identifies the intension of an expression with an algorithm for determining its extension.[6] There are two major points of difference between Musken's theory and the one proposed here. First, he embeds his account in a logic programming approach, which he seems to take as integral to his explanation of hyperintensionality, while I have developed my analysis in a functional programming framework. This is, in fact, not an issue of principle. The same algorithm can be formulated in any programming language. So, for example, the definitions of $predecessorSet_a$ and $predecessorSet_b$ correspond to two Horn clause definitions in Prolog for variant predecessor predicates, `predecessorA(Y, X)` and `predecessorB(Y, X)`.

(7)
```
predecessorA(Y, X) :- predecessorImmediate(Y, X).
predecessorA(Z, X) :-
predecessorImmediate(Y, X),
predecessorA(Y, Z).
```

(8)
```
predecessorB(Y, X) :- successor(X, Y).
```

[6] Duží et al. (2010) also adopt an operational view of hyperintensionality within Tichý (1988)'s Transparent Intensional Logic. However, the computational details of their account are left largely unspecified. Both Muskens (2005) and Duží et al. (2010) regard their respective proposals as working out Frege (1892)'s idea that a sense is a rule for identifying the denotation of an expression.

```
successor(X, Y) :- successorImmediate(X, Y).
successor(X, Z) :-
successorImmediate(X, Y),
successor(Y, Z).
```

Similarly, the DCGs in (5) and (6) that we used to define g_{csg1} and g_{csg2}, respectively, are (close to) Prolog executable code.

However, the functional programming formulation of the operational view of fine-grained intensionality follows straightforwardly from PTCT, where the untyped λ-calculus generates the intensional terms of the semantic representation language, and these encode computable functions. PTCT also offers rich Curry typing with weak polymorphism, and a logic of wffs for reasoning about truth and entailment, within a first-order system. The fact that it implies the operational account of intensional difference without further stipulation renders it attractive as a framework for developing computational treatments of natural language semantic properties.

The second, more substantive point of difference concerns the role of possible worlds in characterizing intensions. Muskens develops his hyperintensional semantics on the basis of Thomason (1980)'s Intentional Logic. In this logic Thomason proposes a domain of propositions as intensional objects, where the set of propositions is recursively defined with intensional connectives and quantifiers. He posits a homomorphism that maps propositions (and their constituents) to their extensions, and he constrains this homomorphism with several meaning postulates that restrict this mapping.[7] Muskens modifies and extends Thomason's logic by specifying a homomorphism between the intensional expressions of the logic and their extensions across the set of possible worlds. Propositions are mapped to the set of worlds in which they are true. As the homomorphism can be many-to-one, distinct propositions can receive the same truth-value across worlds.[8]

By contrast, PTCT adopts Thomason's possible worlds-free strategy of mapping propositions to truth-values. It does this by using a truth

[7]Fox and Lappin (2005) point out that Thomason's logic is problematic because it does not characterize the algebraic structure of the domain of propositions. It does not offer a proof theory that defines entailment for propositions, and so it leaves the relation between intentional identity and extensional equivalence crucially under determined.

[8]Fox et al. (2002); Fox and Lappin (2005); Pollard (2008) adopt a similar view for the fine-grained higher-order logics that they construct. They define worlds as untrafilters in the prelattice of propositions, and they take the truth of a proposition, relative to a world, to be its membership in such an ultrafilter. As entailment in the prelattice is defined by a preorder, distinct propositions can belong to the same set of ultrafilters.

predicate to form a wff $^{\mathsf{T}}(\phi)$ to assert the truth of the proposition that the term $\phi \in \mathsf{Prop}$ represents. Therefore, like Intentional Logic, PTCT de-modalizes intensions. This is a positive result. It is not clear why, on the fine-grained view, possible worlds must be essentially connected with the specification of intensions.

On both Musken's account and the one proposed here, the content of an intension is the set of computational operations through which it determines its denotational value, where these do not make essential reference to possible worlds. In the case of a proposition, the denotation that it determines is a truth-value, rather than a truth-value relative to a world. There may be independent epistemic, or even semantic reasons for incorporating possible worlds into one's general theory of interpretation, but worlds are not required for an adequate explanation of fine-grained intensionality. On the contrary, such an account must dispense with the original characterization of intensions as functions from worlds to extensions in order to explain the persistence of intensional difference beyond provable equivalence. Therefore, a radically possible worlds-free view of fine-grained intensionality offers the cleaner approach.

Moschovakis (2006) proposes an operational treatment of meaning within the framework of the typed λ-calculus. He constructs a language L^{λ}_{ar} as an extension of Gallin (1975)'s Ty_2. He specifies acyclic recursive procedures for reducing the terms of L^{λ}_{ar} to unique cannonical forms, and he identifies the meaning ("referential intension") of a term in this language with the "abstract algorithm" for computing its denotation.

There are two major points of difference between Moschovakis' algorithmic theory of intensions and the account proposed here. First, while in PTCT α, β, and η reduction sustain intensional identity, in L^{λ}_{ar} β reduction does not. His primary motivation for this move seems to be his concern to maintain the non-synonymy of sentences like *John loves himself* on one hand, and those like *John loves John* on the other. In L^{λ}_{ar} the canonical form of the former is $(\lambda(x)loves(x,x))(j)\ where\ j := john$, while that of the latter is $loves(j_1, j_2)\ where\ j_1 := John, j_2 := John$.

But this issue would appear to be an artifact of the way that Moschovakis has chosen to formalize proper names and reflexive pronouns. If one represented them as distinct sorts of generalized quantifiers, or constants, then this problem would not arise. In any case, it is not a deep question of principle. We take α, β, and η reduction to support intensional identity because they are normalizing operations on λ-terms in the semantic representation language, and so they do not correspond, in any obvious way, to processes or relations of natural languages. However, it is perfectly possible to narrow the specification

of intensional identity in PTCT to exclude β (as well as η, and even α) reduction, without altering the proposed account of intensions as computable functions. This would simply involve imposing a particularly fine-grained notion of intensional identity.

The second point of difference is more significant. Moschovakis specifies a Kripke frame semantics for L^{λ}_{ar} which is a variant of Montague's possible worlds models (he refers to them as "Carnap states"). These are n-tuples of indices corresponding to worlds, times, speakers, and other parameters of context. Intensions are characterized as algorithmic procedures for determining the denotation of a term relative to a world and the other elements of such an n-tuple. Therefore, like Muskens Moschovakis' operational view of intensions treats them as inextricably bound up with possible worlds. The arguments that I brought against this view in Muskens' case apply with equal force here. An important advantage of the proposed account is that it factors modality and possible worlds out of the specification of intensions.

6 Conclusion

While theories of fine-grained intensionality may avoid the reduction of intensional identity to provable equivalence, many of them do not go beyond a bare inscriptionalist treatment of intensional difference. Therefore they leave this notion ineffable. On the proposal developed here intensional difference is the operational distinctions among computable functions, and extensional identity is the denotational equivalence of the values that functions compute. This account grounds fine-grained intensionality in a way that naturally accommodates cases of intensional difference combined with provable denotational equivalence.

Given that PTCT uses the untyped λ-calculus to generate the Curry typed term representations for the intensions of the language, and these terms encode computable functions, the proposed operational characterization of intensional difference is already implicit in this semantic framework.

This account yields a radically non-modal view of intensions in which possible worlds play no role in their specification or their interpretation. An intension is identified directly with the sequence of operations performed in computing the value of the function that expresses it. Fine-grained intensionality becomes the operational contents of computable functions.

Acknowledgements

An earlier version of this paper appeared as "An Operational Approach to Fine-Grained Intensionality" in Thomas Graf, Denis Paperno, Anna Szabolcsi, and Jos Tellings (eds.), *Theories of Everything: In Honor of Ed Keenan, UCLA Working Papers in Linguistics 17*, 2012, under the terms of a Creative Commons Non-Commercial License (`http://creativecommons.org/licenses/by-nc/3.0/`).

The main ideas in this paper developed out of my NASSLLI 2012 course Alternative Paradigms of Computational Semantics, and from talks that I gave at the University of Gothenburg in April 2012, and in December 2012. I am grateful to the participants in the course and to the audiences of the talks for stimulating feedback. I would also like to thank Robin Cooper, Chris Fox, and Dag Westerståhl for very helpful discussion of some of the issues addressed here. A more detailed presentation of the approach proposed here is given in Lappin (forthcoming).

References

Carnap, R. 1947. *Meaning and Necessity*. Chicago: University of Chicago Press.

Cooper, Robin. 2012. Type theory and semantics in flux. In R. Kempson, T. Fernando, and N. Asher, eds., *Philosophy of Linguistics*, pages 271–323. Amsterdam: Elsevier.

Duží, M., B. Jespersen, and P. Materna. 2010. *Procedural Semantics for Hyperintensional Logic*. Dordrecht, New York: Springer.

Fox, C. and S. Lappin. 2005. *Foundations of Intensional Semantics*. Oxford: Blackwell.

Fox, C. and S. Lappin. 2010. Expressiveness and complexity in underspecified semantics. *Linguistic Analysis, Festschrift for Joachim Lambek* 36:385–417.

Fox, C., S. Lappin, and C. Pollard. 2002. A higher-order, fine-grained logic for intensional semantics. In G. Alberti, K. Balough, and P. Dekker, eds., *Proceedings of the Seventh Symposium for Logic and Language*, pages 37–46. Pecs, Hungary.

Frege, Gottlob. 1892. On sense and reference. In P. Geach and M. Black, eds., *Translations from the Philosophical Writings of Gottlob Frege, 3rd Edition, 1980*, pages 56–78. Oxford: Basil Blackwell.

Gallin, D. 1975. *Intensional and Higher-Order Modal Logic*. Amsterdam: North-Holland.

Gazdar, G. and C. Mellish. 1989. *Natural Language Processing in Prolog*. Waltham, MA: Addison-Wesley.

Lappin, Shalom. forthcoming. Curry typing, polymorphism, and fine-grained intensionality. In S. Lappin and C. Fox, eds., *The Handbook of Contempo-*

rary Semantic Theory, Second Edition. Malden, MA and Oxford: Wiley-Blackwell.

Martin-Löf, Per. 1984. *Intuitionistic Type Theory*. Napoli: Bibliopolis.

Montague, R. 1974. *Formal Philosophy: Selected Papers of Richard Montague*. New Haven, CT/London, UK: Yale University Press. Edited with an introduction by R. H. Thomason.

Moschovakis, Y. 2006. A logical calculus of meaning and synonymy. *Linguistics and Philosophy* 29:27–89.

Muskens, R. A. 2005. Sense and the computation of reference. *Linguistics and Philosophy* 28:473–504.

Pereira, F. and S. Shieber. 1987. *Prolog and Natural-Language Analysis*. Stanford, CA: CSLI.

Pollard, Carl. 2008. Hyperintensions. *Journal of Logic and Computation* 18:257–282.

Thomason, R. 1980. A model theory for propositional attitudes. *Linguistics and Philosophy* 4:47–70.

Tichý, P. 1988. *The Foundations of Frege's Logic*. Berlin: De Gruyter.