Dynamic Contextual Intensional Logic: Logical Foundations and an Application

Richmond H. Thomason

Philosophy Department
University of Michigan
Ann Arbor, MI 48109-2110, USA
rich@thomason.org

Abstract. This paper develops further aspects of Contextual Intensional Logic, a type-theoretic logic intended as a general foundation for reasoning about context. I motivate and formulate the dynamic version of the logic, prove some results about it, and show how it can be used to specify inter-contextual reasoning.

1 Introduction

Since 1997, I have advocated a type-theoretic approach to the logical formalization of context, which incorporates ideas of Alonzo Church, Richard Montague, and David Kaplan. The most complete available presentation of the core formalism, Contextual Intensional Logic (CIL), can be found in [1], along with references to the earlier work.

As soon as you have the idea of context, you need the idea of context dynamics. This need shows up in its purest form in the semantics of programming languages, where the context is an assignment of values to variables, and where the languages contain expressions (like '$x := 2$', i.e., 'Let x be 2') that change the assignment. Since all that a program can do between receiving an input and producing an output is to change variable assignments, there could be no nontrivial programs without context-changing operators.

Such phenomena are pervasive in natural language as well: an utterance can change the context that then figures in the interpretation of subsequent utterances. In retrospect, you can see this as a central insight of speech act theory; [2] contains one of the first explicit formulations of the point in the philosophical literature. Of course, this idea led to the enormous amount of work that has been invested in developing the various dynamic approaches to natural language semantics. And in computational applications of context, you want reasoning not only to depend on context, but to manipulate it.

2 Static CIL

CIL is a static logic; but one of the lessons of dynamic semantics is that a more or less standard logic can be dynamicized; see, for instance, [3]. In particular, dynamic versions of Intensional Logic have been produced as part of the project of making Montague grammar dynamic; see [4]. Accounting for context-shifting effects by making Contextual Intensional Logic dynamic is therefore a very natural and attractive program.

This program is sketched in [1], which contains a formulation of the logic DCIL. Here, the logic is formulated in greater detail, illustrated with an extended example, and is generalized to incorporate a record of the series of contexts that a reasoner has traversed; this generalization is needed to accommodate McCarthy's "exit" operation.

CIL is based on the architectural principles that underlie Montague's Intensional Logic [5], and is in fact an extension of IL. The syntax and model theory of (static) IL can be summarized as follows.

2.1 Types

(2.1.1) e is a type.
(2.1.2) t is a type.
(2.1.3) i is a type.
(2.1.4) If σ and τ are types, so are $\langle\sigma,\tau\rangle$ and $\langle w,\tau\rangle$.

Here, e is the type of individuals, t is the type of truth-values, and i is the type of indices. Indices are simultaneous disambiguations of every linguistic expression whose interpretation depends on context, including indexical expressions and lexically ambiguous expressions. Thus, indices are similar to "contexts", as David Kaplan uses the term. A type of the form $\langle w,\tau\rangle$ represents functions from possible worlds to objects of type τ. In this formulation there is no primitive type of possible worlds, although worlds can enter into complex types.

2.2 Syntax of CIL

A CIL language L is a function from the set of CIL types to nonempty sets of expressions; where τ is a type, L_τ is the set of basic constants of type τ. For each type τ, we postulate a denumerable set Var_τ of variables of type τ. (This set is the same for all CIL languages.) $L_\tau \cup Var_\tau$ is the set of basic expressions of type τ.

The following recursion extends L to a function L^* taking each type into the set of (basic or complex) expressions of that type.

Basic expressions: $L_\tau \cup Var_\tau \subseteq L^*_\tau$.

Identity: If ζ and $\xi \in L^*_\tau$, then $\zeta = \xi \in L^*_t$.

Functional application: If $\zeta \in L^*_{\langle\sigma,\tau\rangle}$ and $\xi \in L^*_\sigma$, then $\zeta(\xi) \in L^*_\tau$.

Lambda abstraction: If $\zeta \in L^*_\tau$, then $\lambda x_\sigma \zeta \in L^*_{\langle\sigma,\tau\rangle}$.

Intension: If $\zeta \in L^*_\tau$, then ${}^\wedge\zeta \in L^*_{\langle w,\tau\rangle}$.

Extension: If $\zeta \in L^*_{\langle w,\tau\rangle}$, then ${}^\vee\zeta \in L^*_\tau$.

Character formation: If $\zeta \in L^*_\tau$ then ${}^\sqcap\zeta \in L^*_{\langle i,\tau\rangle}$.

Content determination: If $\zeta \in L^*_{i,\tau}$ then ${}^\sqcup\zeta \in L^*_\tau$.

2.3 Domains of Arbitrary Type

Let D be a function taking the basic types e, w, and i into nonempty sets, or *domains.* The following recursive definition extends D to an assignment of a domain to arbitrary types. In this definition, $\top$ and $\bot$ are arbitrary different individuals, taken to stand for truth and falsity; I will assume that these two elements belong only to the domain D_t.

(2.3.1) $D_t = \{\top, \bot\}$.
(2.3.2) $D_{\langle\sigma,\tau\rangle} = D_\tau^{D_\sigma}$.

The organization of type theory around a family of domains that are constructed using the operation in (2.3.2) of forming a domain by taking the set of functions from one domain to another is due to [6]. Functions prove to be a powerful and useful organizing idea for such a logic. Montague extended Church-Henkin type theory by adding a domain of possible worlds; CIL extends Montague's logic by adding a domain of indices. In each case, the model theory is a straightforward extension of the one provided by Church. (Also, see [7].)

2.4 Models

A (static) model $\mathcal{M}$ of L on a domain assignment D is an assignment of a member $[\![x_\tau]\!]_\mathcal{M}$ of D_τ to each variable x_τ in Var_τ, and of a member $[\![\xi]\!]_{\mathcal{M},\mathrm{i},\mathrm{w}}$ of D_τ to each basic constant in L_τ, for each $\mathrm{i} \in D_i$ and $\mathrm{w} \in D_w$. (Note that, unlike the values of constants, the values of variables are world- and index-independent.) In the rule that interprets lambda abstraction (clause (2.4.3), below), we appeal to the following device of semantic substitution for the variable x_τ: where $\mathrm{d} \in D_\tau$, $\mathcal{M}^{\mathrm{d}}/x$ is the model $\mathcal{M}'$ that is like $\mathcal{M}$ except that $[\![x]\!]_{\mathcal{M}'} = \mathrm{d}$.

Such a model assigns a value $[\![\xi]\!]_{\mathcal{M},\mathrm{i},\mathrm{w}} \in \tau$ to each expression $\xi \in L^*_\tau$, for each $\mathrm{i} \in D_i$ and $\mathrm{w} \in D_w$. This assignment conforms to the following rules of semantic interpretation, which uniquely determine the values assigned to complex expressions, given an assignment of values to basic expressions.

(2.4.0) For $x_\tau \in Var_\tau$, $[\![x_\tau]\!]_{\mathcal{M},\mathrm{i},\mathrm{w}} = [\![x_\tau]\!]_\mathcal{M}$.
(2.4.1) $[\![\zeta = \xi]\!]_{\mathcal{M},\mathrm{i},\mathrm{w}} = \top$ if $[\![\zeta]\!]_{\mathcal{M},\mathrm{i},\mathrm{w}} = [\![\xi]\!]_{\mathcal{M},\mathrm{i},\mathrm{w}}$. Otherwise $[\![\zeta = \xi]\!]_{\mathcal{M},\mathrm{i},\mathrm{w}} = \bot$.
(2.4.2) $[\![\zeta(\xi)]\!]_{\mathcal{M},\mathrm{i},\mathrm{w}} = [\![\zeta]\!]_{\mathcal{M},\mathrm{i},\mathrm{w}}([\![(\xi)]\!]_{\mathcal{M},\mathrm{i},\mathrm{w}})$.
(2.4.3) Where $\zeta \in L^*_\tau$ and $x_\sigma \in Var_\sigma$, $[\![\lambda x_\sigma \zeta]\!]_{\mathcal{M},\mathrm{i},\mathrm{w}} =$ the function f from D_σ to D_τ such that $\mathrm{f}(\mathrm{d}) = [\![\zeta]\!]_{\mathcal{M}^{\mathrm{d}}/x,\mathrm{i},\mathrm{w}}$.
(2.4.4) Where $\zeta \in L^*_\tau$, $[\![{}^\wedge\zeta]\!]_{\mathcal{M},\mathrm{i},\mathrm{w}} =$ the function f from D_w to D_τ such that $\mathrm{f}(\mathrm{w}') = [\![\zeta]\!]_{\mathcal{M},\mathrm{i},\mathrm{w}'}$.
(2.4.5) Where $\zeta \in L^*_{\langle w,\tau\rangle}$, $[\![{}^\vee\zeta]\!]_{\mathcal{M},\mathrm{i},\mathrm{w}} = [\![\zeta]\!]_{\mathcal{M},\mathrm{i},\mathrm{w}}(\mathrm{w})$.
(2.4.6) Where $\zeta \in L^*_\tau$, $[\![{}^\sqcap\zeta]\!]_{\mathcal{M},\mathrm{i},\mathrm{w}} =$ the function f from D_i to D_τ such that $\mathrm{f}(\mathrm{i}') = [\![\zeta]\!]_{\mathcal{M},\mathrm{i}',\mathrm{w}}$.
(2.4.7) Where $\zeta \in L^*_{i,\tau}$, $[\![{}^\sqcup\zeta]\!]_{\mathcal{M},\mathrm{i},\mathrm{w}} = [\![\zeta]\!]_{\mathcal{M},\mathrm{i},\mathrm{w}}(\mathrm{i})$.

In CIL, contexts are considered to involve two separate components: (1) a modal component, which has type $\tau_M = \langle\langle w,t\rangle,\langle w,t\rangle\rangle$ and (2) an indexical component, which has type i. According to McCarthy's account of contexts, a context c performs two distinct functions: it provides a source of specialized knowledge, and it serves to disambiguate the meanings of expressions. The two components of CIL contexts represent these two functions. The first component is a function that inputs a proposition and outputs a proposition; given a proposition p, this function returns the proposition that p is known in the context. The second component is an index; indices represent disambiguation policies. Separating these two functions of context is enforced by the type-theoretic framework of CIL. But the distinction also seems to be a highly useful representational and conceptual device.

Definition 2.1. *Contexts and their components.*

A *context*, relative to a domain assignment D, is a pair $\langle \mathrm{M}, \mathrm{i}\rangle$, where $\mathrm{M} \in D_{\tau_M}$ and $\mathrm{i} \in D_i$.[1] The modal component M of a context $\mathrm{c} = \langle \mathrm{M}, \mathrm{i}\rangle$ is denoted by c_M; the indexical component i is denoted by c_i. (Therefore, $\mathrm{c} = \langle \mathrm{c}_M, \mathrm{c}_i\rangle$.)

3 Dynamic CIL

Many natural forms of contextual reasoning involve navigating contextual space. If someone standing on the opposite side of the table tells us to move a pitcher on the table to the right, we may have to imagine things from their perspective to execute the instruction properly. [8] suggest that operations of entering and exiting contexts may be useful in many forms of contextual reasoning.

Dynamic logic (see, for instance, [9]) provides a natural way to integrate change of context into a logical formalism, and, in particular, to provide a logical semantics for context-changing operators. In typical formalizations of dynamic logic, the context is taken to be an assignment of values to variables; but it is perfectly possible to apply the ideas of dynamic logic to the contexts of CIL. That is the project that we will undertake here.

The project of modifying CIL to obtain Dynamic Contextual Intensional Logic (DCIL) is unlike Montague's project of extending the Church-Henkin type theory to IL, and my similar expansion of IL to CIL. In the latter cases, the basic model theoretic techniques are carried over without much change, though the ontology of the models is expanded by adding new primitive domains, and some constructions may be added to the language.

On the other hand, DCIL uses the same types and domains as CIL, and adds no new constructions to the syntax. But there are additions to the parameters on which denotations depend which result in rather fundamental changes to the logic.

We can best explain the nature of the change by concentrating on sentence-like formulas—expressions of type t. A model $\mathcal{M}$ of CIL assigns a denotation

[1] Recall that τ_M is $\langle\langle w,t\rangle,\langle w,t\rangle\rangle$.

$[\![\phi]\!]_{\mathcal{M},\mathrm{i},\mathrm{w}}$ in $\{\top, \bot\}$to a formula $\phi \in L_t^*$. Abstracting on the parameters i and w on which the truth-value depends we can repackage the interpretation of ϕ by saying that $\mathcal{M}$ assigns a set of index-world pairs to ϕ. Or, fixing world parameter w, CIL assigns a set S of indices to ϕ relative to w.

Generalizing this last idea, let's think of the interpretation S of ϕ at w as a set of contexts. Recall that, for us, contexts are modality-index pairs; so that we are simply adding another parameter to the interpretation, which represents a modality. In CIL, this addition is vacuous, because there are no formulas whose interpretation depends on the modality component. Nevertheless, the generalization provides a useful bridge to the dynamic logic.

According to this way of looking at the static logic, then, a sentential formula (relative to a model and a world) corresponds to the set of contexts in which it is true.

In dynamic logic, we are interested in expressions like 'enter context c' that enforce a change of context. We can interpret such expressions as sets of changes of context, where a change of context is simply a pair $\langle c, c' \rangle$ of contexts. This means that in dynamic CIL we want models to assign values to expressions relative to a world and *a pair* of contexts: $[\![\zeta]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}}$ is the value assigned to ζ relative to a world w and the contexts c and c$'$.

The intuitions guiding the extension of a static logic to one that is dynamic are not always as robust as one would like, especially when we are dealing with moderately powerful logics. In the case of DCIL, intuitions about the dynamic interpretation of higher types are not always entirely clear, and the fact that we are working without truth-value gaps also creates some difficulties. But the core of the following development of DCIL is, I believe, quite plausible. In explaining the logic I will concentrate on this core, and will try to show in terms of an example that the logic is potentially useful in specifying certain forms of reasoning about context.

3.1 Types and Syntax of DCIL

The types and the syntax of DCIL are the same as in CIL; there are no new syntactic constructions.

The complex expressions of DCIL are formed as in CIL. We add two basic expressions with fixed model-theoretic interpretations: $@_i$, which has type i, and $@_M$, which has type τ_M.[2] $@_i$ denotes the index of the current context, and $@_M$ denotes the modality of the current context.

These two new constants are (in a sense to be explained) static. Later, we will add other constants which are nontrivially dynamic.

[2] The symbol '@' is sometimes used in modal logic to denote the actual world ; this is a generalization of that notation to other types of semantic parameters.

3.2 Model Theory of DCIL

Definition 3.2. *Dynamic models.*

A *dynamic model* $\mathcal{M}$ of L on a triple $\langle D_e, D_w, D_i\rangle$ of domains is an assignment of a member $[\![\xi]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}} \in D_\tau$ to each basic expression $\xi \in L_\tau$, for each $c, c' \in D_{\tau_M} \times D_i$ and each $\mathrm{w} \in D_w$.

This definition allows expressions of any type to denote nontrivial sets of context changes. For instance, a model $\mathcal{M}$ could assign a constant a of type e the following value.

$[\![a]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}} = \mathrm{a}$ if and only if $\mathrm{c}' = \langle \mathrm{M}, \mathrm{i}\rangle$, and
$[\![a]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}} = \mathrm{a}'$ if and only if $\mathrm{c}' = \langle \mathrm{M}', \mathrm{i}'\rangle$.

I can't think of any useful purpose that such a constant could serve. It seems that our core intuitions concerning dynamic interpretations only concern expressions of type t. No doubt this is related to the fact that it is utterances that that have the potential to change the context, and utterances are sentential.

In any case, the only plausible dynamic constructions—and the only ones with which we will be concerned here—have the sentential type t or, like dynamic conjunction, have a functional type that produces type t values.

We want the other expressions of DCIL to be vacuously or trivially dynamic; that is, we want these expressions to be static. In dynamic logic, it would be most natural to treat a static expression as one that is undefined except at identity transitions (transitions from c to c). But in the version of DCIL with which we are working, there are no "denotation gaps"; in every model $\mathcal{M}$, every expression ζ receives a denotation at every world, for every transition from c to c′. This is a logic development policy; adding denotation gaps is a complex matter that I would prefer to address in a separate paper, or series of papers. For the present, then, we have to find a solution to the problem of characterizing vacuously dynamic expressions in a gapless logic.

In the case of expressions of type t, it is natural to say that a static formula is false except at identity transitions. This definition must be generalized to expressions of arbitrary type. For this purpose, for each type τ we need to define a "null denotation" of this type. The following definition accomplishes this.

Definition 3.3. *Null value* $\perp_{\mathcal{M},\tau}$.

The null value of type τ is defined by the following recursion.

Basis. $\perp_{\mathcal{M},t} = \perp$; $\perp_{\mathcal{M},e} \in \mathrm{Dom}_{\mathcal{M}}(e)$; $\perp_{\mathcal{M},w} \in \mathrm{Dom}_{\mathcal{M}}(w)$; $\perp_{\mathcal{M},i} \in \mathrm{Dom}_{\mathcal{M}}(i)$.

Induction. $\perp_{\mathcal{M},\langle\sigma,\tau\rangle}$ is the constant function f from $\mathrm{Dom}_{\mathcal{M}}(\sigma)$ to $\mathrm{Dom}_{\mathcal{M}}(\tau)$ such that $\mathrm{f(d)} = \perp_{\mathcal{M},\tau}$ for all $\mathrm{d} \in \mathrm{Dom}_{\mathcal{M}}(\sigma)$.

Definition 3.4. *Static expression.*

An expression ζ is *static* relative to a model $\mathcal{M}$ iff for all c, c′, w, if $\mathrm{c} \neq \mathrm{c}'$ then $[\![\zeta]\!]_{M,c,c',w} = \perp_{\mathcal{M},\tau}$.

The semantic rules for the basic expressions $@_M$ and $@_i$ are as follows.

If $\mathrm{c} = \mathrm{c}'$ then $[\![@_M]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}} = \mathrm{c}_M$; otherwise, $[\![@_M]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}} = \bot_{\mathcal{M},\tau_M}$.
If $\mathrm{c} = \mathrm{c}'$ then $[\![@_i]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}} = \mathrm{c}_i$; otherwise, $[\![@_i]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}} = \bot_{\mathcal{M},i}$.

Note that according to these rules, $@_M$ and $@_i$ are interpreted as static expressions.

The denotations of complex expressions in DCIL for the most part are characterized by clauses that look just like the corresponding clauses for CIL. The only exceptions are clause (3.2.2) for identity, and clause (3.2.6), for character formation. Both clauses render their constructions static. These restrictions are hard to motivate using direct intuitions, but they simplify things, and they are crucial for the proof of Theorem 3.1.

The semantic rules are as follows.

(3.2.0) For $x_\tau \in \mathit{Var}_\tau$, $[\![x_\tau]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}} = [\![x_\tau]\!]_{\mathcal{M}}$.
(3.2.1) Where ζ and ξ are expressions in L^*_τ, $[\![\zeta = \xi]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}} = \top$ iff $c = c'$ and $[\![\zeta]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}} = [\![\xi]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}}$.
(3.2.2) Where $\zeta \in L^*_{\langle\sigma,\tau\rangle}$ and $\xi \in L^*_\sigma$,
$$[\![\zeta(\xi)]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}} = [\![\zeta]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}}([\![(\xi)]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}}).$$
(3.2.3) Where $\zeta \in L^*_\tau$, $[\![\lambda x_\sigma \zeta]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}} =$ the function f from D_σ to D_τ such that $\mathrm{f}(\mathrm{d}) = [\![\zeta]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}}\mathcal{M}^{\mathrm{d}}/x$.
(3.2.4) Where $\zeta \in L^*_\tau$, $[\![{}^\wedge\zeta]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}} =$ the function f from D_w to D_τ such that $\mathrm{f}(\mathrm{w}') = [\![\zeta]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}'}$.
(3.2.5) Where $\zeta \in L^*_{\langle w,\tau\rangle}$, $[\![{}^\vee\zeta]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}} = [\![\zeta]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}}(\mathrm{w})$.
(3.2.6) Where $\zeta \in L^*_\tau$, if $\mathrm{c} = \mathrm{c}'$ then $[\![{}^\sqcap\zeta]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}} =$ the function f from D_i to D_τ such that $\mathrm{f}(\mathrm{i}) = [\![\zeta]\!]_{\mathcal{M},\langle c_M,i\rangle,\langle c_M,i\rangle,\mathrm{w}}$. If $\mathrm{c} \neq \mathrm{c}'$ then $\mathrm{f} = \bot_{\mathcal{M},\langle i,\tau\rangle}$.
(3.2.7) Where $\zeta \in L^*_{\langle i,\tau\rangle}$, then $[\![{}^\sqcup\zeta]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}} = [\![\zeta]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}}(\mathrm{c}_i)$.

The following result shows that DCIL is, in a sense, a conservative extension of CIL.

Definition 3.5. *Converting static to dynamic models.*
Let $\mathcal{M}$ be a static model of a language L (without $@_M$ or $@_i$).[3] The *dynamic equivalent* $\mathit{dyn}(\mathcal{M})$ of $\mathcal{M}$ is the dynamic model $\mathcal{M}'$ defined as follows:

If x_τ is a variable of type τ, then $[\![x_\tau]\!]_{\mathcal{M}'}$ is $[\![x_\tau]\!]_{\mathcal{M}}$.
If $\zeta_\tau \in L^*_\tau$ is a constant of type τ, then $[\![\zeta]\!]_{\mathcal{M}',\mathrm{c},\mathrm{c}',\mathrm{w}}$ is $[\![\zeta]\!]_{\mathcal{M}',\mathrm{c}_i,\mathrm{w}}$ if $\mathrm{c} = \mathrm{c}'$, and is $\bot_{\mathcal{M},\tau}$ otherwise.

Definition 3.6. $\mathit{cl}(\mathcal{M})$.
Let $\mathcal{M}$ be a (static or dynamic) model. Then $\mathit{cl}(\mathcal{M})$ is the smallest set of models containing $\mathcal{M}$ and such that if $\mathcal{M}' \in \mathit{cl}(\mathcal{M})$, $x \in \mathit{Var}_\tau$, and $\mathrm{d} \in D_\tau$, then $\mathcal{M}'\,{}^{\mathrm{d}}/x \in \mathit{cl}(\mathcal{M})$.

Theorem 3.0. For any static model $\mathcal{M}$ of L, the dynamic equivalent $\mathit{dyn}(\mathcal{M})$ of $\mathcal{M}$ is equivalent to $\mathcal{M}$, in the sense that for any type τ and any $\zeta \in L^*_\tau$, if $[\![\zeta]\!]_{\mathit{dyn}(\mathcal{M}),\mathrm{c},\mathrm{c}',\mathrm{w}} \neq \bot_{\mathcal{M},\tau}$ then $\mathrm{c} = \mathrm{c}'$ and $[\![\zeta]\!]_{\mathit{dyn}(\mathcal{M}),\mathrm{c},\mathrm{c}',\mathrm{w}} = [\![\zeta]\!]_{\mathcal{M},\mathrm{c}_i,\mathrm{w}}$.

[3] I have left out $@_M$ and $@_i$ to simplify things; the theorems apply if they are present, but CIL would need to be reformulated to accommodate these constants.

Proof. Let $\mathcal{M}_1$ be a static model. The proof is a straightforward induction on the syntactic complexity of expressions ζ. The hypothesis of induction is that the following holds for all $\zeta \in L^*_\tau$:
For all $\mathcal{M}'_1 \in cl(\mathcal{M}_1)$, if $c = c'$ then
$[\![\zeta]\!]_{dyn(\mathcal{M}'_1),\mathrm{c},\mathrm{c}',\mathrm{w}} = [\![\zeta]\!]_{\mathcal{M}'_1,c_i,\mathrm{w}}$, and if $c \neq c'$
then $[\![\zeta]\!]_{dyn(\mathcal{M}'_1),\mathrm{c},\mathrm{c}',\mathrm{w}} = \bot_{\mathcal{M}',\tau}$.
The proof of the theorem consists in verifying the hypothesis of induction for each type of expression; each of the cases is straightforward.

A vacuous dynamic model is one in which the modal constituent of a context has no effect on semantic values, and in which all constants are interpreted as tests.

Definition 3.7. *Vacuous dynamic models.*
A dynamic model $\mathcal{M}$ is *vacuous* in case (1) for all τ and all $\zeta \in L_\tau$, $[\![\zeta]\!]_{\mathcal{M},\langle\mathrm{M},\mathrm{i}\rangle,\langle\mathrm{M},\mathrm{i}\rangle,\mathrm{w}} = [\![\zeta]\!]_{\mathcal{M},\langle\mathrm{M}',\mathrm{i}\rangle,\langle\mathrm{M}',\mathrm{i}\rangle,\mathrm{w}}$ for all modalities $\mathrm{M}, \mathrm{M}' \in D_{\tau_M}$ and (2) for all constants $\zeta \in L_\tau$, $[\![\zeta]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}} = \bot_{\mathcal{M},\tau}$ if $c \neq c'$.

Theorem 3.0. Every vacuous dynamic model is generated by a static model.
Proof. If $\mathcal{M}$ is vacuous, let $\mathcal{M}'$ be the static model that assigns variables the same values as $\mathcal{M}$, and that assigns constants ζ in L_τ the value $[\![\zeta]\!]_{\mathcal{M}',\mathrm{i},\mathrm{w}} = [\![\zeta]\!]_{\mathcal{M},\langle\mathrm{M},\mathrm{i}\rangle,\langle\mathrm{M},\mathrm{i}\rangle,w}$, where M is an arbitrary modality in D_{τ_M}.
It is not difficult to see that $\mathcal{M} = dyn(\mathcal{M}')$.

Putting Theorems 3.1 and 3.2 together, it follows that every vacuous dynamic model is equivalent to a static model.

4 Adding Some Dynamic Basic Expressions

For the dynamics of DCIL to be at all useful, we clearly must add some expressive power. We do this by adding nontrivially dynamic basic expressions.

4.1 Static Boolean Connectives

Church-style type theories in general, including CIL, provide powerful definitional mechanisms: in particular, boolean operations and quantifiers of arbitrary type can be defined using only identity and lambda abstraction. The following definitions are taken from [10]. The definition for $\wedge$ can be a bit puzzling; what it says is that Boolean conjunction is the function that gives $\top$ to arguments x and y if and only if every function that gives value y to x also gives $\top$ to $\top$.

$$\begin{aligned}
&\top =_{\mathrm{df}} \lambda x_t\, x = \lambda x_t\, x\\
&\bot =_{\mathrm{df}} \lambda x_t\, x_t = \lambda x_t\, \top]\\
&\neg =_{\mathrm{df}} \lambda x_t\, [x_t = \bot]\\
&\wedge =_{\mathrm{df}} \lambda x_t \lambda y_t\, [\lambda z_{\langle t,t\rangle}\, [z(x) = y] = \lambda z_{\langle t,t\rangle}\, [z(\top)]]\\
&\forall_\tau =_{\mathrm{df}} \lambda x_{\langle \tau,t\rangle}\, [x = \lambda y_\tau \top]
\end{aligned}$$

These definitions were designed for a static logic; when incorporated into a dynamic type theory like CIL, they produce static or "test" operations, which produce expressions ζ such that if $c \neq c'$ then $[\![\zeta]\!]_{M,c,c'} w = \bot_{\mathcal{M},\tau}$.

4.2 Nontrivial Dynamicism

To begin with, I will introduce three new constants: ;, M-ENTER, and I-ENTER. ; is dynamic 'and'; its definition is standard. McCarthy proposed a context-shifting operator that allowed one to enter a designated context. Since in DCIL, contexts are divided into two components, the process of entering a context is divided between two dynamic operators. M-ENTER changes the modality of the current context, and I-ENTER changes its index.

The new constants and their formal interpretations are summarized as follows.

;, a basic expression of type $\langle t, \langle t, t\rangle\rangle$.[4]
$[\![;(\phi)(\psi)]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}} = \top$ iff there is a c'' such that $[\![\phi]\!]_{\mathcal{M},c,c'',\mathrm{w}}$ and $[\![\psi]\!]_{\mathcal{M},c'',c',\mathrm{w}}$.

M-ENTER, a basic expression of type $\langle \tau_M, t\rangle$.
Where $\zeta \in L^*\tau_M$ and $\mathrm{c}_i = \mathrm{i}$, $[\![\text{M-ENTER}(\zeta)]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}} = \top$ iff $c' = \langle [\![\zeta]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}}, \mathrm{i}\rangle$.

I-ENTER, a basic expression of type $\langle i, t\rangle$.
Where $\zeta \in L^*\tau_i$ and $\mathrm{c}_M = \mathrm{M}$, $[\![\text{I-ENTER}(\zeta)]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}} = \top$ iff $c' = \langle \mathrm{M}, [\![\zeta]\!]_{\mathcal{M},\mathrm{c},\mathrm{c}',\mathrm{w}}\rangle$.

5 A Database Example

I will use the database example from [1] to illustrate how DCIL can be used in the specification of applications. In this example, Ann, Bob and Charlie use personal databases to manage their calendars. The databases use the first-person pronoun to refer to the database user. They also have constants referring to Ann, Bob and Charlie. The databases contain information about meetings; it is important for the databases to agree on scheduled meetings. The databases communicate in order to ensure this. The purpose of the example is to make a convincing case that DCIL could be useful in specifying this communication process.

5.1 Database-Level Formalizations

In [1], I distinguished three levels at which the databases can be formalized. The *database format level* uses a representation that is close to the one actually manipulated by the databases. At this level, for instance, Ann's database might represent a meeting with Bob at 9 o'clock in the form of a triple $\langle I, b, 9\rangle$, and represent the fact that Bob is aware of this meeting as a quadruple $\langle b, I, a, 9\rangle$. (This second triple says that Bob's database contains a triple $\langle I, a, 9\rangle$; so here, I refers to Bob.)

[4] We use the more familiar notation '$\phi;\psi$' in place of '$;(\phi)(\psi)$'.

5.2 Knowledge-Level Formalizations

The *knowledge level* describes the databases in much the same way that indirect discourse is used in natural language. The reference of I is fixed by the initial context, and modalities are used to characterize what various databases know. If we describe the databases at the knowledge level from a neutral context that is free of indexicals, all occurrences of I will have to be replaced by nonindexical equivalents. Using an expression $\textsc{Meet}$ of type $\langle e, \langle e, \langle e, t\rangle\rangle\rangle$ for the 3-place meeting relation recorded by the databases, and $\Box a$ for the modality associated with Ann's database, the knowledge-level formulation of Ann's entry $\langle I, b, 9\rangle$ is $[a](^\wedge\textsc{Meet}(a_e, b_e, 9))$. And the knowledge-level formulation of Ann's entry $\langle b, I, a, 9\rangle$ is $[a]([b](^\wedge\textsc{Meet}(a_e, b_e, 9)))$.

5.3 The *ist* Operator and Context-Level Formalizations

The *context level* formalization of Ann's database, uses *ist*. For McCarthy, *ist* is a relation between contexts and sentences (or perhaps their meanings), and keeps track of the sentences that hold in contexts. In DCIL, *ist* is a function that inputs a modality and an index (the components of a context) and a propositional character (that is, a function from indices to propositions), and that outputs a proposition.[5] Therefore, *ist* has the type

$$\langle i, \langle \tau_M, \langle \tau\text{-Char-Prop}, \tau\text{-Prop}\rangle\rangle\rangle,$$

where τ-Prop $= \langle w, t\rangle$ and τ-Char-Prop $= \langle i, \tau\text{-Prop}\rangle$. *ist* can be defined as follows, using lambda abstraction.

$$(5.3.1)\ \ \mathit{ist} = \lambda u_i \lambda x_{\tau_M} \lambda p_{\tau\text{-Char-Prop}}\, x(p(u)).$$

Definition (5.3.1) gives rise to the following rule of *ist*-conversion.

$$(5.3.2)\ \ \mathit{ist}(\eta)(\zeta)(\phi) \text{ is equivalent to } \zeta(\phi(\eta)).$$

At the knowledge level, the formulation of Ann's entry $\langle I, b, 9\rangle$ is $[a]\textsc{Meet}(a_e, b_e, 9)$. The context-level formulation of this same entry $\langle I, b, 9\rangle$ is $^\vee\mathit{ist}(a_i, [a], {}^{\ulcorner\wedge}\textsc{Meet}(I, b_e, 9))$, and the context-level formulation of $\langle b, I, a, 9\rangle$ is $^\vee\mathit{ist}(a_i, [a], {}^\ulcorner\mathit{ist}(b_i, [b], {}^{\ulcorner\wedge}\textsc{Meet}(I, a_e, 9)))$.

DCIL provides another level—the *procedural level*—at which protocols can be formalized and proved correct. To illustrate this use of DCIL, suppose that Ann's database maintains its information about Bob's database by regularly communicating with Bob's database and fetching records. For instance, this procedure, on finding $\langle I, a, 9\rangle$ in Bob's database, will record an entry $\langle b, I, a, 9\rangle$ in Ann's database.

[5] In the example that I will develop, for instance, *ist* would input the modality of knowledge relative to Ann's database, the indices that disambiguates expressions using the policies of Ann's database, and the character associated with the sentence 'I have a meeting with Bob at 9', and would return the proposition that Ann has a meeting with Bob at 9.

Suppose that this procedure revises Ann's records about Bob's database by replacing them with a copy of Bob's meeting calendar. Then for all formulas $\textsc{Meet}(I, \zeta, \xi)$, where ζ and ξ are constants of type τ-Char-Prop = $\langle i, \tau\text{-Prop}\rangle$ (these are the formulas that could express the characters of propositions about Bob's meetings in his calendar), the following will hold immediately after the procedure has been applied. The sense of (5.3.3) is that in the context of Ann's database, Ann's database knows about Bob's entry for a meeting if and only if visiting Bob's database produces this entry.

$$
\begin{array}{lll}
(5.3.3) & [@_M = \texttt{[}a\texttt{]} \wedge @_i = a_i] \rightarrow & \%\,\text{Start in Ann's DB} \\
& [{}^{\vee}ist(a_i, \texttt{[}a\texttt{]}, {}^{\sqcap}ist(b_i, \texttt{[}b\texttt{]}, {}^{\sqcap\wedge}\textsc{Meet}(I,\zeta,\xi))) \leftrightarrow & \\
& \text{M-}\textsc{enter}(\texttt{[}b\texttt{]}); \text{I-}\textsc{enter}(b_i); & \%\,\text{Enter Bob's DB} \\
& {}^{\vee}@_M([{}^{\sqcap\wedge}\textsc{Meet}(I,\zeta,\xi)](@_i)); & \%\,\text{Test for } \textsc{Meet}(I,\zeta,\xi) \\
& \text{M-}\textsc{enter}(\texttt{[}a\texttt{]}); \text{I-}\textsc{enter}(a_i)] & \%\,\text{Re-enter Ann's DB}
\end{array}
$$

As the annotations to the right side of (5.3.3) indicate, this dynamic formula corresponds to an algorithm that realizes the fetching procedure. The correspondence to such an algorithm is fairly coarse. To a large extent, this coarseness is due to the fact that the target procedure involves changes other than changes of context. For instance, the changes to Ann's database can't be formalized in DCIL; nor can the procedure of scanning all the records of Bob's database, which involves dynamic quantification. Also, we have left all temporal considerations out of the picture. By using a more powerful dynamic logic, it should be possible to have a more systematic and compositional relationship between algorithms and formulas of the logic.

The purpose of fetching records from Bob's database is to provide information about Bob's calendar in Ann's database. This purpose can be specified in DCIL by the following scheme, which says that what Ann knows (in her terms) about Bob's knowledge of his meetings is exactly what Bob knows (in his terms) about his meetings.

$$
\begin{array}{ll}
(5.3.4) & {}^{\vee}ist(a_i, \texttt{[}a\texttt{]}, {}^{\sqcap}ist(b_i, \texttt{[}b\texttt{]}, {}^{\sqcap\wedge}\textsc{Meet}(I,\zeta,\xi))) \\
& \quad \leftrightarrow {}^{\vee}ist(b_i, \texttt{[}b\texttt{]}, {}^{\sqcap\wedge}\textsc{Meet}(I,\zeta,\xi))
\end{array}
$$

Theorem 5.0. (5.3.4) follows in DCIL from (5.3.3); that is, every model of DCIL that satisfies (5.3.3) also satisfies every instance of (5.3.4), where ζ and ξ are constants in L_e^*.

*Proof sketch.*We assume that the interpretations of the constants ζ, ξ, and Meet—like most expressions of DCIL[6]—do not depend on the modal component of context. That is,
$[\![\zeta]\!]_{\mathcal{M},\langle \mathrm{M,i}\rangle,\langle \mathrm{M,i}\rangle,w} = [\![\zeta]\!]_{\mathcal{M},\langle \mathrm{M',i}\rangle,\langle \mathrm{M',i}\rangle,w}$, and similarly for ξ and Meet. We also assume that these expressions are static. Furthermore, $D_i = \{\mathrm{a,b,c}\}$ and the interpretations of I, b_i, $\texttt{[}b\texttt{]}$, are static and are fixed as follows, where $\mathrm{K_b}$ is a subset of D_w:

[6] Cases like $@_M$ are an exception.

$[\![I]\!]_{\mathcal{M},\langle \mathrm{M,i}\rangle,\langle \mathrm{M,i}\rangle,\mathrm{w}} = \mathrm{i}$, for $\mathrm{i} \in \{\mathrm{a,b,c}\}$.
$[\![\mathrm{b}_i]\!]_{\mathcal{M},\langle \mathrm{M,i}\rangle,\langle \mathrm{M,i}\rangle,\mathrm{w}} = \mathrm{b}$.
For $\mathrm{p} \in D_{\langle w,t\rangle}$, $[\![\mathtt{[}\,b\,\mathtt{]}]\!]_{\mathcal{M},\langle \mathrm{M,i}\rangle,\langle \mathrm{M,i}\rangle,\mathrm{w}}(\mathrm{p})$ is the constant function from D_w to $\top$ if for all w', $\mathrm{w}' \in \mathrm{K_b}$ if $\mathrm{p(w')} = \top$; and otherwise is the constant function from D_w to $\bot$.

It follows that ${}^{\vee}\mathit{ist}(b_i, \mathtt{[}\,b\,\mathtt{]}, {}^{\sqcap\wedge}\textsc{Meet}(I,\zeta,\xi))$ is static and context-independent;

$[\![{}^{\vee}\mathit{ist}(b_i, \mathtt{[}\,b\,\mathtt{]}, {}^{\sqcap\wedge}\textsc{Meet}(I,\zeta,\xi))]\!]_{\mathcal{M},\mathrm{c,c,w}} =$
$[\![{}^{\vee}\mathit{ist}(b_i, \mathtt{[}\,b\,\mathtt{]}, {}^{\sqcap\wedge}\textsc{Meet}(I,\zeta,\xi))]\!]_{\mathcal{M},\mathrm{c',c',w}}$

for all $\mathrm{c,c'}$. By similar reasoning, ${}^{\vee}\mathit{ist}(a_i, \mathtt{[}\,a\,\mathtt{]}, {}^{\sqcap}\mathit{ist}(b_i, \mathtt{[}\,b\,\mathtt{]}, {}^{\sqcap\wedge}\textsc{Meet}(I,\zeta,\xi)))$ is also static and context-independent. Now, suppose that $\mathcal{M}$ satisfies (5.3.3) at $\mathrm{c,c',w}$. Since (5.3.3) is static, $\mathrm{c = c'}$. Because (5.3.3) is context-independent, $\mathcal{M}$ satisfies (5.3.3) at $\langle \mathrm{M_a,a}\rangle, \langle \mathrm{M_a,a}\rangle, \mathrm{w}$, where $\mathrm{M_a}$ is the denotation in $\mathcal{M}$ of the modality $\mathtt{[}\,a\,\mathtt{]}$. Further, suppose that $\mathcal{M}$ satisfies ${}^{\vee}\mathit{ist}(a_i, \mathtt{[}\,a\,\mathtt{]}, {}^{\sqcap}\mathit{ist}(b_i, \mathtt{[}\,b\,\mathtt{]}, {}^{\sqcap\wedge}\textsc{Meet}(I,\zeta,\xi)))$ at $\mathrm{c,c,w}$. By context-independence of this formula, $\mathcal{M}$ satisfies ${}^{\vee}\mathit{ist}(a_i, \mathtt{[}\,a\,\mathtt{]}, {}^{\sqcap}\mathit{ist}(b_i, \mathtt{[}\,b\,\mathtt{]}, {}^{\sqcap\wedge}\textsc{Meet}(I,\zeta,\xi)))$ at $\langle \mathrm{M_a,a}\rangle, \langle \mathrm{M_a,a}\rangle, \mathrm{w}$. Therefore $\mathcal{M}$ satisfies the right side of (5.3.3) at $\langle \mathrm{M_a,a}\rangle, \langle \mathrm{M_a,a}\rangle, \mathrm{w}$. But this means that $\mathcal{M}$ satisfies ${}^{\vee}@_M({}^{\sqcap\wedge}\textsc{Meet}(I,\zeta,\xi)$ at $\langle \mathrm{M_b,b}\rangle, \langle \mathrm{M_b,b}\rangle, \mathrm{w}$. By context-independence, $\mathcal{M}$ must satisfy ${}^{\vee}\mathit{ist}(b_i, \mathtt{[}\,b\,\mathtt{]}, {}^{\sqcap\wedge}\textsc{Meet}(I,\zeta,\xi))$ at $\mathrm{c,c,w}$.

The converse is proved in similar fashion.

6 The Exit Operation and Stack-Structured Memory

The space limitations for this paper don't allow for an extended presentation of how to treat the operation of exiting a context. This section is merely a brief sketch of the topic.

Where κ and κ' are nonempty sequences of contexts and $\phi \in L_t^*$, $[\![\phi]\!]_{\mathcal{M},\kappa,\kappa',\mathrm{w}}$ represents a transition from a history in which the sequence κ is remembered to one in which κ' is remembered. We call an evaluation of an expression that uses context histories in this fashion a *historical* evaluation.[7]

In the following definition, $\kappa\frown c$ is the sequence resulting from appending c to κ.

Definition 6.8. *Locality.*

An expression ζ is *local* with respect to a historical interpretation iff $[\![\zeta]\!]_{\mathcal{M},\kappa\frown c,\kappa'\frown c',\mathrm{w}} = [\![\zeta]\!]_{\mathcal{M},\langle c\rangle,\langle c'\rangle,\mathrm{w}}$.

An expression is local, then, in case its interpretation involves only the current context. Nonlocal expressions will be exceptional in historical DCIL.

[7] A sequence κ represents only the part of the history of contexts traversed in an inquiry that is remembered. Some things may be forgotten; see the rule for exiting for an example.

The clauses defining historical interpretations for the static part of DCIL are straightforward adaptations of the clauses from Section 3.2. For instance, the clause for functional application goes as follows.

(6.2) Where $\zeta \in L^*_{\langle\sigma,\tau\rangle}$ and $\xi \in L^*_\sigma$, $[\![\zeta(\xi)]\!]_{\mathcal{M},\kappa,\kappa',\mathrm{w}} = [\![\zeta]\!]_{\mathcal{M},\kappa,\kappa',\mathrm{w}}([\![(\xi)]\!]_{\mathcal{M},\kappa,\kappa',\mathrm{w}})$.

Dynamic operations are more complicated, but still straightforward; this is illustrated by the clause for dynamic 'and'.

$[\![;(\phi)(\psi)]\!]_{\mathcal{M},\kappa,\kappa',\mathrm{w}} = \top$ iff there is a κ'' such that $[\![\phi]\!]_{\mathcal{M},k,k'',\mathrm{w}}$ and $[\![\psi]\!]_{\mathcal{M},k'',k',\mathrm{w}}$.

I'll replace $@_i$ and $@_M$ by expressions $\textsc{Current}_i$ of type $\langle i,t\rangle$ and $\textsc{Current}_M$ of type $\langle \tau_M, t\rangle$. $\textsc{Current}_i$ holds of an index i iff i is the index of the context currently being visited; similarly for $\textsc{Current}_M$.

If $\kappa = \kappa'$, $\kappa = \kappa_1{}^\frown\mathrm{c}$, and ζ is a constant of type τ_M, then
$[\![\textsc{Current}_M(\zeta)]\!]_{\mathcal{M},\kappa,\kappa',\mathrm{w}} = \top$ iff $\mathrm{c}_M = [\![\zeta]\!]_{\mathcal{M},\kappa,\kappa',\mathrm{w}}$;
otherwise, $[\![\textsc{Current}_M(\zeta)]\!]_{\mathcal{M},\kappa,\kappa',\mathrm{w}} = \bot$.
If $\kappa = \kappa'$, $\kappa = \kappa_1{}^\frown\mathrm{c}$, and ζ is a constant of type τ_i, then
$[\![\textsc{Current}_i(\zeta)]\!]_{\mathcal{M},\kappa,\kappa',\mathrm{w}} = \top$ iff $\mathrm{c}_i = [\![\zeta]\!]_{\mathcal{M},\kappa,\kappa',\mathrm{w}}$;
otherwise, $[\![\textsc{Current}_i(\zeta)]\!]_{\mathcal{M},\kappa,\kappa',\mathrm{w}} = \bot$.

The clause for EXIT simply pops the current context off of the history, returning the value $\bot$ if there is no such context.

Where $\zeta \in L^*\tau_M$ and $\mathrm{c}_i = \mathrm{i}$, $[\![\textsc{exit}(\zeta)]\!]_{\mathcal{M},\kappa,\kappa',\mathrm{w}} = \top$ iff $\kappa = \kappa_1{}^\frown c_1$ for some c_1 (where κ_1 is non empty) and $\kappa' = \kappa_1$.

The following variation on (5.3.3) uses historical DCIL to characterize a procedure that uses the exit operation to update Ann's database with information from Bob's database.

(6.3) $[\textsc{Current}_M(\texttt{[}a\texttt{]}) \wedge \textsc{Current}_i(a_i)] \rightarrow$ % Start in Ann's DB
$[{}^\vee ist(a_i, \texttt{[}a\texttt{]}, {}^\sqcap ist(b_i, \texttt{[}b\texttt{]}, {}^{\sqcap\wedge}\textsc{Meet}(I,\zeta,\xi))) \leftrightarrow$
$\textsc{M-enter}(\texttt{[}b\texttt{]}); \textsc{I-enter}(b_i);$ % Enter Bob's DB
${}^\vee @_M([{}^{\sqcap\wedge}\textsc{Meet}(I,\zeta,\xi)](@_i));$ % Test for $\textsc{Meet}(I,\zeta,\xi)$
$\textsc{exit}; \textsc{exit}]$ % Re-enter Ann's DB

In the last step, we have to exit twice because Bob's database was entered in two operations.

7 Conclusion

The application described in Section 5 of DCIL is too simple to do more than indicate a possible use of the logic in specifying tasks involving reasoning about context and proving correct the algorithms that do this reasoning. It remains to be seen if the logic is actually useful in formalizing domains that are complex enough to be realistic and useful. The success of this program depends on the development of generally applicable techniques for formalizing domains in DCIL. Also, as was mentioned in the discussion of the correspondence between (5.3.3) and the implemented algorithm, it might be useful to extend the dynamic logic so that the correspondence between the dynamic constructions and actual algorithms for reasoning about context is more systematic.

At several places, I indicated ways in with this approach to contextual reasoning could be extended and developed. For reasons that became evident when we introduced the null value in Section 3.2, the logic needs to be made partial. And of course, a nonmonotonic version of the logic remains to be developed. So there still remains much work to do on this approach to contextual logic, both to ensure a better relation to applications and a better match to general requirements on a fully adequate logic of context.

Acknowledgements. I owe thanks to the two referees who commented on this paper for Context'03.

References

1. Thomason, R.H.: Contextual intensional logic: Type-theoretic and dynamic considerations. In Bouquet, P., Serafini, L., eds.: Perspectives on context. CSLI Publications, Stanford, California (2003)
2. Lewis, D.K.: Scorekeeping in a language game. Journal of Philosophical Logic **8** (1979) 339–359
3. van Benthem, J.: Exploring Logical Dynamics. CSLI Publications, Stanford, California (1996)
4. Groenendijk, J., Stokhof, M.: Dynamic Montague grammar. Technical Report LP–90–02, Institute for Language, Logic and Information, University of Amsterdam, Faculty of Mathematics and Computer Science, Roeterssraat 15, 1018WB Amsterdam, Holland (1990)
5. Montague, R.: Pragmatics and intensional logic. Synthese **22** (1970) 68–94 Reprinted in *Formal Philosophy*, by Richard Montague, Yale University Press, New Haven, CT, 1974, pp. 119–147.
6. Church, A.: A formulation of the simple theory of types. Journal of Symbolic Logic **5** (1940) 56–68
7. Henkin, L.: Completeness in the theory of types. Journal of Symbolic logic **15** (1950) 81–91
8. McCarthy, J., Buvač, S.: Formalizing context (expanded notes). In Aliseda, A., van Glabbeek, R., Westerståhl, D., eds.: Computing Natural Language. CSLI Publications, Stanford, California (1998) 13–50
9. Harel, D.: Dynamic logic. In Gabbay, D., Guenther, F., eds.: Handbook of Philosophical Logic, Volume II: Extensions of Classical Logic. Volume 2. D. Reidel Publishing Co., Dordrecht (1984) 497–604
10. Gallin, D.: Intensional and Higher-Order Logic. North-Holland Publishing Company, Amsterdam (1975)