

Lucid and Intensional Logic

David Israel
AI Laboratory
SRI International
Menlo Park, CA 94025

1 Introduction

Lucid is often described as an *intensional* and/or an *indexical* language. In this note, we attempt to explain these categories and to place **Lucid** more clearly within them.

We begin by contrasting *extensional* and *intensional* constructs. A language is said to be intensional if it contains at least one intensional construct. We shall use, as a focus for this first contrast, a comparison between classical propositional logic (PL) and a simple propositional temporal logic (PTL). We then generalize from this case to characterize *functional* or *compositional* semantic accounts. Next we turn to the distinction between indexical and nonindexical languages. Finally, after all these preliminaries, we turn to a description of **Lucid** in terms of the categories we have delineated.

2 Classical Propositional Logic and Temporal Logic

We begin with a (denumerable) set \mathcal{P} of atomic sentence letters and close under the Boolean connectives $\neg, \&$. Let \mathcal{L}_{PL} be the resulting language, which for present purposes we can identify with set of its well-formed sentences. The elements of \mathcal{L}_{PL} are thought of as uninterpreted, but interpretable. Interpretations for arbitrary sentences are built up by way of models. A model M for \mathcal{L}_{PL} is an assignment of T (true) or F (false) to the members of \mathcal{P} . Such models are then extended to interpretations for the whole language in the usual way, by an inductive definition whose cases are based on the principal connective of the sentence. The clauses of this definition are as follows.

- For $p \in \mathcal{P}$, $I(p) = M(p)$. We also write $M \models p$ if $M(p) = T$ and $M \not\models p$ if $M(p) = F$.
- For $\Phi = \phi \& \psi$, $I(\Phi) = T$ iff $I(\phi) = I(\psi) = T$. ($M \models \Phi$ iff $M \models \phi$ and $M \models \psi$.)
- For $\Phi = \neg\phi$, $I(\Phi) = T$ iff $I(\phi) = F$. ($M \models \Phi$ iff $M \not\models \phi$.)

Thus the truth-value of a sentence on an interpretation, that is relative to a model, is a function of the truth-values of its components (its subsentences) on that interpretation.

One *can* think of the truth assignments as representing different ways the world might be; but within (classical) propositional logic, there is no construction *in* the language to capture this dependence of truth on a dimension of variation. The semantics of sentences is ‘local’; the truth-value on a model of a complex sentence is determined by—is a function of—the truth-values *on that same model* of its component sentences; hence the term *truth-functional* logic. Conversely, various kinds of global, or at least nonlocal, dependence are precisely the focus of intensional logics.

Consider an informal example: the sentences “Philadelphia is the capital of the United States” and “Palo Alto is the capital of the California” are both false at the present time. But compare: “It once was the case that Philadelphia was the capital of the United States” (or more simply: “Philadelphia was once the capital of the U.S.”) and “Palo Alto once was the capital of California.” The first is true; the second, false. Thus, the truth-values of the compound sentences are not a function of the truth-values of their components. In particular, the truth-value of a temporal sentence at a time t may depend on the truth-values of its component sentences at other times earlier than or later than t .

To characterize \mathcal{L}_{PTL} , we begin with \mathcal{L}_{PL} and close it under two temporal (tense) operators, F and P . Where Φ is a sentence of \mathcal{L}_{TL} , $F\Phi$ is understood as saying that it will be the case that Φ ; $P\Phi$ as saying that it once was the case that Φ . There are many different ways to capture this intuition of the time-relative dependence of such tensed propositions. Here we choose to do it in an especially simple way.

Model structures for \mathcal{L}_{PTL} are pairs $\mathcal{M} = \langle \langle T, < \rangle, v \rangle$. The first element of \mathcal{M} is a (linear) temporal structure, consisting of a set T of moments (times), ordered by $<$, the relation of one time being earlier than another. We have stipulated that $<$ be linear; a natural model to have in mind is that of the integers, with $<$ the relation of one integer being less than another. In line with our earlier treatment of \mathcal{L}_{PL} , we shall think of v as associating with each $p \in \mathcal{P}$ a set of ordinary propositional models to p , one for each moment of time. On this understanding, v is a function from $\mathcal{P} \times T$ to $\{T, F\}$. We can also think of v as a propositional assignment, associating with each $p \in \mathcal{P}$ the set $v(p)$ of times at which it is true. On this understanding, v is a function from \mathcal{P} to $Pow(T)$, the set of all subsets of T . We could even present v as function from times to propositional models, that is, from T to $Pow(\mathcal{P})$; v thus assigns to a time t those atomic sentences true at t . It should be clear that these three ways of conceiving of v are equivalent. In what follows, we shall follow the first convention.

Given that the truth-values of the sentences of our temporal language are to be relative to a time, that is, they are to take on different truth-values at different times, it is clear that we can not define the crucial notion of the truth-value (interpretation) of a sentence, relative to a propositional model, solely in terms of model structures. We must rather take as our models or *points of evaluation*, pairs $\langle \mathcal{M}, t \rangle$ of model structures and times $t \in T$. We shall notate the interpretation (truth-value) of a formula Φ at such a pair as $I_t^{\mathcal{M}}(\Phi)$. Thus the base clause and the clauses for the

temporal operators are as follows:

- For $p \in \mathcal{P}$, $I_t^M(p) = v(p, t)$. (Alternatively, $M, t \models p$ iff $t \in v(p)$.)
- $I_t^M(F(\Phi)) = \mathbf{T}$ iff $\exists t'(t < t' \ \& \ I_{t'}^M(\Phi) = \mathbf{T})$. (Alternatively, $M, t \models F\Phi$ iff $\exists t'(t < t' \ \& \ M, t' \models \Phi)$.)
- $I_t^M(P(\Phi)) = \mathbf{T}$ iff $\exists t'(t' < t \ \& \ I_{t'}^M(\Phi) = \mathbf{T})$. (Alternatively, $M, t \models P\Phi$ iff $\exists t'(t' < t \ \& \ M, t' \models \Phi)$.)

Given this account, we can model the situation sketched above as follows. Let p be associated with the sentence “Philadelphia is the capital of the U.S.” and q with “Palo Alto is the capital of California.” Let the elements of T be years (we shall ignore the Big Bang theory and imagine the universe is infinite in temporal duration), and let $t \in T$ be 1991.¹ On the intended model $v(p, t) = v(q, t) = \mathbf{F}$; that is, both p and q are false at t . But Pp is true and Pq false, as there is a $t' < t$, such that $v(p, t') = \mathbf{T}$, but there is no such t' for q .

It is important to note that in the language \mathcal{L}_{PTL} there is no explicit reference to times, to members of T at all. The semantics of the temporal operators are given in terms of quantification (existential quantification, in particular) over T , but there are no terms for times in the language. Indeed, there are no individual terms at all in the language, and hence there can be no explicit quantification over times, either. This is typical of *intensional* (but nonindexical) languages. Their semantic account will involve positing some domain or domains relative to which the truth-values of sentences or the references of terms (for which see below) may vary and some operators or constructions understood in terms of such variation—in our case, F and P —but no explicit reference to or quantification over such domains. In other words, implicit relativization to a domain of variation is the hallmark of intensional languages.

This intensional treatment of the phenomenon of time-relative truth-values, in which there is no explicit reference to times at all, should be contrasted with a treatment that involves explicit quantification over times. In this second kind of treatment, we might have a(n at least) two sorted first-order language, with a subcollection of individual variables V_t , perhaps individual constants of sort T , and a binary relation symbol E whose arity is $\langle t, t \rangle$. The class of structures \mathcal{M} associated with this language would include as a subdomain a set T of times, and a binary relation $E^{\mathcal{M}}$, understood as the ordering relation $< \subseteq T \times T$. The analogues of sentence letters would not be treated as sentence letters but as monadic predicates on times. If one were now to give an account of the first-order language just sketched, there would be no operators such as F or P whose treatment required us to posit models more complex than those of standard many-sorted first-order logic. That is, there would be no relativity of truth to some nonexplicit dimension of variation—other than the

¹We have said nothing until now about the elements of T ; the granularity of T (the size of the elements), in particular, is something to be determined by some theoretical or practical purpose for which the logic is being applied. Here we suppose that the times are year-long intervals.

quite standard one of relativity to assignments to variables, which is in any event irrelevant in the case of sentences (formulas with no free variables).

3 Compositionality and Transparency; Extensions and Intentions

We noted above that in propositional logic, the semantic value of a complex sentence in a model, its truth-value in that model, was a function of the semantic values of its components on that model, but that this was not so in propositional temporal logic *so long as we take truth-values as the semantic values of sentences*. But the lesson of even this simple example is that we cannot always take the truth-value of a sentence on a truth-value assignment, a model, to be determined locally by (i) the syntactic structure of the sentence and (ii) the truth-values of its component sentences—if any—relative to that model. Rather, we must look to the whole of some dimension of variation to determine the truth-value of a temporally modified sentence at some point on that dimension: the truth-value of a tensed sentence at a time depends the truth-values of its constituent sentences at other times, and some of these constituent sentences might themselves be tensed, of course. So even where we fix a model structure, the semantic value of interest is not so much the truth-value of a sentence at a time, but the pattern of truth-values it takes on as we vary the relevant parameter, in our case the time. On this way of looking at things, the central semantic value of a sentence Φ of \mathcal{L}_{PTL} in a model structure is the set of times at which Φ is true, or equivalently, that function f^Φ from times to $\{T, F\}$ such that $f^\Phi(t) = T$ iff Φ is true at t . Let us call this—the set or its associated characteristic function—the proposition expressed by Φ , again, relative to a model structure M . (In what follows we shall take the functional perspective.)

For \mathcal{L}_{PTL} let us call the truth-value of Φ at t , in M , the *extension* of Φ at M, t and notate it as follows: $Ext_{M,t}(\Phi)$, and let us call the proposition expressed by Φ , relative to M , its *intension* and notate it as follows: $Int_M(\Phi)$. We expect, of course, that $Int_M(\Phi)(t) = Ext_{M,t}(\Phi)$.

To say that F and P are non truth-functional, that is, nonextensional contexts, is just to say that the following condition of extensionality does *not* hold. Where Φ, Ψ are sentences,

$$Ext_{M,t}(\Phi) = Ext_{M,t}(\Psi) \Rightarrow Ext_{M,t}(P(\Phi)) = Ext_{M,t}(P(\Psi))$$

This principle does hold for negation. The truth-value of $\neg\Phi$ at a time t is a function of the truth-value (the extension) of Φ at t .

It should also be clear though that F and P introduce intensional contexts, that is, that one can substitute sentences that are co-intensional (intensionally equivalent) and preserve truth-value. Thus, the following condition is satisfied: For $t \in T$,

$$Int_M(\Phi) = Int_M(\Psi) \Rightarrow Ext_{M,t}(P(\Phi)) = Ext_{M,t}(P(\Psi))$$

That is, the truth-value of a tensed sentence at a time (relative to a model structure) is a function of the intension of that sentence (relative to that model structure).

Let us say a context $C[...]$ is *referentially transparent* when the extension of the result of any grammatically correct substitution into that context is determined solely by the extension of the expression so substituted, and hence is preserved under substitution by extensionally equivalent expressions. A context is *referentially opaque* when it is not transparent. $\neg[...]$ is referentially transparent. The grammatically correct substituends are sentences, and $Ext_{M,t}(\neg\Phi)$ is a function solely of $Ext_{M,t}(\Phi)$. Hence where $Ext_{M,t}(\Phi) = Ext_{M,t}(\Psi)$, $Ext_{M,t}(\neg\Phi) = Ext_{M,t}(\neg\Psi)$. The context $F[...]$ is referentially opaque. Though the context has the same class of substituends as \neg , there is *no* function \mathcal{G} such that $Ext_{M,t}(F\Phi) = \mathcal{G}(Ext_{M,t}(\Phi))$. Hence it is false that if $Ext_{M,t}(\Phi) = Ext_{M,t}(\Psi)$, then $Ext_{M,t}(F\Phi) = Ext_{M,t}(F\Psi)$.

We now present a general framework within which we can express the *Principle of Compositionality*, the requirement or desiderata that the semantic value of a complex expression be functionally determined by the values of its constituent parts.

A language can be characterized in terms of the basic categories Cat_i to which its atomic expressions belong and a set of syntactic operations taking expressions of given categories into an expression of some category. Given such a characterization, we have rules of the following form. Where \mathcal{F} is a syntactic operation:

If α_1 is an expression of category Cat_1 , ..., and α_n is an expression of Cat_n , then $\mathcal{F}(\alpha_1, \dots, \alpha_n)$ is of category Cat_{n+1} .

The semantics of the language will include assignments of atomic expressions to semantic types (or domains) \mathcal{D}_i corresponding to the assignment of atomic expressions to categories. Corresponding to a syntactic rule such as that above, we should expect to have the following form for a rule of meaning, where we continue to notate the basic semantic value of an expression α by $I(\alpha)$.

If α_1 is an expression of category Cat_1 , ..., and α_n is an expression of Cat_n , then $I(\mathcal{F}(\alpha_1, \dots, \alpha_n)) = \mathcal{G}(I(\alpha_1), \dots, I(\alpha_n))$, of semantic type \mathcal{D}_{n+1} .

The precise mathematical content of the *Principle of Compositionality* is that there be a *homomorphism* from the syntactic algebra of the language to the semantic algebra. Thus, more generally and abstractly, we have the following requirement:

For every n-ary syntactic operator $\mathcal{F} : Cat_1 \times \dots \times Cat_n \rightarrow Cat_{n+1}$, there exists a function $\mathcal{G} : \mathcal{D}_1 \times \dots \times \mathcal{D}_n \rightarrow \mathcal{D}_{n+1}$ such that, for every complex expression $\mathcal{F}(\alpha_1, \dots, \alpha_n) \in Cat_{n+1}$, where $\alpha_i \in Cat_i$,

$$I(\mathcal{F}(\alpha_1, \dots, \alpha_n)) = \mathcal{G}(I(\alpha_1), \dots, I(\alpha_n)) \in \mathcal{D}_{n+1}$$

Thus it is that compositional semantic accounts might also be said to be functional or algebraic.

If we apply this scheme to the case of propositional temporal logic, the syntax might be characterized as consisting of three basic categories of expressions: Sentences (S), Unary Sentential Connectives (USC): F , P , \neg , and Binary Sentential Connectives (BSC): $\&$. One would then posit various syntactic operations making sentences out of sentences, for example, prefixing F to a sentence yields a sentence. If we stipulated that the semantic type of sentences was **Bool**, i.e., $\{T, F\}$ and the semantic type of all UCS's was **Bool** \rightarrow **Bool**, we would not successfully honor the *Principle of Compositionality*, and we would get the semantics of our tensed sentences, and of the temporal operators F and P , wrong. That was why, in our treatment just above, we moved to the intension of a sentence as its central semantic value.

To honor the Principle of Compositionality and to get the semantics of F and P right, we must change our account of the semantic types of our (basic) categories of expressions. We could do this in more than one way. In particular, we may decide to change our description of the syntax, allowing now for two different categories of unary sentential connective: one for the extensional, truth-functional negation connective \neg , and a separate category for the intensional, nontruth-functional temporal connectives, F and P . We shall not do this; we shall instead follow standard practice, the one suggested earlier in our treatment of F and P , and make wholesale changes in the semantic types.

Where before the semantic type or semantic domain of sentences was **Bool**, it is now **Bool** ^{T} , or $T \rightarrow \text{Bool}$, the domain of functions from times into truth-values. Let's call this domain **Prop**, for Propositions. Propositions, then, are the *intensions* of sentences; truth-values, which they have only relative to a time, their *extensions*. In general, the intensions of expressions will be functions from the domain (parameter) of variation to the domain of the extensions of those expressions. The binary connectives are then of type **Prop** \times **Prop** \rightarrow **Prop**. Finally, all unary connectives are of type: **Prop** \rightarrow **Prop**. Now once again the principle of compositionality is honored. Let us see how this works.

Where Φ is a sentence, let $f^\Phi \in \text{Prop}$ be $I(\Phi)$, the intension of Φ . Then, suppressing mention of M , where $t \in T$,

- $I(\neg(\Phi))(t) = I(\neg)(f^\Phi)(t) = (f^\neg(f^\Phi))(t)$, where $f^\neg(f)(t) = T$ if $f(t) = F$ and $f^\neg(f)(t) = F$ if $f(t) = T$.
- $I(P(\Phi))(t) = I(P)(f^\Phi)(t) = f^P(f^\Phi)(t)$, where $f^P(f)(t) = T$ if for some $t' < t$, $f(t') = T$ and $f^P(f)(t) = F$ if, for all $t' < t$, $f(t') = F$.
- The meaning of F is similar in form to that of P .
- $I(\Phi \& \Psi)(t) = I(\&)(f^\Phi, f^\Psi)(t) = f^\&(f^\Phi, f^\Psi)(t)$, where $f^\&(f_1, f_2)(t) = T$ if $f_1(t) = f_2(t) = T$ and $f^\&(t) = F$ if either $f_1(t) = F$ or $f_2(t) = F$.

This rather complicated treatment satisfies the Principle of Compositionality, and gets the semantics of F and P right.

4 Indexicality

Let us return to temporal logic. According to the semantics we presented, the truth-value of an atomic sentence at a time (in a model structure) was determined locally, that is simply by its value at that time, whereas the truth-value of a tensed sentence (a sentence governed by a temporal operator) at a time was determined by a condition that involved quantification over the entire domain of times. We allowed ourselves no way to move from one fixed time t to some other time t' specified in terms of its relation to t . It may seem that this is easily done. We add two new unary temporal operators, O and O^- . The intuitive understanding of $O\Phi$ is “at the next time, (it will be the case that) Φ ” and of $O^-\Phi$, “at the previous time”, (it was the case that) Φ . In the alternative format in terms of \models , the clauses are quite simple:

- $M, t \models O(\Phi)$ iff $M, t + 1 \models \Phi$
- $M, t \models O^-(\Phi)$ iff $M, t - 1 \models \Phi$

Each of O, O^- is an iterable operator, like \neg or F , and each is also freely intermixable with any other unary operator. Nothing new here, yet. But how do our two new operators, or their natural language counterparts, compare e.g., with “tomorrow” or “yesterday”? And for that matter, how does the semantics of a simple tenseless sentence Φ compare with “Now, Φ ”?

Compare, for instance: “Palo Alto is the capital of California” and “Now, Palo Alto is the capital of California” (or: “Palo Alto is the capital of California now.”) It certainly seems that they are true or false together. Of course, the same was true of our two sentences about Palo Alto and Philadelphia, but here something stronger seems to be the case: it seems to be a truth of logic that $\phi \leftrightarrow \text{Now}, \phi$. Now compare:

- The psychic predicted that there would be an earthquake.
- The psychic predicted that there would be an earthquake *now*.

These seem quite different. On the first, the psychic would have gotten it right just in case at least once in the past he predicted that there would be an earthquake and there later was an earthquake. On the second, for the psychic to be right he must in the past have made a prediction about a specific time, which time is fixed by the context in which the sentence is uttered. That is, fix the time of utterance of the second sentence at t . Then for the psychic to have predicted correctly, he must have said, at some time before t , that there would be an earthquake at t . Notice, that if the second sentence is uttered at different times, but about the same psychic, it records different predictions, different commitments, by the psychic. Not so, at

least not necessarily so, with the first sentence. And these same contrasts would hold if we substituted “yesterday” or “tomorrow” for “now”. So, what is the difference between, e.g., “yesterday” and “at the previous time” (O^-)?

The use of “yesterday” in a sentence S refers to a particular time (day) relative to a distinguished time, namely the time (day) of the utterance of S , namely the day before the day of the utterance. Thus, on different days, the sentence, “Yesterday Φ ” expresses different propositions. “At the previous time (day)” refers, given an *arbitrary* time (day) t to the previous time (day), that is, to $t - 1$. This sentence, always expresses the same time-relative proposition.

To capture the intended interpretation of “yesterday”, we need to tie the point of evaluation of the embedded sentence Φ to a distinguished time, intuitively the referent of “now” or “today”, in a way that is invariant with respect to the point of evaluation of the containing sentence. Another way to put this is that we need to consider the proposition (the function from times to truth-values) expressed by “Yesterday, Φ relative to the distinguished index determined by the time of utterance, and only then to evaluate that proposition at arbitrary times. This suggests that to handle such phenomena, times will play two different roles: they will be points of evaluation, or arguments to the intensions of sentences, thus yielding truth-values and they must also be points of determination, determining what proposition, what intension, is associated with a given sentence. Such expressions are called *indexicals*, and languages that contain them, *indexical*.

There are various ways to capture this double role. Syntactically, we simply add a unary sentence connective N ; $N(\Phi)$ is to be read as “Now, Φ .” Semantically, the picture is somewhat more complicated. One option is to posit a family of *pointed temporal structures*, $M_t, t \in T$. That is, for each t in T , we have a linear-time temporal model structure $\langle T, <, t, v \rangle$, within which t is the distinguished point of determination. Fixing a particular such structure M_t and continuing to treat v as a function from $\mathcal{PXT} \rightarrow \text{Bool}$, the base clause and the clause we add for N are as follows:

- For $p \in \mathcal{P}$, $I(p)(t') = v(p, t')$. (Alternatively, $M_t, t' \models p$ iff $v(p, t') = \text{T}$. Intuitively, where M is a standard, unpointed temporal structure and v the corresponding valuation: $M, t' \models p$.)
- $I(N(\Phi))(t') = I(\Phi)(t)$. (Alternatively, $M_t, t' \models N\Phi$ iff $M_t, t \models \Phi$.)

Notice the special role played by the distinguished point t . It may seem that this role will get in the way of achieving a desiderata: namely that $\Phi \leftrightarrow N(\Phi)$ should be logically valid. Indeed, to capture this, we must give distinguished elements a distinguished role also in the definition of validity. A formula Φ is valid in a pointed linear-time structure $\langle T, <, t \rangle$ iff, relative to all propositional assignments v , $M_t, t \models \Phi$. Φ is valid in the class of pointed linear-time structures iff valid in each structure in the class.

Now compare the clauses for $O^-\Phi$ and for $O^-N\Phi$, where this last expresses “Yesterday, Φ .” (We could, of course, introduce a primitive “yesterday” operator.)

Again suppressing reference in favor of reference to t .

- $I^t(O^-(\Phi))(t') = I^t(\Phi)(t' - 1)$. (Alternatively, $M_t, t' \models O^-(\Phi)$ iff $M_t, t' - 1 \models \Phi$.)
- $I^t(O^-N(\Phi))(t') = I^t(\Phi)(t - 1)$. (Alternatively, $M_t, t' \models O^-N(\Phi)$ iff $M_t, t - 1 \models \Phi$.)

Another way of treating the semantics of N is to go from temporal models in which v is a function $\in \mathcal{P} \times T \rightarrow \mathbf{Bool}$, to one in which we take seriously the idea of the point of determination, determining, given a time, a propositional intension, that is a function from times to truth values. This suggests taking v as function: $(\mathcal{P} \times T) \rightarrow (T \rightarrow \mathbf{Bool})$ or, in curried version: $\mathcal{P} \rightarrow (T \rightarrow (T \rightarrow \mathbf{Bool}))$. Here the first time is the point of determination, yielding a propositional intension, the second that point of evaluation, yielding a truth-value.

The base clause and the clause for N now look like this:

- For $p \in \mathcal{P}$, $I(p)(t)(t') = v(p)(t)(t')$. (Alternatively, and *uncurrying*: $M, (t, t') \models p$ iff $v(p)(t)(t') = \mathbf{T}$.)
- $I(N(\Phi))(t)(t') = I(\Phi)(t)(t)$. (Alternatively, $M, (t, t') \models N\Phi$ iff $M, (t, t) \models \Phi$.)

On this account, Φ is valid in a structure $\langle T, < \rangle$ iff $M, (t, t) \models \Phi$, for every $t \in T$ and every v . And a sentence is valid in a family of structures iff valid in each. Notice again, the special role played by the point of determination in the definition of validity.

The addition of indexicals to an intensional language such as \mathcal{L}_{PTL} affects the compositional semantics of the resulting language. In particular, given that we aim for a uniform semantic treatment of all expressions of a given category, and given our treatment of the unary sentence connective N , how should we treat the semantics of F or P . The second and more direct of our two treatments suggests that when we moved from \mathcal{L}_{PTL} to \mathcal{L}_{PTL+N} ‘simply’ by adding a new operator, retaining the old, we were committing a kind of pun. We can see this by looking at the clause for, e.g., F in \mathcal{L}_{PTL+N} :

- $I(F(\Phi))(t)(t') = I(F)(f^\Phi)(t)(t') = f^F(f^\Phi)(t)(t')$, where $f^F(f)(t)(t') = \mathbf{T}$ if for some $t'' > t'$, $f(t, t'') = \mathbf{T}$ and $f^P(f)(t)(t') = \mathbf{F}$ if, for all $t'' > t'$, $f(t, t'') = \mathbf{F}$. (Alternatively, $M, (t, t') \models F(\Phi)$ iff $\exists t''(t' < t'' \ \& \ M, (t, t'') \models \Phi)$.)

Thus, we continue to class all unary sentential connectives together, though their semantic type has changed, along with that of propositions. Propositions are no longer functions from times to truth-values; they are now functions from ordered pairs of times to truth-values. Thus USC’s can still be thought of as associated with functions from **Prop** to **Prop**, but now with a different structure to the domain of propositions. In any event, in this sense the logic is now *two-dimensional*, though here it is two copies of the same dimension.

Notice that “now” (N), does in a way introduce a new dimension of variation. The elements of that dimension are still members of T , but, on both of the above treatments, each such member gets to play two different roles. One role is as before: as a point of evaluation of the truth of formulas. The other, new role, is a distinguished point relative to which a point of evaluation is determined; of course the relation involved may be identity, as it is with “now” (N). In the first treatment, this role is realized by the device of distinguished elements of the domain (T); in the second more direct treatment, by having the relation in question, that between the distinguished index and an arbitrary index, play a role in the models themselves. This proposition-determining role only comes into play in the presence of constructs like N .

5 A State-based semantics for \mathcal{L}_{PTL}

Before turning to **Lucid**, we shall present an alternative semantic account of our original temporal language \mathcal{L}_{PTL} , one closer in motivating considerations to **Lucid**. This alternative is based on the notion of a computation sequence of states of a machine, and is thus more appropriate than the time-based semantics presented in §2. It is also slightly more complicated, and we thought it best to place the general discussion of intensionality and (indexicality) in the context of presenting the simpler account.

We begin in the same way, by positing an underlying (linear-time) temporal structure $\langle T, < \rangle$, isomorphic to the natural numbers, relative to their ordering by $<$. Indeed, we shall now use $\langle N, < \rangle$ directly, that is, we shall identify times by numbers, and the *earlier than relation* with $<$ on numbers. Further, we assume a universe of states S ; let computation sequences or *paths* be elements of S^ω , that is infinite sequences of states. We shall use p, q , etc., as path variables. Where $p = (p(0), p(1), \dots)$, we shall also write (p_0, p_1, \dots) . The kind of temporal structure, then, that entered into our earlier account of temporal logic will here be implicit. But something that was there implicit must now be made explicit, at least in the metalanguage. In our earlier treatment, we interpreted sentences as true relative to times. We introduced no notion of state. We now want to capture the notion of a sentence being true at (relative to) a certain stage (state) in a path. What should be our analogue of v , the propositional assignment in a linear-time model?

In order to capture the intuitions behind this new path-based version of temporal logic, we must allow for access to arbitrary times or stages of a computation sequence. The most natural way to accomplish this is to move to an account that holds that we now have two dimensions of variation. A given state can occur more than once in a path and can occur in more than one path, and a sentence (at least if governed by an intensional operator) can access any point along a path. Thus our points of evaluation or indices will be pairs (q, n) , $q \in S^\omega$, $n \in \omega$. The clause for $p \in \mathcal{P}$ is as follows: $I(p, \langle q, n \rangle) = v(p, q_n)$. In terms of \models , this comes to $M, (q, n) \models p$ iff $v(p, q_n) = T$. This suggests that $v \in \mathcal{P} \times (S^\omega, \omega) \rightarrow \mathbf{Bool}$.

We have suggested that our language is \mathcal{L}_{PTL} . We shall include the two operators,

O and O^- from our discussion of indexicality. The semantic domain or intension, of sentences, **Prop**, is now that of functions from ordered pairs of paths and numbers to truth-values; thus the semantic type of unary sentence connectives is now:

$$(\langle S^\omega X \omega \rangle \rightarrow \mathbf{Bool}) \rightarrow (\langle S^\omega X \omega \rangle \rightarrow \mathbf{Bool})$$

The interesting clauses of the inductive definition of truth in a model for the propositional fragment of the language are now as follows:

1. $I(F(\Phi))(p, n) = \mathbf{T}$ iff $\exists m \geq n \ \& \ I(\Phi)(p, m) = \mathbf{T}$. (Alternatively, $M, (p, n) \models F\Phi$ iff $\exists m \geq n (M, (p, m) \models \Phi)$.) The clause for P is similar.
2. $(O(\Phi))(p, n) = \mathbf{T}$ iff $I(\Phi)(p, n+1) = \mathbf{true}$. ($M, (p, n) \models O(\Phi)$ iff $M, (p, n+1) \models \Phi$.) The clause for O^- is similar, except that all sentences whose principal connective is O^- are false at $(p, 0)$ — p arbitrary.

Notice that though a sentence is true only relative to a path and a state on that path, none of the operators involve switching paths, though they do involve switching states.

6 Lucid

In [2], Ashcroft and Wadge present Lucid as both a programming language and “a formal system for proving properties of programs.” We do not follow them in this regard; we shall also make note of other deviations as we proceed. We choose instead to develop an intensional logic, a variant of linear time temporal logic, in which to express and prove properties of Lucid programs. A full description of the Lucid programming language can be found in [1].

6.1 Preliminaries

The values of Lucid variables are ω -sequences of values from some underlying domain; where S is a set, we shall notate the set of ω -sequences over S as ${}^\omega S$. We shall follow [2] in stipulating that these underlying structures be cpo’s. We begin with a brief presentation of the relevant structural notions.

A *many-sorted signature* Σ consists of a set $I = \text{Sort}(\Sigma)$ of *sort indices*, two disjoint sets $\text{Op}(\Sigma)$ and $\text{Const}(\Sigma)$ of operation and constant symbols, with associated *arities* and for each $i \in I$ the binary relation symbol $=_i$. For each sort i there will be a constant symbol \perp_i . The *arity* of a constant symbol is a sort index; that of an operation symbol is a pair (\vec{i}, j) where $\vec{i} \in \vec{I} =$ the set of nonempty finite sequences of elements of I . The arity of $=_i$ is $(i, i) \in \vec{I}$.

A Σ -*algebra* \mathcal{A} is an assignment to each $i \in I$ of a set A_i , a function $f^{\mathcal{A}}: A_{\vec{i}} \rightarrow A_j$ for each $f \in \text{Op}(\Sigma)$ of arity (\vec{i}, j) , and an element $c^{\mathcal{A}} \in A_j$ for each $c \in \text{Const}(\Sigma)$ of

arity j . In particular, for each $i \in I$, \perp_i is assigned the least element in A_i . Moreover, each $=_i$ is assigned the identity on A_i . (That is, $\forall x, y \in A_i, x =_i y \Leftrightarrow x = y$.) We shall assume that this relation is decidable, and that the associated characteristic functions eq_i are among the basic operations of \mathcal{A} . Given this, we shall suppress mention of eq_i , in favor of $=_i$.

All signatures will contain Σ_{KB} . This is a single sorted signature (that is, $I = \{1\}$), with two basic operation symbols **neg** and **conj**, the first of arity $\langle(1), 1\rangle$, the second of arity $\langle(1, 1), 1\rangle$, and three constants, tt, ff, \perp . The associated algebra (structure), then is $KB = \langle KB, =, \neg, \wedge, tt, ff, \perp \rangle$, where $\neg = \text{neg}^{KB}$, $\wedge = \text{conj}^{KB}$, etc.

Let us explain KB . We follow Kleene's strong 3-valued logic, so \wedge and \neg have the following matrices:

p	$\neg p$	q	tt	\perp	ff
tt	ff	tt	tt	\perp	ff
\perp	\perp	p	\perp	\perp	ff
ff	tt	ff	ff	ff	ff

We define $p \vee q$ as $\neg(\neg p \wedge \neg q)$ and $p \supset q$ as $\neg p \vee q$, $p \equiv q$ as $p \supset q \wedge q \supset p$. Finally all signatures will be equipped with a conditional operator **if-then-else**. Actually we have a family of such operators indexed by I . Where α and β are terms of sort i and $\delta \in KB$, **if δ then α else β** = α if $\delta = tt$, β if $\delta = ff$ and \perp_i if $\delta = \perp$.

We shall write $\Sigma_{\mathcal{A}}$ for the two-sorted signature where $Op(\Sigma_{\mathcal{A}}) = \{\text{succ}, \text{plus}, \text{times}, \text{le}, \text{ge}, \text{diff}, \text{neg}, \text{conj}\}$. $Const(\Sigma_{\mathcal{A}}) = \{0, 1, \perp_{Nat}, tt, ff, \perp\}$. Where $I = \{Nat_{\perp}, KB\}$, the arity of **succ** is $\langle(Nat_{\perp}), Nat_{\perp}\rangle$ and the arity of **plus**, **times** and **diff** is $\langle(Nat_{\perp}, Nat_{\perp}), Nat_{\perp}\rangle$. The arity of both **le** and **ge** is $\langle(Nat_{\perp}, Nat_{\perp}), KB\rangle$.

The Σ -algebra we have in mind, then is:

$$\mathcal{A} = \langle Nat_{\perp}, KB, =_{Nat}, =_{KB}, ', +, *, <, >, -, \neg, \wedge, 0, 1, \perp_{Nat}, tt, ff, \perp \rangle$$

Here Nat_{\perp} is the flat cpo of natural numbers, with the order relation $x \sqsubseteq y \Leftrightarrow x = y \vee x = \perp_{Nat}$. All the operations on Nat_{\perp} are strict; not so those on KB .²

Where Σ -structure \mathcal{A} is as above, $LU(\mathcal{A})$ is the unique Σ -structure whose two sorts are the set of all ω -sequences of Nat_{\perp} or KB , respectively. (We do not allow sortally heterogeneous sequences.) For $n \in \omega$ and $\alpha \in {}^{\omega}A_i, i = Nat_{\perp}$ or KB , we write α_n rather than $\alpha(n)$. Moreover, where f is an operation symbol in $\Sigma_{\mathcal{A}}$, of arity $\langle \vec{i}, j \rangle, \alpha, \beta, \dots \in {}^{\omega}A_i$, and $n \in \omega$, $(f^{LU(\mathcal{A})}(\alpha, \beta, \dots))_n = f^{\mathcal{A}}(\alpha_n, \beta_n, \dots)$. Here (α, β, \dots) is a finite sequence whose length = $|\vec{i}|$ and where, given our disallowing of heterogeneous sequences, each of α, β, \dots is a sequence of elements of A_i , depending on whether each $i \in \vec{i}$ is Nat_{\perp} or KB . Finally $c \in Const(\Sigma_{\mathcal{A}})$ whose arity is i are assigned constant sequences in ${}^{\omega}A_i$. $LU(\mathcal{A})$ is a cpo with the order induced by the

²An operation f over \mathcal{D} is *strict* if $f(\vec{x}) = \perp_{\mathcal{D}}$, whenever $\perp_{\mathcal{D}} \in \vec{x}$. Kleene's weak three-valued rules are strict; we choose to follow the rules for the strong 3-valued system both because they are closer to [?] and have a more direct computational motivation.

pointwise \sqsubseteq -order on elements; so, for $\alpha, \beta \in {}^\omega Nat_\perp$, $\alpha \sqsubseteq \beta$ iff $\forall n (n \in \omega) \alpha_n \sqsubseteq \beta_n$. The bottom element is the empty sequence $\langle \rangle$.³

The foregoing construction transfers the operators (and constants) of \mathcal{A} pointwise to the structure $LU(\mathcal{A})$. To obtain a Lucid signature and a corresponding algebraic structure, we must add further operator symbols associated with functions on ω -sequences. In what follows we shall limit ourselves to the nonnested fragment of Lucid (the so-called Luswim fragment).⁴ We add the following operation symbols: **first**, **next**, **assoonas**, **whenever**, **fby**. Where $I = \{{}^\omega Nat_\perp, {}^\omega KB\}$, the arities of these symbols are as follows:

- $arity(\mathbf{first}) = arity(\mathbf{next}) = (\langle {}^\omega i \rangle, {}^\omega i)$, $i = Nat_\perp$ or KB
- $arity(\mathbf{assoonas}) = arity(\mathbf{whenever}) = (\langle {}^\omega i, {}^\omega KB \rangle, {}^\omega i)$
- $arity(\mathbf{fby}) = (\langle {}^\omega i, {}^\omega i \rangle, {}^\omega i)$.

The associated operators $f^{LU(\mathcal{A})}$ are governed by the following equations: Where $\alpha, \beta \in {}^\omega A_i$, and suppressing reference to the fixed structure $LU(\mathcal{A})$:

- $\mathbf{first}(\alpha) = \langle \alpha_0 \rangle^\omega$. Thus the value of $\mathbf{first}(\alpha)$ is the ω -sequence whose only element is the first element of α .
- $\mathbf{next}(\alpha) = \alpha^1$. The value of $\mathbf{next}(\alpha)$ is the ω sequence determined by removing α 's first (0^{th}) element.
- $\mathbf{assoonas}(\alpha, \beta) = \alpha^n$, if there is an n such that $\beta^n = tt$ and $\forall m < n, \beta^m = ff$; \perp_i , otherwise. The value of $\mathbf{assoonas}(\alpha, \beta)$ is the sequence α^n , where the n^{th} element of β is tt , and no earlier element of β is tt .
- $\mathbf{whenever}(\alpha, \beta) = \langle \alpha_j \rangle_{j \in J}$, where $J = \{m \mid \beta_m = tt\}$ and where if there is an n such that $\beta_n = tt$ & $\forall r > n, \beta_r \neq tt$, then for all $r > n, \alpha_r = \perp_i$. The value of $\mathbf{whenever}(\alpha, \beta)$ is the sequence σ determined as follows: if the m^{th} element of β is tt , then the m^{th} element of α is in σ , and it follows in σ the k^{th} element of α where k is the greatest predecessor of m such that $\beta_k = tt$. Moreover, if there is a number n such that β_n is the last tt in β , then for all numbers $r > n, \sigma_r = \perp_i$.
- $\mathbf{fby}(\alpha, \beta) = \alpha_0 \frown \beta$. The value of $\mathbf{fby}(\alpha, \beta)$ is the concatenation of the first element of α with β .

One important consequence of these equations is the following crucial difference between streams and sequences. Let $\alpha = \langle 0, 1, 2, \perp_{Nat}, 4 \dots \rangle$. Then $\mathbf{first}(\mathbf{next}^4(\alpha)) = \langle 4, 4, 4, \dots \rangle$. That is, we can access an element of a sequence even when some

³Lucid algebras whose carriers are sets of infinite sequences, cannot have a computable identity relation, that is the computable eq operation is not real identity between ω -sequences.

⁴Full Lucid allows nested sequences, that is, ω -sequences whose elements are themselves such sequences.

earlier element represents a divergent computation; this is not true of streams. Thus even if the operations over the underlying value domain are strict, the sequence operations of **Lucid** are non-strict.

7 Is Lucid an Intensional Language?

In the semantics of both (linear-time) temporal logic and **Lucid**, ω -sequences play a crucial role. In the first case what is involved are ω -sequences of states or, on the simpler semantics with which we began, an ω -sequence of times. In the semantics of temporal logic, these sequences model the dimension(s) of variation relative to which the truth-values of sentences are evaluated. If we had extended our treatment to first-order intensional languages, we would have seen that these sequences play a similar role with respect to the values of individual terms. Moreover, as we have stressed above, it is characteristic of intensional languages that they include no constructions that directly or explicitly refer either to these dimensions of variation or to their elements. Thus, in a standard first-order temporal logic, there are no terms for times or for sequences of times. It is rather in the semantic, metalinguistic, account of constructions involving intensional operators that these implicit parameters play their part.

With **Lucid**, on the other hand, the role of sequences is very different. The native data structures of **Lucid** are precisely ω -sequences. The distinctive operators, e.g., **first**, **assoonas**, are operators on such sequences, and the values of such operators are, again, ω -sequences. Moreover consider the difference in role between the elements of computation sequences or paths and the elements of **Lucid** sequences. In the case of paths, we are dealing with abstract models of the states of a machine, states that are alterable via the execution of the statements, in particular, the side-effecting assignments of an imperative programming language. In the case of **Lucid**, we are dealing rather with ω -sequences of elements from some underlying domains of values. These values, e.g., natural numbers or booleans, need not be thought of as models of any aspect of a state-machine. Indeed, **Lucid** is a purely functional programming language; none of its constructs effect changes in state.

7.1 On Assignments

Compare the following two programs. The first is from a generic, otherwise unspecified imperative programming language; the second, is a simple **Lucid** program.

I. $x := 1; x := x + 1$ II. $\text{first } x = 1, \text{ next } x = x + 1$

Clearly program I *cannot* be understood as a set of equations in which x is e.g., a numerical variable. Thus for instance, we cannot take it that value of the variable x is fixed throughout the program, relative to a numerical assignment, for then the

second statement would assert that $1 = 1 + 1$. If we do think of x as a numerical variable, denoting a number relative to an assignment, then the context $x := \dots$ is referentially opaque. Indeed, within the second assignment, it is not all clear that the two occurrences of x refer to the same thing, for what is it to add 1 to a register?

Program I is in a language for which it is appropriate to think of some notion of a *machine state* and of *commands* that change that state. In the simplest case, we can think of these states as determined by the values (contents) of a set of registers (locations). The meaning of program I is as follows: initialize register x to have the value 1, then for every succeeding stage (state) in the execution sequence of I, increment the value of x by 1. Thus, the x in $x + 1$ on the right of ‘ $=$ ’ does not occur as a numerical variable but again, as an identifier; but in this usage it denotes the value of the register denoted by x in the state, let us call it, p_i , in which the assignment is executed. Thus, at least in the simple case we are imagining, the value of x at p_{i+1} is to be one greater than the value of x at p_i . Notice that we understand $\alpha; \beta$ not as the composition of the numerical functions denoted by α, β —for in program I, no numerical functions are denoted—but as expressing that β is to be executed in the state that results from the execution of α . Thus, if one thinks of the meaning (intension) of deterministic programs as functions from state to state, then $;$ corresponds to composition of state-functions.

Here we have an implicit relativity, not of the truth-value of a sentence to time, but of the denotation of a term to state. That is, it is precisely of the essence that the value of a register is a state-dependent, that is, variable, feature of that register. Registers (locations) are, then, to be modeled by functions from computational states to values, e.g., numbers. How then should we think of identifiers, which are singular terms for locations? We take our clue from the treatment of the semantic value of sentences in temporal logic. There, in order to capture the implicit relativity of the truth of sentence to time we moved from assigning truth-values to sentences to assigning functions from time to truth-values. In the present context, in order to capture the relativity of denotation to state, we assign to locations functions from state to numbers (or more generally, functions from state to elements of the set of storable values).

If we were to present a full account of the syntax and semantics of some imperative programming language, we would introduce (at least) two sorts of constants, one for locations (identifiers) and one for numbers. Where S is the set of states, a natural way to treat the semantic domain of the former would be $S \rightarrow N$. (This treatment essentially identifies a location and its unique identifier. We thus ignore the complications that ensue from having locations among the storeable values or from allowing complex expressions on the left of the $:=$.) We won’t present such an account. Rather we shall simply display, using our previous notational conventions, the condition on execution sequences associated with program I. Where p is an execution sequence for the programming language in question: $I(x)(p_0) = 1 \ \& \ \forall i \geq 1, I(x)(p_{i+1}) = 1 + I(x)(p_i)$. The crucial point is that the value (reference) of an identifier is relative to an implicit parameter of state, and thus, that a natural framework for the semantics of imperative programming languages is intensional logic.

7.2 Program II in Lucid

What of program II?

II. first $x = 1$, next $x = x + 1$

Here the x in all of its occurrences on either side of the identity is a variable ranging over ω -sequences of natural numbers. It is not an identifier associated with or referring to an updatable register of some state-machine. Nor does it range over the set of computation sequences corresponding to some programming language being executed on such a machine. That is, it is neither an object language nor a meta-language variable corresponding to a dimension of variation relative to which x or terms containing it are to be evaluated.

In fact, there are two ways to think of this, and any other **Lucid** program; (i) statically or denotationally; (ii) procedurally or dynamically. On the static conception, **Lucid** programs are simply sets of equations (meeting certain constraints) in an interpreted algebraic language of a certain kind (as described in §6.1), and the values (interpretations) of such programs are elements of the algebra of the interpretation. In particular, the value of a program is the least solution to the equations in the program. In the case of program II, the sequence $\langle 1, 2, 3, \dots \rangle$ is provably the least solution, and this is the value of the program. This same sequence is also the value of program III:

III. $x = 1$ fby $x + 1$

Lucid is a functional purely extensional language, and any occurrence of the equation in program II can be replaced by the two equations in program III in any larger program, without effecting the value (solution) of the larger containing program. (There are syntactic restrictions on the well-formedness of programs, of course; the intent of the main such restriction is that any variable in a set of equations can only be defined once.)

As noted, the variable x does not refer to, nor is it in any way associated with an aspect of the state of a machine that can be effected by the execution of a **Lucid** program in which the variable occurs. Its reference is in no way state-dependent. Despite this, there is a dynamic way to understand **Lucid** programs that does involve something akin to change of state. Each variable in a **Lucid** program refers to an ω -sequence. An ω -sequence of elements of a domain \mathcal{D} is a function from the natural numbers or finite ordinals to \mathcal{D} . In this respect, the reference of **Lucid** variable is like a path, an ω -sequence of states. In our treatment of \mathcal{L}_{PTL} in §5, we noted that the value of an expression was determined only relative to a state or stage on a path and that thus there was no such entity as the nonrelative value of the expression. (There we were interested only in the truth-values of sentences, but the same point could

be made for individual terms.) As we have seen in **Lucid** the situation is different. We do have explicit object-language terms for infinite sequences, and explicit object-language operators on them, and such sequences are themselves the absolute, non relative, values of our variables and terms. But, if we are interested only in the elements of the underlying domains of values, e.g., numbers and booleans, and in the operations on them, e.g., $+$, \wedge , then looks quite different, and it is the dynamic perspective on **Lucid** that captures this difference. For the dynamic perspective focuses on the pointwise nature of the evaluation of **Lucid** operators.

No **Lucid** variable can refer to a number or a boolean, and none of the proper **Lucid** operators apply to them. If we want to know what is the numerical value (referent) of the sequence $\langle 1, 2, 3, 4, \dots \rangle$, obviously there is no answer, but just as obviously, there is an answer to the question, what number is the n^{th} element of the sequence. Moreover, as in the case with states and paths, a value can occur in many different positions in a single sequence and can occur as well as an element in many different sequences.

Since we are not dealing with an imperative, side-effecting programming language, we shall replace talk of states with talk of *stages*. For each **Lucid** program P , we posit a linearly ordered set of stages S , isomorphic to the linearly ordered set of times, hence to the numbers under $<$. Let s_i be the i^{th} stage. As we remarked above, each **Lucid** program P consists of a set of equations; let Var_P be the set of variables occurring in P . We can thus think of a stage s_i as associating with each $v \in \text{Var}_P$, the i^{th} element of the sequence α^v , where α^v is the minimal solution of v in P . We call this element (α_i^v) the *basic value* of v (in P), relative to s_i . The notion of the basic value of a sequence relative to a stage is analogous to that of the extension of a term at a state/time. Correlatively, the proper value of a **Lucid** variable is *like* the intension of a term in (first-order) temporal logic; the former is a function from natural numbers to basic values, the latter a function from times/states, identified by a natural number index, to (basic) values.

7.3 Conclusion

At the end of §2, we briefly sketched an alternative treatment, in terms of quantification over a domain of times, of the temporal relativity of the truth-values of sentences captured by \mathcal{L}_{PTL} . The quantified theory can be thought of as making explicit what was implicit in the associated intensional logic. A proper treatment of a path- or state-based intensional logic would involve quantification over paths—infinite sequences of states. As the static conception of **Lucid** makes clear, the elements of **Lucid** algebras are precisely infinite sequences, but they are not infinite sequences of states or times. That is, **Lucid** should not be thought of as in some sense making explicit what is implicit in path-based temporal logics, or in state-based logics of programs. In particular, within the static conception there is no constraint that the n^{th} element of a sequence can be evaluated only after the first $n - 1$ elements have been evaluated, for from the static conception, the entire infinite sequence is given simply as an element of the model of the program.

From the dynamic perspective however, the set of indices for the sequence, that is the elements of ω , appear as representing stages in the evaluation or even construction of the sequence. As we have noted, this appearance is in certain respects misleading, but its naturalness accounts for fact that **Lucid** can be so easily (mis)understood as an imperative programming language based on assignment and iteration. It also accounts for the ease with **Lucid** is understood as an intensional language.

References

- [1] E. A. Ashcroft and W. W. Wadge. *Lucid—A Formal System for Writing and Proving Programs*. **SIAM J. Comput.**, Vol. 5, No. 3, Sept. 1976, ps. 336-354.
- [2] W. W. Wadge and E. A. Ashcroft. **Lucid, The Dataflow Programming Language**. Academic Press, London. 1985.