Incremental interpretation and dynamic semantics

Adrian Brasoveanu and Jakub Dotlačil

May 9, 2015

Abstract

The goal of this paper is to motivate and define a strictly incremental semantics for dynamic predicate logic (Groenendijk and Stokhof 1991). In particular, we will extend the incremental semantics for dynamic propositional logic introduced in Vermeulen (1994) to first-order predicate logic (borrowing central notions from Visser 2002). We call the resulting logical system Incremental Dynamic Predicate Logic and we show how this system can be used to derive correct truth conditions for apparently non-incremental structures like donkey conditionals *in a strictly incremental fashion*. That is, the correct meanings for donkey conditionals are derived by means of a strictly left-to-right compositional procedure. This is accomplished without having to type-shift the meanings of the individual words (as in Steedman 2001, for example), and with dynamic conjunction / sequencing as the only compositional operation.

1 Introduction and basic proposal

The goal of this paper is to motivate and define an incremental semantics for Dynamic Predicate Logic (DPL; Groenendijk and Stokhof 1991), i.e., to extend the incremental semantics for Dynamic Propositional Logic (DPropL) introduced in Vermeulen (1994) to first-order predicate logic. We call the resulting logical system Incremental Dynamic Predicate Logic (IDPL) and we show how this system can be used to derive correct truth conditions for apparently non-incremental structures like donkey conditionals in a strictly incremental fashion, i.e., strictly left-to-right and word by word.

Let us follow Vermeulen (1994, pp. 244-246)) for the time being, and focus exclusively on semantics at the propositional / sentential level. We want an incremental semantics for DPropL to respect three principles:

(1) a. **Incrementality**: we can interpret texts as we hear them.
    b. **Pure compositionality**: 'pure' – we do not assume that a full syntactic analysis precedes interpretation; unlike standard (neo)Montagovian semantics, or even the incremental DRS construction algorithm in Kamp and Reyle (1993)
    c. **Break-in**: every segment of a text should be interpretable, even if what comes after, or came before, is unknown; wherever we 'break in' in a text, interpretation should be possible

Together, (1a-1c) entail **associativity**: text meanings have to form an algebra with an associative operation ('merger' / conjunction) by which the meanings can be glued together.
This straightforwardly captures texts that are actually conjoined:

(2) a. Bob inherited a donkey (p), and Jane bought it from him (q), and she sold it to Bill (r).
    b. $[\![(p; q); r]\!] = [\![p; (q; r)]\!]$

The problem is that conditionals do not have an associative semantics. The text in (3) below is intuitively interpreted as in (4), not as in (5): if p is false, the text is false, not true.

(3) The driver was not working that night (p) and if the butler was working that night (q), the butler committed the murder (r). $\rightsquigarrow p; \mathbf{if}; q; \mathbf{then}; r; \mathbf{end}$

(4) Intended interpretation: $p; (\mathbf{if}; q; \mathbf{then}; r; \mathbf{end})$ (in classical propositional logic: $p \wedge (q \to r)$)

(5) By associativity: $(p; \mathbf{if}; q); \mathbf{then}; r; \mathbf{end}$ (in classical propositional logic, basically: $(p \wedge q) \to r$)

An incremental, fully associative semantics forces the bracketing in (6) below, which is equivalent to the incorrect interpretation in (5):

(6) $((((p; \mathbf{if}); q); \mathbf{then}); r); \mathbf{end}$

How can we provide an associative semantics for conditionals that derives the right truth conditions? The formulas in (3-6) already hint at the basic solution: everything is conjoined / merged with an associative operation – namely, ; – and the right truth conditions are derived by using three special formulas **if**, **then**, and **end**.

Of course, what matters are not the syntactic objects **if**, **then**, and **end**, but the way they are interpreted, i.e., the way they update / manipulate semantic evaluation contexts.

Which brings us to the basic solution pursued in Vermeulen (1994): memory. The semantic evaluation contexts will keep track of the denotations of the previous formulas, i.e., they will be update histories. The formulas **if**, **then**, and **end** leverage these detailed, structured evaluation contexts and manipulate them in specific ways so that the correct truth conditions for conditionals are derived – in a purely associative manner.

(7) We interpret (4) as follows:

> We store the information that p in our memory before we interpret q. This information [contributed by q] is again stored before we interpret r. Now we can construct from the information that we have stored the information that **if** q **then** r. Finally this information can be added to the information that p. [W]e do not need brackets to tell us how [. . .] to store the information: the special elements **if**, **then** and **end** will tell us exactly what has to be done. (Vermeulen 1994, p. 248)

2 Incremental semantics for Dynamic Propositional Logic (DPropL) with sequences

2.1 DPropL syntax

(8) **The syntax of DPropL.**
Given a set of atomic texts (i.e., atomic propositional variables) A, the set of texts based on A (i.e., the set of well-formed propositional formulas), abbreviated T_A, is the smallest set such that:

a. $A \subseteq T_A$
b. $\perp \in T_A$
c. $\{\mathbf{if}, \mathbf{then}, \mathbf{end}\} \subseteq T_A$
d. If $\varphi, \psi \in T_A$, then $\varphi; \psi \in T_A$ (text concatenation / conjunction).

Intuitively, $\perp$ is the formula that is always false, i.e., $[\![\perp]\!] = \mathbb{F}$. We will overload the symbol $\perp$ and use it both for the syntactic object and for its semantic value.

The choice of basic expressions in DPropL is driven by our main goal for this logic: show how we can provide an associative semantics for a conditional appearing in a text like (3) above. In particular, show how semantic composition can proceed fully incrementally in a left-to-right, strictly word-by-word fashion, and yet derive the intuitively correct truth conditions for the text, i.e., truth conditions in which the conditional antecedent consists of only the second sentence (q), and the first sentence (p) is interpreted as conjoined with the full conditional, not only with the conditional antecedent. That is, we want our logic to derive the equivalence below:

(9) $((((p; \mathbf{if}); q); \mathbf{then}); r); \mathbf{end} \Leftrightarrow p; (\mathbf{if}; q; \mathbf{then}; r; \mathbf{end})$

The choice of atomic formulas and sentential operators in (8) above is driven by the minimal syntactic resources we need to formally set up the problem of deriving the equivalence in (9) above. That is, we need a basic supply of atomic sentences A (8d), and a basic way of concatenating / conjoining / merging simpler texts into more complex ones (8d). Furthermore, our basic idea of how to derive the equivalence in (9) is to introduce three special atomic formulas $\mathbf{if}$, $\mathbf{then}$, and $\mathbf{end}$ that can be conjoined / merged with any other texts in the usual ;-based way, but that will receive a particular interpretation / semantic value. Whatever semantic values we end up assigning them, they need to end up deriving the correct truth conditions for a conditional. These truth conditions are assembled by means of three atomic formulas rather than a single binary operator (as in classical logic) precisely because we want the interpretation to be fully incremental.

Why go to all this trouble only for conditionals? Conditionals are just a very simple, propositional-level example of texts with non-associative meanings. If we find a way to interpret them fully incrementally, i.e., in a fully associative semantics, we can probably generalize that solution to all other similarly non-associative semantic operators, for example, quantifiers or adverbs of quantification. Quantifier restrictors have to be semantically combined with nuclear scopes first, and with the surrounding text only later, in much the same way that conditional antecedents need to be combined with the consequents first.

Similarly, we hope to be able to generalize our solution to other structures, e.g., conditionals with sentence final *if*-clauses:

(10) The butler committed the murder (r) if the butler was working that night (q). $\rightsquigarrow \mathbf{then}; r; \mathbf{if}; q; \mathbf{end}$

As Milward and Cooper (1994) were the first to explicitly notice (to our knowledge), incremental left-to-right interpretation on one hand, and semantic interpretation as needed to derive the correct truth conditions on the other hand place opposite requirements on how the interpretation of sentences like (10) should proceed. Incremental interpretation requires the consequent r to be interpreted before the antecedent q, while truth-conditionally, the antecedent q needs to be interpreted first and the consequent r needs to be interpreted relative to, and therefore after, the antecedent. This problem is not restricted to classical static semantics, it applies equally to dynamic frameworks (Kamp 1981, Heim 1982, Groenendijk and Stokhof 1991) despite the fact that these frameworks have a notion of incrementally 'threading' interpretation contexts between sentences (as reflected in the meaning of dynamic conjunction). For more discussion of incremental interpretation and related issues from this perspective, see also Chater et al. (1995).

Finally, the only clause in (8) above that is left to discuss is the one introducing *falsum* / bottom (8c). We basically need this clause for the expressive completeness of our propositional logic: we have conditionals and conjunctions, but no negation. The simplest thing to do is to add a 0-ary propositional operator in terms of which we can define the unary negation operator, rather than add negation directly – and that 0-ary operator is $\bot$.

As shown in (11) below, negation can be defined in terms of implication and $\bot$, following the classical abbreviation $\neg\varphi := \varphi \to \bot$. Similarly, the formula $\top$ that is always true $[\![\top]\!] = \top$ can be defined in the classical way $\top := \neg\bot$ $(= \bot \to \bot)$. We will similarly overload the symbol $\top$ and use it both for the syntactic object and its semantic value. Finally, disjunction can be defined via the De Morgan laws, as shown below.

(11) Abbreviations:

a. $\neg\varphi := \mathbf{if}; \varphi; \mathbf{then}; \bot; \mathbf{end}$

b. $\top := \neg\bot$ $(= \mathbf{if}; \bot; \mathbf{then}; \bot; \mathbf{end})$

c. $\varphi \vee \psi := \neg(\neg\varphi;\ \neg\psi)$

2.2 A sequence-based associative Semantics for DPropL: the basic idea

As we already mentioned, Vermeulen (1994, pp. 244-246) argues that if the semantics for DPropL is to respect the principles of Pure Compositionality, Incrementality, and Break-in, the semantics will have to be associative, i.e.:

(12) For any texts / formulas φ, ψ, χ, we have that:
$[\![(\varphi;\psi);\chi]\!] = [\![\varphi;(\psi;\chi)]\!]$.

That is, the semantic value of text conjunction ; has to be an associative operation over the semantic values of the concatenated texts. Letting the semantic value of conjunction $[\![\,;\,]\!]$ be the operation $\bullet$, associativity requires it to satisfy the following constraint:

(13) For any texts / formulas φ, ψ, χ, we have that:
$([\![\varphi]\!] \bullet [\![\psi]\!]) \bullet [\![\chi]\!] = [\![\varphi]\!] \bullet ([\![\psi]\!] \bullet [\![\chi]\!])$.

The problem with this general associativity requirement is that the conditional formula $\mathbf{if}; \varphi; \mathbf{then}; \psi; \mathbf{end}$ does not have an associative semantics, as exemplified by (4) above.

The basic solution is *memory*, i.e., make our semantic evaluation contexts more structured / fine-grained in such a way that the history of previous updates (or at least the recent updated history) is kept track of. "In our semantics we will allow ourselves to have more than one slot where information can be stored. We will not only have a slot for our current state of information, but we will also have slots for some specific information states that we used to be in. So we remember our information *history*." (Vermeulen 1994, pp. 247-248)

In particular, we interpret (4) as follows: "We store the information that p in our memory before we interpret q. This information is again stored before we interpret r. Now we can construct from the information that we have stored the information that $\mathbf{if}\ q\ \mathbf{then}\ r$. Finally this information can be added to the information that p. Note that we do not need brackets to tell us how we have to store the information: the special elements $\mathbf{if}$, $\mathbf{then}$ and $\mathbf{end}$ will tell us exactly what has to be done." (Vermeulen 1994, p. 248)

Our goal now is to formally define the semantics of DPropL so that we derive the informal interpretation of conditionals outlined in (7) above.

All the update / interpretation operations in (7) will be encoded in the recursive definition of the interpretation function $[\![\cdot]\!]^{\mathfrak{M},\mathfrak{c}}$; in particular, they will be encoded in the ways evaluation contexts $\mathfrak{c}$ (which will encode update histories) are manipulated / updated when we interpret conjunction ; and the special formulas $\mathbf{if}$, $\mathbf{then}$, and $\mathbf{end}$.

2.3 Model structures: extended monoids

To define the interpretation function $[\![\cdot]\!]^{\mathfrak{M},\mathfrak{c}}$, we need to discuss the kind of structures we will use for models $\mathfrak{M}$, and for semantic evaluation contexts $\mathfrak{c}$.

While the semantic evaluation contexts $\mathfrak{c}$ are more complex than the usual ones in classical (static) or dynamic logic on account of the fact that they have to encode update histories, the models $\mathfrak{M}$ themselves are going to have a fairly simple structure: we will take them to be *extended monoids* in the sense of Visser (2002). Vermeulen (1994) takes them to be Heyting algebras, but the extended monoids of Visser (2002) are both more general, which will be useful when we move on to Dynamic Predicate Logic, and more directly related to the relational models used in dynamic semantics.

Despite their mathematical simplicity, DPropL models can seem fairly abstract and unfamiliar to semanticists, so we will take some time to introduce them more leisurely. Part of the abstractness of the models is due to the fact that we are looking at propositional logic, so we will abstract away from the richness of meaning that we find at sub-sentential level. This is just as in classical logic: the meaning / semantic value of the sentence *John likes Mary* is very abstract when we're only concerned with meanings at the sentential / propositional level; the meaning is either $\top$ or $\bot$. In contrast, models for first-order logic are more complex, but they are also more intuitive because they give us a basic handle on how various sub-sentential pieces contribute to the meaning of the overall sentence.

We face a similar trade-off when we introduce models for DPropL: mathematically, they are very simple, but this simplicity / abstractness obscures the intuitions that underlie various mathematical notions and formalization choices. We will therefore discuss in a fair amount of detail how these simple / abstract models can be made more complex / concrete in a way that anticipates the semantics of our Incremental Dynamic Predicate Logic.

What should our models consist of? At a minimum, they need to contain an associative operation over denotations that corresponds to conjunction – and we want the space of denotations to be closed under this operation. For conjunction to behave in the intended way, there has to exist an identity element (basically, $\top$) for it. Finally, we also want a $\bot$ element and an binary operation that would correspond to implication. An extended monoid in the sense of Visser (2002) provides exactly this kind of structure.

2.3.1 Monoids

In particular, our monoid will be defined over a set $\mathbf{I} = \{i, j, k, \dots\}$ of propositional denotations. In the spirit of dynamic semantics, we label the elements of this set information states, but we should be clear that $\{i, j, k, \dots\}$ do not encode the partial variable assignments ('embeddings') of Discourse Representation

Theory (DRT; Kamp 1981, Kamp and Reyle 1993) or File Change Semantics (FCS; Heim 1982), or the total variable assignments of Dynamic Predication Logic (DPL; Groenendijk and Stokhof 1991). They are instead meant to encode full Discourse Representation Structure (DRS) / DPL formula denotations, i.e., binary relations over partial / total variable assignments. Of course, since we're working at the propositional level right now, the information states $i, j, k, \dots \in \mathbf{I}$ are atomic – they do not have any internal structure. But various abstract notions and definitions at the propositional level become easier to understand if we already anticipate how they will be applied when we move to predicate logic.

Thus, the operation $\bullet$ denoted by conjunction / concatenation ; together with the space of information states $\mathbf{I}$ over which updates operate (and which are assembled into update histories) have to form a monoid – defined in (14) below. We will first discuss two examples of monoids, after which we will discuss how to extend monoids with the two operators $\perp$ (*falsum*) of arity 0 and $\twoheadrightarrow$ (implication) of arity 2.

(14) A monoid is a tuple $\langle \mathbf{I}, \bullet, \mathbf{id} \rangle$ consisting of a set $\mathbf{I}$, a binary operation $\bullet$ over that set, and a designated element $\mathbf{id} \in \mathbf{I}$ that satisfies the following three axioms:

Ax1 Closure: for all $i, j \in \mathbf{I}$, $i \bullet j \in \mathbf{I}$.

Ax2 Associativity: for all $i, j, k \in \mathbf{I}$, $(i \bullet j) \bullet k = i \bullet (j \bullet k)$.

Ax3 Identity element: there is an element $\mathbf{id} \in \mathbf{I}$ such that for all $i \in \mathbf{I}$, $\mathbf{id} \bullet i = i \bullet \mathbf{id} = i$.

Example 1: the finite-string monoid. For example, the set of all finite strings over a fixed alphabet Σ, e.g., $\Sigma = \{a, b, c\}$, forms a monoid with string concatenation $^\frown$ as its binary operation and the empty string ϵ as the identity element. This set of finite strings, called Σ^*, is of the form:

(15) $\Sigma^* = \{\epsilon, a(= \epsilon ^\frown a = a ^\frown \epsilon), b, c, aa(= a ^\frown a), bb(= b ^\frown b), cc(= c ^\frown c),$
$ab(= a ^\frown b), ba, ac, ca, bc, cb, aaa, aab, bcb, cccaabb, \dots\}$

We can verify that the tuple $\langle \Sigma^*, ^\frown, \epsilon \rangle$ is a monoid by checking that it satisfies the three axioms in (14) above.

(16) **Ax1** Closure: for all $\alpha, \beta \in \Sigma^*$, $\alpha ^\frown \beta \in \Sigma^*$; e.g., $aabb, ccbb \in \Sigma^*$, and also $aabbccbb \in \Sigma^*$ ✓

Ax2 Associativity: for all $\alpha, \beta, \gamma \in \Sigma^*$, $(\alpha ^\frown \beta) ^\frown \gamma = \alpha ^\frown (\beta ^\frown \gamma)$, e.g., $(a ^\frown b) ^\frown c = a ^\frown (b ^\frown c) = abc$ ✓

Ax3 Identity element: the empty string ϵ, since for all $\alpha \in \Sigma^*$, $\epsilon ^\frown \alpha = \alpha ^\frown \epsilon = \alpha$ ✓

Example 2: the binary-relation monoid. Another example, which will be directly useful to us, is the monoid formed by the set of all binary relations over a set S, with relation composition $\circ$ as the binary operation and the identity relation on S as the identity element (the three axioms in (14) are easy to verify):

(17) Given a set S (of possible worlds, variable assignments, etc.), the tuple $\langle \mathbf{R}, \circ, R_{\mathbf{id}} \rangle$ is a monoid, where:

a. $\mathbf{R} = \{R : R \subseteq S \times S\} = \wp(S \times S)$,
where $S \times S = \{\langle x, y \rangle : x \in S \wedge y \in S\}$ and $\wp$ is the powerset operation

b. $R \circ R' = \{\langle x, y \rangle : \exists z (xRz \wedge zR'y)\}$
(in prefix notation: $R \circ R' = \{\langle x, y \rangle : \exists z (R(x, z) \wedge R'(z, y))\}$)

c. $R_{\mathbf{id}} = \{\langle x, x \rangle : x \in S\}$

2.3.2 The 'entailment' partial order $\leqslant$

Let us return now to the general definition of monoids in (14) above with an eye towards extending it with a zero element $\perp$, which we will use as the denotation for *falsum*, and another binary operation $\twoheadrightarrow$, which we will use as the denotation for implication. First, for any monoid we can define a partial order $\leqslant$ as follows:

(18) We define a partial order $\leqslant$ over the elements of $\mathbf{I}$: for any $i, j \in \mathbf{I}$, $i \leqslant j$ iff $i \bullet j = i$.

Intuitively, this partial order encodes a notion of entailment. Its definition has the same structure as the following basic theorem of propositional logic ($\vDash$ symbolizes entailment): $p \vDash q$ iff $p \wedge q \mathrel{=\!\!|\!\!\vDash} p$.

Example 1: $\leqslant$ for the finite-string monoid. For the finite-string monoid, this partial order includes only pairs of strings whose second member is the empty string ϵ: $\alpha ^\frown \beta = \alpha$ can be the case only if $\beta = \epsilon$.

Example 2: $\leqslant$ for the binary-relation monoid. For the binary-relation monoid, this partial order is much richer: $R \circ R' = R$ can be the case whenever the range of R (defined as (20) below) is a subset of the domain of R' (defined as in (19) below) – which is exactly the notion of entailment in DPL (Groenendijk and Stokhof 1991, p. 67, Definition 20), symbolized as $\vDash_{\text{DPL}}$ below.

(19) $\mathbf{Dom}(R) := \{x \in S : \exists y \in S(xRy)\}$

(20) $\mathbf{Ran}(R) := \{y \in S : \exists x \in S(xRy)\}$

(21) $\varphi \vDash_{\text{DPL}} \psi$ iff $\mathbf{Ran}([\![\varphi]\!]) \subseteq \mathbf{Dom}([\![\psi]\!])$ (Groenendijk and Stokhof 1991, p. 67, Definition 20)

(22) A DPL-style definition of the partial order $\leqslant$ for the binary-relation monoid:
$R \leqslant R'$ iff $\mathbf{Ran}(R) \subseteq \mathbf{Dom}(R')$ and $R' \subseteq R_{\mathbf{id}}$

For the DPL-style notion of partial order in (22) above to actually satisfy the condition in (18), we need the additional assumption that $R' \subseteq R_{\mathbf{id}}$ – as we already indicated in (22). But this is in fact a harmless assumption for our purposes. If following Visser (2002), we define $\mathbf{diag}(R)$ (the diagonal of the binary relation R) as follows

(23) $\mathbf{diag}(R) = \{\langle x, x\rangle : x \in \mathbf{Dom}(R)\}$

it is easy to see that:

(24) For any two DPL formulas φ and ψ: $\mathbf{Ran}([\![\varphi]\!]) \subseteq \mathbf{Dom}([\![\psi]\!])$ iff $\mathbf{Ran}([\![\varphi]\!]) \subseteq \mathbf{Dom}(\mathbf{diag}([\![\psi]\!]))$.

Since by definition $\mathbf{diag}(R) \subseteq R_{\mathbf{id}}$ for any R, we see that the definition of DPL entailment can still be applied in full generality even with the additional assumption in (22).

It is easy to show that the DPL-style definition of the partial order $\leqslant$ in (22) satisfies the condition in (18):

(25) Let the partial order $\leqslant$ over binary relations be defined as: $R \leqslant R'$ iff $\mathbf{Ran}(R) \subseteq \mathbf{Dom}(R')$ and $R' \subseteq R_{\mathbf{id}}$. If $R \leqslant R'$, it then follows that $R \circ R' = R$. To see that, consider an arbitrary pair $\langle x, y\rangle$:
a. $\langle x, y\rangle \in R \circ R'$ iff [by (17b)]
b. there is a z such that xRz and $zR'y$ iff [since $R' \subseteq R_{\mathbf{id}}$]
c. xRy and $y \in \mathbf{Dom}(R')$ iff [since $\mathbf{Ran}(R) \subseteq \mathbf{Dom}(R')$]
d. xRy iff
e. $\langle x, y\rangle \in R$

2.3.3 *Falsum* $\perp$

Defining the 'entailment' partial order $\leqslant$ is just the first step towards extending our monoids with a notion of *falsum* and a binary operation that corresponds to implication – since this is what we need for our DPropL semantics.

With $\leqslant$ in hand, we define – and require the existence of – a zero element $\perp$ as follows:

(26) There exists $\perp \in \mathbf{I}$ such that for any $i \in \mathbf{I}$, $\perp \bullet i = i \bullet \perp = \perp$.[1]

That is, $\perp$ is the least element in the partial order $\leqslant$ (both on the left and on the right of the $\bullet$ operation).

Not all monoids can be extended in this way. For example, it is not clear how to extend the finite-string monoid with a $\perp$ element. But the binary-relation monoid has a natural $\perp$ element:

(27) $R_\perp = \varnothing \quad (= \{\langle x, x\rangle : x \in S \wedge x \neq x\})$

It is easy to see that for any relation $R \in \mathbf{R}$, we have that $R \circ R_\perp = R_\perp \circ R = R_\perp$.

[1] Note that $\perp$ is unique. If not, there would be a $\perp' \in \mathbf{I}$, $\perp' \neq \perp$ satisfying the same 'least element' condition. By the definition of $\perp$, we would have $\perp \bullet \perp' = \perp' \bullet \perp = \perp$. By the definition of $\perp'$, we would have $\perp' \bullet \perp = \perp \bullet \perp' = \perp'$. Hence $\perp' = \perp$. Contradiction.

2.3.4 Implication $\twoheadrightarrow$

We also require the existence of a binary operation $\twoheadrightarrow$, which will provide the basic denotation for implication, satisfying the constraint below:

(28) We define a binary operation $\twoheadrightarrow$ from $\mathbf{I} \times \mathbf{I}$ to $\mathbf{I}$ such that for any $i, j, k \in \mathbf{I}$, $i \leqslant j \twoheadrightarrow k$ iff $i \bullet j \leqslant k$.

Continuing with the binary-relation monoid example, the binary operation $\twoheadrightarrow$ can be defined in the same way that dynamic implication is defined in DRT (Kamp 1981, Kamp and Reyle 1993), FCS (Heim 1982), and DPL (Groenendijk and Stokhof 1991).

Dynamic implication is externally static and internally dynamic. This means that the result of combining two binary relations R and R' is a subset of $R_{\mathbf{id}}$ (this is the externally static aspect), and the way the in which R and R' are combined involves linking the range of R and the domain of R' in some fashion, in a way that is similar but not identical to the way this happens in the case of relation composition $\circ$ (this is the internally dynamic aspect).

(29) For any $R, R' \in \mathbf{R}$: $\quad R \twoheadrightarrow R' = \{\langle x, x\rangle : x \in S \quad \wedge \quad \{y \in S : xRy\} \subseteq \{y \in S : \exists z(yR'z)\}\}$

Informally, $R \twoheadrightarrow R'$ is that subset of $R_{\mathbf{id}}$ which retains only the elements $x \in S$ whose image under the first relation R is included in the domain of the second relation R'. That is, any element y that is R-accessible from x is a good starting point for the second accessibility relation R'.

We can express the formula in (29) above more concisely if we use several abbreviations, in particular, the abbreviation $\mathbf{Dom}(R)$ for the domain of a relation R, and the abbreviation for the image of an element $x \in S$ under the relation R, symbolized as xR, and defined in (30) below. The image of x under R is the set of all possible R outputs for the input x.

(30) $xR := \{y \in S : xRy\}$

The definition of the binary operator $\twoheadrightarrow$ can now be rewritten as follows:

(31) For any $R, R' \in \mathbf{R}$: $\quad R \twoheadrightarrow R' = \{\langle x, x\rangle : x \in S \quad \wedge \quad xR \subseteq \mathbf{Dom}(R')\}$

Let us check that the definition of $R \twoheadrightarrow R'$ in (31) satisfies the condition in (28) above. To show this, we need an extra assumption, namely that $R'' \subseteq R_{\mathbf{id}}$. Just as in the case of the 'entailment' partial order $\leqslant$ above, this is harmless: it is easily seen that $R \twoheadrightarrow R' = R \twoheadrightarrow \mathbf{diag}(R')$. And the relation $\mathbf{diag}(R)$ is a subset of $R_{\mathbf{id}}$ by definition.

With the extra assumption that $R'' \subseteq R_{\mathbf{id}}$, we can show that the definition of $R \twoheadrightarrow R'$ in (31) satisfies the condition in (28).

(32) $R \leqslant R' \twoheadrightarrow R''$ iff $R \circ R' \leqslant R''$ (assuming $R'' \subseteq R_{\mathbf{id}}$):

a. $R \leqslant R' \twoheadrightarrow R''$ iff [by (18)]
b. $R \circ (R' \twoheadrightarrow R'') = R$ iff [since $R' \twoheadrightarrow R'' \subseteq R_{\mathbf{id}}$]
c. $\mathbf{Ran}(R) \subseteq \mathbf{Dom}(R' \twoheadrightarrow R'')$ iff [by (31)]
d. $\forall x \in \mathbf{Ran}(R)(xR' \subseteq \mathbf{Dom}(R''))$ iff [by (17b)]
e. $\forall x \in \mathbf{Dom}(R)(x(R \circ R') \subseteq \mathbf{Dom}(R''))$ iff [by (19, 20, 30)]
f. $\mathbf{Ran}(R \circ R') \subseteq \mathbf{Dom}(R'')$ iff [since $R'' \subseteq R_{\mathbf{id}}$]
g. $(R \circ R') \circ R'' = R \circ R'$ iff [by (18)]
h. $R \circ R' \leqslant R''$

The resulting class of extended monoids $\langle \mathbf{I}, \bullet, \mathbf{id}, \bot, \twoheadrightarrow\rangle$, i.e., monoids $\langle \mathbf{I}, \bullet, \mathbf{id}\rangle$ extended with $\bot$ and $\twoheadrightarrow$, will provide the right kind of models for DPropL.

2.4 Dynamic Predicate Logic (DPL) models as extended monoids

To make this even more concrete, we will anticipate our discussion of Incremental DPL here by showing how the space of Dynamic Predicate Logic (DPL; Groenendijk and Stokhof 1991) formula denotations together with the right kind operations forms an extended monoid. This will be a variation on the general binary-relation extended monoid we just discussed.

Recall that in DPL, the denotation of a formula φ, symbolized $[\![\varphi]\!]$, is a binary relation over variable assignments. The set of variable assignments $\mathcal{G}$ is the set of all functions g from the set of variables $\mathcal{V}$ to the domain of individuals $\mathbf{D}$, i.e., $\mathcal{G} = \mathbf{D}^{\mathcal{V}}$ for short.

The domain of the relation denoted by $[\![\varphi]\!]$, symbolized as $\mathbf{Dom}([\![\varphi]\!])$, is the set of all assignments g that can be input assignments for φ. That is, $\mathbf{Dom}([\![\varphi]\!])$ is the set of all assignments g relative to which φ is true. The range of this relation $\mathbf{Ran}([\![\varphi]\!])$ is the set of all assignments h that are output assignments after some input assignment or other is updated with φ. That is, $\mathbf{Ran}([\![\varphi]\!])$ is the set of all assignments that are the result of the update contributed by φ; subsequent formulas are interpreted relative to these assignments. We also define the image $g\mathcal{R}$ of an assignment g under the binary relation $\mathcal{R}$, which is the set of all assignments that we can get when we update g with $\mathcal{R}$.

(33) $\mathbf{Dom}(\mathcal{R}) := \{g : \exists h(g\mathcal{R}h)\}$

(34) $\mathbf{Ran}(\mathcal{R}) := \{h : \exists g(g\mathcal{R}h)\}$

(35) $g\mathcal{R} := \{h : g\mathcal{R}h\}$

In particular:

- the DPL denotations of formulas over a set of variable assignments $\mathcal{G}$, together with
- the denotation of dynamic conjunction ;, and
- the identity relation **id** over variable assignments, which is a test (in fact, the maximal test),

form a monoid.

The reason for this is that the DPL semantic values for formulas are binary relations over variable assignments, i.e., subsets of the Cartesian product $\mathcal{G} \times \mathcal{G}$, and the DPL denotation of dynamic conjunction is relation composition $\mathcal{R} \bullet \mathcal{R}'$, defined as shown below.

(36) $\mathcal{R} \bullet \mathcal{R}' = \{\langle g, h\rangle : \exists k(g\mathcal{R}k \wedge k\mathcal{R}'h)\}$

(37) $\mathbf{id} = \{\langle g, g\rangle : g \in \mathcal{G}\} = \{\langle g, h\rangle \in \mathcal{G} \times \mathcal{G} : g = h\}$

Just as we did when we discussed the more general binary-relation monoid in the previous section, we use the more intuitive infix notation $g\mathcal{R}h$ to indicate that the binary relation $\mathcal{R}$ relates g and h. The relation $\mathcal{R}\bullet\mathcal{R}'$ that is the result of composition will relate two assignments g and h iff there exists some intermediate assignment k such that $\mathcal{R}$ takes us from g to k and $\mathcal{R}'$ takes us from k to h.

We can easily check that $\langle \wp(\mathcal{G} \times \mathcal{G}), \bullet, \mathbf{id}\rangle$ (where $\wp$ is the powerset operator) is a monoid.

(38) $\langle \wp(\mathcal{G} \times \mathcal{G}), \bullet, \mathbf{id}\rangle$ is a monoid:

Ax1: for any binary relations $\mathcal{R}, \mathcal{R}' \subseteq \mathcal{G} \times \mathcal{G}$, their composition $\mathcal{R} \bullet \mathcal{R}'$ is also a subset of $\mathcal{G} \times \mathcal{G}$.

Ax2: $\langle g, g'\rangle \in (\mathcal{R} \bullet \mathcal{R}') \bullet \mathcal{R}''$ iff
$\exists h'(\ \exists h(\ g\mathcal{R}h \wedge h\mathcal{R}'h'\ ) \wedge h'\mathcal{R}''g'\ )$ iff
$\exists h, h'(\ g\mathcal{R}h \wedge h\mathcal{R}'h' \wedge h'\mathcal{R}''g'\ )$ iff
$\exists h(\ g\mathcal{R}h \wedge \exists h'(\ h\mathcal{R}'h' \wedge h'\mathcal{R}''g'\ )\ )$ iff
$\langle g, g'\rangle \in \mathcal{R} \bullet (\mathcal{R}' \bullet \mathcal{R}'')$.

Ax3: since the identity element **id** is the diagonal $\{\langle g, h\rangle \in \mathcal{G} \times \mathcal{G} : g = h\}$, we have that for any binary relation $\mathcal{R} \subseteq \mathcal{G} \times \mathcal{G}$,
$\langle g, g'\rangle \in \mathbf{id} \bullet \mathcal{R}$ iff
$\exists h(g\mathbf{id}h \wedge h\mathcal{R}g')$ iff
$\exists h(g = h \wedge h\mathcal{R}g')$ iff

$\langle g, g' \rangle \in \mathcal{R}$ iff
$\exists h(g\mathcal{R}h \wedge h = g')$ iff
$\exists h(g\mathcal{R}h \wedge h\mathbf{id}g')$ iff
$\langle g, g' \rangle \in \mathcal{R} \bullet \mathbf{id}$.

To complete our extended monoid construction, we only need to define $\perp$ and $\twoheadrightarrow$ relative to the DPL monoid. Following Visser (2002), we let $\perp$ be the empty binary relation and define $\twoheadrightarrow$ as the dynamic implication of Kamp (1981) (see also Heim 1982 and Groenendijk and Stokhof 1991).

(39) $\perp = \varnothing \subseteq \mathcal{G} \times \mathcal{G}$

(40) $\mathcal{R} \twoheadrightarrow \mathcal{R}' = \{\langle g, g \rangle : \text{for all } h \text{ such that } g\mathcal{R}h, \text{there is an } i \text{ such that } h\mathcal{R}'i\}$
$= \{\langle g, g \rangle : g\mathcal{R} \subseteq \mathbf{Dom}(\mathcal{R}')\}$

We can check that for any $\mathcal{R} \subseteq \mathcal{G} \times \mathcal{G}$, we have that $\perp \bullet \mathcal{R} = \mathcal{R} \bullet \perp = \perp$.[2]

We can also check that the above definition of dynamic implication satisfies the constraint $\mathcal{R} \bullet \mathcal{R}' \leqslant \mathcal{R}''$ iff $\mathcal{R} \leqslant \mathcal{R}' \twoheadrightarrow \mathcal{R}''$ – again, with the additional, and harmless, assumption that the binary relation $\mathcal{R}''$ is a test, i.e., $\mathcal{R}'' \subseteq \mathbf{id}$. The reasoning is the same as in the more general proof outlined in (32) above.

As Visser (2002, p. 112) notes, if the semantics of negation $\neg\varphi$ negation is defined in the expected way:

(41) $[\![\neg\varphi]\!] = \overline{[\![\varphi]\!]}, \quad \text{where } \overline{\mathcal{R}} := \mathcal{R} \twoheadrightarrow \perp \text{ (for any } \mathcal{R} \subseteq \mathcal{G} \times \mathcal{G})$

we derive the DRT / FCS / DPL interpretation for dynamic negation:[3]

(42) $\overline{\mathcal{R}} := \mathcal{R} \twoheadrightarrow \perp$

$\mathcal{R} \twoheadrightarrow \perp =$
$\{\langle g, g \rangle : g\mathcal{R} \subseteq \mathbf{Dom}(\perp)\} =$
$\{\langle g, g \rangle : g\mathcal{R} \subseteq \varnothing\} =$
$\{\langle g, g \rangle : g\mathcal{R} = \varnothing\} =$
$\{\langle g, g \rangle : g \notin \mathbf{Dom}(\mathcal{R})\} =$
$\{\langle g, g \rangle : g \in (\mathcal{G} \backslash \mathbf{Dom}(\mathcal{R}))\}$

Finally, if we define the always-true formula $\top$ in the expected way, i.e., as $\overline{\perp} = \perp \twoheadrightarrow \perp$ we see that $\top$ is the maximal test, i.e., $\top = \mathbf{id}$:

(43) $\top := \overline{\perp} \; (= \perp \twoheadrightarrow \perp)$

$\perp \twoheadrightarrow \perp =$
$\{\langle g, g \rangle : g\perp \subseteq \mathbf{Dom}(\perp)\} =$
$\{\langle g, g \rangle : \varnothing \subseteq \varnothing\} =$
$\{\langle g, g \rangle : g \in \mathcal{G}\} =$
$\mathbf{id}$

We will return to a discussion of the semantics of DPL after we introduce the incremental semantics for the simpler DPropL system.

[2]By the definition of relation composition $\bullet$, a pair of assignments $\langle g, h \rangle$ belongs to the relation $\perp \bullet \mathcal{R}$ iff $\exists k(g\perp k \wedge k\mathcal{R}h)$. Since $\perp$ is the empty set, it does not contain any pair of assignments, so there is no k such that $g\perp k$. Therefore, $\perp \bullet \mathcal{R}$ is the empty relation $\perp$. The same reasoning also establishes that $\mathcal{R} \bullet \perp$ is the empty relation $\perp$.

[3]In addition, anaphoric closure !, a.k.a. double negation, also receives the expected interpretation – its denotation is the **diag** operator introduced above. Recall that the operator ! is called anaphoric closure because for any formula φ, $!\varphi$ is a test, i.e., $[\![!\varphi]\!] = [\![\neg\neg\varphi]\!] = \mathbf{diag}([\![\varphi]\!]) \subseteq \mathbf{id}$.

It also follows that a doubly negated relation $\overline{\overline{\mathcal{R}}} = \mathbf{diag}(\mathcal{R})$ is not in general identical to the original relation $\mathcal{R}$ although their domains are the same – just as in DPL.

(1) $\mathbf{diag}(\mathcal{R}) := \overline{\overline{\mathcal{R}}} \; (= \mathbf{id} \twoheadrightarrow \mathcal{R})$

$\overline{\overline{\mathcal{R}}} =$
$\{\langle g, g \rangle : g \in (\mathcal{G} \backslash \mathbf{Dom}(\neg\mathcal{R}))\} =$
$\{\langle g, g \rangle : g \in (\mathcal{G} \backslash (\mathcal{G} \backslash \mathbf{Dom}(\mathcal{R})))\} =$
$\{\langle g, g \rangle : g \in \mathbf{Dom}(\mathcal{R})\}$

2.5 Sequence semantics for DPropL

We already introduced the kind of models $\mathfrak{M}$ that we will use in our recursive definition of the interpretation function for DPropL. They are extended monoids, i.e., monoids

- whose binary 'merge' operator $\bullet$ will provide the basic interpretation for conjunction

extended with

- a $\perp$ element that will provide the basic interpretation for *falsum*, and
- a binary operator $\twoheadrightarrow$ that will provide the basic interpretation for conditionals.

In this subsection, we first introduce the notion of semantic evaluation contexts $\mathfrak{c}$ that we will use in the recursive definition of the interpretation function $[\![\cdot]\!]^{\mathfrak{M},\mathfrak{c}}$ for DPropL. We then provide that definition and show how the resulting semantics for DPropL derives the intuitively correct truth conditions for our initial example in (3) in a fully incremental, left-to-right fashion.

As already indicated, the elements $i, j, \dots \in \mathbf{I}$ of the extended DPropL monoid $\langle \mathbf{I}, \bullet, \mathbf{id}, \perp, \twoheadrightarrow \rangle$ will be called *info(rmation) states*. But the reader should keep in mind that their intended interpretation – once we get to predicate logic – is as binary relations over variable assignments rather than simply variable assignments.

Given our basic info states $i, j, k, \dots$, we define an info history / info sequence σ as a finite non-empty sequence of info states. The set of all info histories $\mathbf{H_I}$ based on the set of info states $\mathbf{I}$ is therefore defined as:

(44) $\mathbf{H_I} = \bigcup_{n \in \mathbb{N}*} \mathbf{I}^n$

We will often 'unpack' an info sequence σ with n components ($n \geqslant 1$) as $\langle \sigma_1, \dots, \sigma_n \rangle$, where σ_1 is the first info state (e.g., info state i) in the history and σ_n is the last info state in the history. Assuming this kind of unpacking, we can rewrite the definition in (44) above in a more verbose but probably more readable way as follows:

(45) $\mathbf{H_I} = \{\langle \sigma_1, \dots, \sigma_n \rangle : n \geqslant 1 \wedge \sigma_1, \dots, \sigma_n \in \mathbf{I}\}$

A semantic evaluation context $\mathfrak{c}$ is an info sequence / info history $\sigma \in \mathbf{H_I}$. That is:

(46) The interpretation function for DPropL has the form $[\![\cdot]\!]^{\mathfrak{M},\sigma}$, where $\mathfrak{M}$ is an extended monoid over the set $\mathbf{I}$ and σ is an info history over the same set.

Using info histories as semantic evaluation contexts formally encodes the basic proposal we put forth above (following Vermeulen 1994): a fully incremental dynamic semantics requires a notion of memory / update history. Info sequences encode this idea in a direct way.

Now that we fully defined the parameters of the interpretation function $[\![\cdot]\!]$ (i.e., models $\mathfrak{M}$ and semantic evaluation contexts $\mathfrak{c}$), we can turn to its recursive definition. The exact form of the definition is determined by the way we conceptualize denotations of DPropL formulas.

At a very general level, the denotation / semantic value of a DPropL formula φ will have the same general form as the semantic value of a formula in DPL: a binary relation over semantic evaluation contexts $\mathfrak{c}$, i.e., a binary relation over the set of info histories $\mathbf{H_I}$.

Upon closer inspection, however, it turns out that we do not need the non-deterministic aspect of binary relations, and instead we can restrict ourselves to partial functions. That is, we can restrict ourselves to binary relations satisfying the extra condition that any element in their domain (i.e., any element for which they are defined) is mapped to exactly one element in their range.

The reason for this is simple. In DPL, we need non-deterministic binary relations over evaluation contexts as formula denotations because DPL evaluation contexts are variable assignments, so existential quantification has to be modeled as the non-deterministic introduction of a witness, i.e., the non-deterministic update of the input evaluation context (this is exactly the same in classical, static predicate logic).

In DPropL, and as we will soon see, in Incremental DPL, evaluation contexts are sequences of info states. What matters here is not that they are sequences, but that the info states $i, j, k, \dots$ inside the sequences are 'fully-fledged' binary relations over variable assignments and not merely variable assignments. The non-determinism needed to properly interpret existential quantification is therefore encapsulated inside info states. So there is no need to be non-deterministic at the higher level of info sequences, which means that formula denotations, which manipulate / update info sequences, can be simpler, i.e., deterministic.

In sum:

(47) The denotation of any DPropL formula φ is a partial function over info histories: $[\![\varphi]\!] : \mathbf{H_I} \to \mathbf{H_I}$.
That is, for an info sequence $\sigma \in \mathbf{H_I}$, if $\sigma \in \mathbf{Dom}([\![\varphi]\!])$, we have that: $[\![\varphi]\!](\sigma) = \sigma'$, for some $\sigma' \in \mathbf{H_I}$.

Following Vermeulen (1994), we will use the postfix notation $\sigma[\![\varphi]\!]$ instead of the usual $[\![\varphi]\!](\sigma)$ because it is more intuitive given our purposes:

(48) In postfix notation: $\sigma[\![\varphi]\!] = \sigma'$, i.e., interpreting a formula φ relative to an input info history σ means that we update the info history σ with the semantic value $[\![\varphi]\!]$, and the (deterministic) result is a new info history σ'.

The intuition motivating the postfix notation generalizes to sequences of updates, i.e., to partial function composition, which is simply relation composition $\bullet$ restricted to partial functions:

(49) We say that $\sigma([\![\varphi]\!] \bullet [\![\psi]\!]) = \sigma'$, or even more simply $\sigma[\![\varphi]\!][\![\psi]\!] = \sigma'$, iff σ' is the result of (deterministically) updating the info history σ with $[\![\varphi]\!]$ first, and then with $[\![\psi]\!]$.
That is, $\sigma[\![\varphi]\!][\![\psi]\!] = (\sigma[\![\varphi]\!])[\![\psi]\!] = \sigma'$.

The postfix notation for partial functions is more intuitive, and more in-line with the infix notation we use for binary relations.

The last issue we need to address before providing the recursive definition of the interpretation function $[\![\cdot]\!]^{\mathfrak{M},c}$ for DPropL concerns the denotations of atomic formulas. For each atomic text / formula $p \in A$, we assume that an info state $i_p \in \mathbf{I}$ exists that encodes the information that p. These atomic formulas are unanalyzable in DPropL but in DPL, they are tests contributed by lexical relations like $\textsc{sleep}(x)$ or $\textsc{like}(x, y)$. The corresponding info states encode that x sleeps and that x likes y respectively.

(50) $i_{\textsc{sleep}(x)} = \{\langle g, g\rangle : g(x) \in [\![\textsc{sleep}]\!]\}$

(51) $i_{\textsc{like}(x,y)} = \{\langle g, g\rangle : \langle g(x), g(y)\rangle \in [\![\textsc{like}]\!]\}$

Note that $i_{\textsc{sleep}(x)}$ and $i_{\textsc{like}(x,y)}$ are just the regular DPL semantic values for the atomic formulas $\textsc{sleep}(x)$ and $\textsc{like}(x, y)$.

We are now ready to define a fully incremental semantics for DPropL. The recursive definition of the interpretation function $[\![\cdot]\!]^{\mathfrak{M}}$ is provided below. The model $\mathfrak{M}$ is left implicit throughout.

(52) **DPropL semantics** (Vermeulen 1994)

a. Atomic formulas:
   i. $\sigma[\![\bot]\!] = \langle \sigma_1, \ldots, \sigma_{n-1}, \sigma_n \bullet \bot\rangle$
   ii. $\sigma[\![p]\!] = \langle \sigma_1, \ldots, \sigma_{n-1}, \sigma_n \bullet i_p\rangle$
   iii. $\sigma[\![\mathbf{if}]\!] = \langle \sigma_1, \ldots, \sigma_{n-1}, \sigma_n, \top\rangle$
   iv. $\sigma[\![\mathbf{then}]\!] = \langle \sigma_1, \ldots, \sigma_{n-1}, \sigma_n, \top\rangle$
   v. $\sigma[\![\mathbf{end}]\!] = \langle \sigma_1, \ldots, \sigma_{n-2} \bullet (\sigma_{n-1} \twoheadrightarrow \sigma_n)\rangle$

b. Conjunction:
   i. $\sigma[\![\varphi; \psi]\!] = \sigma([\![\varphi]\!] \bullet [\![\psi]\!]) = (\sigma[\![\varphi]\!])[\![\psi]\!]$

c. Truth:
   i. A formula φ is true in model $\mathfrak{M}$ relative to an input info state $i \in \mathbf{I}$ iff $\langle i\rangle[\![\varphi]\!] = \langle i\rangle$.
   ii. A formula φ is true in model $\mathfrak{M}$ *simpliciter* iff it is true relative to the input info state $\top$, i.e., iff $\langle\top\rangle[\![\varphi]\!] = \langle\top\rangle$.

The definition in (52) might be easier to follow if we rewrite it in a more familiar format:

(53) **DPropL semantics** rewritten in a 'classical' format

a. Atomic formulas:
   i. $[\![\bot]\!]^\sigma = \langle \sigma_1, \ldots, \sigma_{n-1}, \sigma_n \bullet \bot \rangle$
   ii. $[\![p]\!]^\sigma = \langle \sigma_1, \ldots, \sigma_{n-1}, \sigma_n \bullet i_p \rangle$
   iii. $[\![\mathbf{if}]\!]^\sigma = \langle \sigma_1, \ldots, \sigma_{n-1}, \sigma_n, \top \rangle$
   iv. $[\![\mathbf{then}]\!]^\sigma = \langle \sigma_1, \ldots, \sigma_{n-1}, \sigma_n, \top \rangle$
   v. $[\![\mathbf{end}]\!]^\sigma = \langle \sigma_1, \ldots, \sigma_{n-2} \bullet (\sigma_{n-1} \twoheadrightarrow \sigma_n) \rangle$

b. Conjunction:
   i. $[\![\varphi; \psi]\!]^\sigma = [\![\psi]\!]^{\sigma'}$, where $\sigma' = [\![\varphi]\!]^\sigma$

c. Truth:
   i. A formula φ is true in model $\mathfrak{M}$ relative to an input info state $i \in \mathbf{I}$ iff $[\![\varphi]\!]^{\langle i \rangle} = \langle i \rangle$.
   ii. A formula φ is true in model $\mathfrak{M}$ *simpliciter* iff it is true relative to the input info state $\top$, i.e., iff $[\![\varphi]\!]^{\langle \top \rangle} = \langle \top \rangle$.

Let us look at some examples. An atomic formula p (when we move to predicate logic, this could be $\text{SLEEP}(x)$, or $\text{LIKE}(x, y)$, for example) is true in a model $\mathfrak{M}$ *simpliciter* iff:

(54) p is true in model $\mathfrak{M}$ iff [by the definition of truth]
$[\![p]\!]^{\langle \top \rangle} = \langle \top \rangle$ iff [by the clause for atomic formulas]
$\langle \top \bullet i_p \rangle = \langle \top \rangle$ iff [by the properties of $\bullet$ and $\top = \mathbf{id}$]
$\langle i_p \rangle = \langle \top \rangle$ iff $i_p = \top$

This is as it should be: a formula p is true iff the info state i_p that corresponds to it is in fact $\top$.
The official version of DPropL semantics in (52) enables us to express this even more concisely:

(55) p is true in model $\mathfrak{M}$ iff $\langle \top \rangle = \langle \top \rangle [\![p]\!] = \langle \top \bullet i_p \rangle = \langle i_p \rangle$

The definition in (52) gives us the right result for the text in (4) above. The reason we derive the correct truth conditions for the text, and in particular for the conditional, while preserving a fully incremental left-to-right interpretation procedure is due to the way the interpretations of **if**, **then**, and **end** work together.

In particular, **if** and **then** do not make any truth-conditional contribution; they simply open two new slot in the info history relative to which the conditional is interpreted. Then, the contentful information contributed by the antecedent and consequent of the conditional is stored in these two slots. This two pieces of contentful information are merged in the appropriate way, i.e., by means of the implication operator $\twoheadrightarrow$, only when we reach **end**. This is shown in full detail below:

(56) $\langle \top \rangle [\![p; \mathbf{if}; q; \mathbf{then}; r; \mathbf{end}]\!] =$
$\langle \top \bullet i_p \rangle [\![\mathbf{if}; q; \mathbf{then}; r; \mathbf{end}]\!] =$
$\langle i_p \rangle [\![\mathbf{if}; q; \mathbf{then}; r; \mathbf{end}]\!] =$
$\langle i_p, \top \rangle [\![q; \mathbf{then}; r; \mathbf{end}]\!] =$
$\langle i_p, \top \bullet i_q \rangle [\![\mathbf{then}; r; \mathbf{end}]\!] =$
$\langle i_p, i_q \rangle [\![\mathbf{then}; r; \mathbf{end}]\!] =$
$\langle i_p, i_q, \top \rangle [\![r; \mathbf{end}]\!] =$
$\langle i_p, i_q, \top \bullet i_r \rangle [\![\mathbf{end}]\!] =$
$\langle i_p, i_q, i_r \rangle [\![\mathbf{end}]\!] =$
$\langle i_p \bullet (i_q \twoheadrightarrow i_r) \rangle$

We can actually rewrite this derivation in a way that makes its similarities to typical semantic derivations in classical first-oder logic much more obvious:

(57) $[\![p; \mathbf{if}; q; \mathbf{then}; r; \mathbf{end}]\!]^{\langle \top \rangle} =$
$[\![\mathbf{if}; q; \mathbf{then}; r; \mathbf{end}]\!]^{\langle \top \bullet i_p \rangle} =$
$[\![\mathbf{if}; q; \mathbf{then}; r; \mathbf{end}]\!]^{\langle i_p \rangle} =$
$[\![q; \mathbf{then}; r; \mathbf{end}]\!]^{\langle i_p, \top \rangle} =$

$$\begin{aligned}
&[\![\mathbf{then}; r; \mathbf{end}]\!]^{\langle i_p, \top \bullet i_q \rangle} = \\
&[\![\mathbf{then}; r; \mathbf{end}]\!]^{\langle i_p, i_q \rangle} = \\
&[\![r; \mathbf{end}]\!]^{\langle i_p, i_q, \top \rangle} = \\
&[\![\mathbf{end}]\!]^{\langle i_p, i_q, \top \bullet i_r \rangle} = \\
&[\![\mathbf{end}]\!]^{\langle i_p, i_q, i_r \rangle} = \\
&\langle i_p \bullet (i_q \twoheadrightarrow i_r) \rangle
\end{aligned}$$

Depending on what the monoid elements i_p, i_q, and i_r happen to be, the element $i_p \bullet (i_q \twoheadrightarrow i_r)$ could be $\top$, in which case our sentence will be true relative to our model $\mathfrak{M}$, or not, in which case our sentence will be false.

To summarize, the info-history-based semantics for DPropL:

- is fully incremental and associative because the semantic values / updates contributed by formulas are partial functions, and conjunction / concatenation is interpreted as function composition, which is by definition associative,

- and is able to incrementally derive the correct truth conditions for non-associative operators like conditionals because its evaluation contexts have memory, i.e., are able to store update histories

2.5.1 Problems with the sequence semantics for DPropL

But the semantics in (52) is not completely satisfactory because the meanings of *if* and *then* are identical. This incorrectly predicts that *then* could introduce the antecedent of a conditional, while *if* could introduce the consequent, i.e., that $\mathbf{if}; p; \mathbf{then}; q; \mathbf{end}$ is equivalent to $\mathbf{then}; p; \mathbf{if}; q; \mathbf{end}$.

Conditionals with a sentence-final antecedent exist, e.g., *I will buy you a toy if you behave*,[4] but that does not make them equivalent to the conditionals in which the antecedent and the consequent are exchanged, e.g., *If I buy you a toy, you will behave*.

3 Incremental semantics for DPropL with trees

The sequence semantics for DPropL is incremental and associative, but it sacrifices the difference in truth-conditional / update status between the antecedent and the consequent of conditionals. Intuitively, we want *if* to mark the 'beginning' of the conditional and *then* to mark the 'elaboration'. In the sequence semantics for DPropL in (52) above, both *if* and *then* simply marked that a new info state slot should be opened at the end of the current info history.

One way to make finer-grained, more structured distinctions between info states in a history is to add more structure to our notion of history. This is what Vermeulen (1994) basically proposes. We should think of a conditional not simply as an element of the current info history but as a kind of embedded / subordinate history. In Vermeulen's terms, we should think of info histories not simply as sequences, but as trees.

To improve readability, we will modify many aspects of the original formulation in Vermeulen (1994): notational conventions, various definitional details, tree representations etc., To avoid cluttering the main points, we will not indicate all these differences in the text; the reader should consult Vermeulen (1994, pp. 250-253) for the original formulation.

3.1 From sequences to trees

Switching from sequences to trees is simply adding more structure to sequences. We can think of sequences as trees with only one level of embedding or alternatively, we can think of trees as recursively-specified sequences, i.e., as sequences that can contain other sequences.

To see this, consider two simple sequences of info states $\langle i \rangle$ and $\langle i, j \rangle$. We can think of these as trees with one level of embedding: the mother node is introduced by the angular brackets $\langle\rangle$ and the info states inside the angular brackets specify the daughter nodes from left to right, as shown below.

[4] Although adding *then* at the beginning of the matrix clause is infelicitous in these cases.

(58) $\langle i \rangle :=$
$$\begin{tikzcd}\Diamond \arrow[d] \\ i\end{tikzcd}$$
$\langle i, j \rangle :=$
$$\begin{tikzcd} & \Diamond \arrow[dl] \arrow[dr] & \\ i & & j\end{tikzcd}$$

In general, sequences that contain other sequences can be similarly represented by means of trees. Some examples are provided below.

(59) $\langle\langle i \rangle\rangle :=$
$$\begin{tikzcd}\Diamond \arrow[d] \\ \Diamond \arrow[d] \\ i\end{tikzcd}$$
$\langle i, \langle j \rangle\rangle :=$
$$\begin{tikzcd} & \Diamond \arrow[dl] \arrow[dr] & \\ i & & \Diamond \arrow[d] \\ & & j\end{tikzcd}$$
$\langle i, \langle j \rangle, \langle k \rangle\rangle :=$
$$\begin{tikzcd} & \Diamond \arrow[dl] \arrow[d] \arrow[dr] & \\ i & \Diamond \arrow[d] & \Diamond \arrow[d] \\ & j & k\end{tikzcd}$$

In general, we might need info trees, i.e., recursive info-state sequences, of arbitrary complexity. But for the limited purposes of this paper, we simply need trees with unary, binary and ternary branching nodes of a particular structure, defined below.

(60) The set $\mathbf{H_I}$ of info histories over a set of info states $\mathbf{I}$ is the smallest set such that:

a. for any info state $i \in \mathbf{I}$, $\langle i \rangle \in \mathbf{H_I}$; in tree format:
$$\begin{tikzcd}\Diamond \arrow[d] \\ i\end{tikzcd} \in \mathbf{H_I};$$

b. if $i \in \mathbf{I}$ and $\sigma \in \mathbf{H_I}$, then $\langle i, \sigma \rangle \in \mathbf{H_I}$; in tree format:
$$\begin{tikzcd} & \Diamond \arrow[dl] \arrow[dr] & \\ i & & \sigma\end{tikzcd} \in \mathbf{H_I};$$

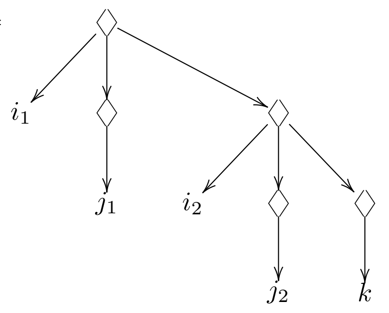
c. if $i, j \in \mathbf{I}$ and $\sigma \in \mathbf{H_I}$, then $\langle i, \langle j \rangle, \sigma \rangle \in \mathbf{H_I}$; in tree format:
$$\begin{tikzcd} & \Diamond \arrow[dl] \arrow[d] \arrow[dr] & \\ i & \Diamond \arrow[d] & \sigma \\ & j & \end{tikzcd} \in \mathbf{H_I};$$

Several examples of info histories licensed by the definition in (60) above are provided below, in both sequence and tree format.

(61) a. $\langle i, \langle j, \langle k \rangle\rangle\rangle =$
$$\begin{tikzcd} & \Diamond \arrow[dl] \arrow[dr] & & \\ i & & \Diamond \arrow[dl] \arrow[dr] & \\ & j & & \Diamond \arrow[d] \\ & & & k\end{tikzcd}$$

b. $\langle i, \langle j \rangle, \langle k_1, \langle k_2 \rangle\rangle\rangle =$
$$\begin{tikzcd} & \Diamond \arrow[dl] \arrow[d] \arrow[drr] & & & \\ i & \Diamond \arrow[d] & & \Diamond \arrow[dl] \arrow[dr] & \\ & j & k_1 & & \Diamond \arrow[d] \\ & & & & k_2\end{tikzcd}$$

c. $\langle i_1, \langle j_1 \rangle, \langle i_2, \langle j_2 \rangle, \langle k \rangle\rangle\rangle =$



3.2 Ways of updating tree-like info histories

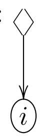
There are three basic types of trees that are relevant for our incremental semantics:

- trees that are updated with a matrix clause or with a conjunction (either the first or the second conjunct),
- trees that are updated with the antecedent of a conditional, and
- trees that are updated with the consequent of a conditional.

Typical examples of such trees are provided below. The info state in the tree that is targeted by the update is enclosed in a circle.

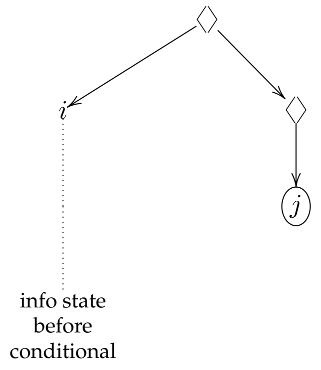
(62) Types of updates:

a. Update with a main clause or conjunct:



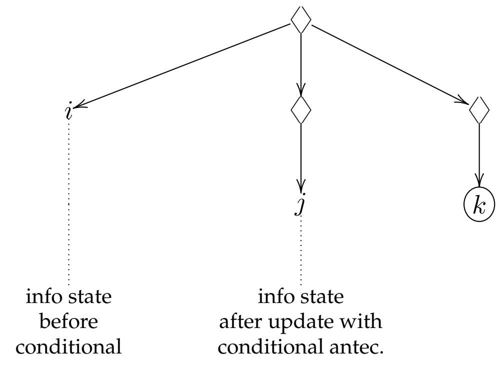
– or in sequence format: $\langle \textcircled{i} \rangle$;

b. Update with a conditional antecedent:



– or in sequence format: $\langle i, \langle \textcircled{j} \rangle\rangle$;

c. Update with a conditional consequent:


i
j
k
info state before conditional
info state after update with conditional antec.

– or in sequence format: $\langle i, \langle j \rangle, \langle \textcircled{k} \rangle\rangle$.

As the examples above show, different updates target different positions in the current tree-like info history. This is how we are able to distinguish between updated contributed by *if*, i.e., updates with a conditional antecedent, and updates contributed by *then*, i.e., updates with a conditional consequent.

The definition of the interpretation function will simply incorporate these informal ideas. To make this definition more readable, we need to introduce a notational convention:

(63) a. An info history σ whose final subtree is ρ, i.e., such that $\mathbf{ftree}(\sigma) = \rho$, is symbolized as $\sigma\{\rho\}$.

b. The final subtree of an info history σ is recursively defined as follows:

i. $\mathbf{ftree}(\langle i \rangle) = \langle i \rangle$, for any $i \in \mathbf{I}$;

ii. $\mathbf{ftree}(\langle i, \tau \rangle) = \begin{cases} \langle i, \tau \rangle & \text{if } \tau = \langle j \rangle \text{, for some } j \in \mathbf{I} \\ \mathbf{ftree}(\tau) & \text{otherwise} \end{cases}$

iii. $\mathbf{ftree}(\langle i, \langle j \rangle, \tau \rangle) = \begin{cases} \langle i, \langle j \rangle, \tau \rangle & \text{if } \tau = \langle k \rangle \text{, for some } k \in \mathbf{I} \\ \mathbf{ftree}(\tau) & \text{otherwise} \end{cases}$

c. If we have an info history of the form $\sigma\{\rho\}$, i.e., an info history σ whose final subtree is ρ, and we replace the final subtree ρ with an arbitrary tree ρ', we abbreviate the resulting info history by doubling the curly braces around the replacement tree ρ', as in $\sigma\{\{\rho'\}\}$.

Note that given the definition of info histories in (60) above, and of the **ftree** function in (63b) above, the final subtree of any info history σ can have only one of the following three forms:

i. $\langle i \rangle$ ('matrix' level only);

ii. $\langle i, \langle j \rangle \rangle$ ('matrix' and conditional antecedent);

iii. $\langle i, \langle j \rangle, \langle k \rangle \rangle$ ('matrix', conditional antecedent, and conditional consequent).

3.3 Semantics for DPropL with tree-Like info histories

We can now provide the final definition of the interpretation function $[\![\cdot]\!]^{\mathfrak{M}}$ for DPropL. The model superscript is omitted throughout.

(64) **DPropL semantics – final version** (Vermeulen 1994)

a. Atomic formulas:

i. $[\![\bot]\!] = \begin{bmatrix} \sigma\{\langle i \rangle\} & \mapsto & \sigma\{\{\langle i \bullet \bot \rangle\}\} \\ \sigma\{\langle i, \langle j \rangle \rangle\} & \mapsto & \sigma\{\{\langle i, \langle j \bullet \bot \rangle \rangle\}\} \\ \sigma\{\langle i, \langle j \rangle, \langle k \rangle \rangle\} & \mapsto & \sigma\{\{\langle i, \langle j \rangle, \langle k \bullet \bot \rangle \rangle\}\} \end{bmatrix}$

That is:
$\sigma\{\langle i \rangle\}[\![\bot]\!] = \sigma\{\{\langle i \bullet \bot \rangle\}\}$ – we update at 'matrix' level,
$\sigma\{\langle i, \langle j \rangle \rangle\}[\![\bot]\!] = \sigma\{\{\langle i, \langle j \bullet \bot \rangle \rangle\}\}$ – we update inside a conditional antecedent, and
$\sigma\{\langle i, \langle j \rangle, \langle k \rangle \rangle\}[\![\bot]\!] = \sigma\{\{\langle i, \langle j \rangle, \langle k \bullet \bot \rangle \rangle\}\}$ – we update inside a conditional consequent.

ii. $[\![p]\!] = \begin{bmatrix} \sigma\{\langle i \rangle\} & \mapsto & \sigma\{\{\langle i \bullet i_p \rangle\}\} \\ \sigma\{\langle i, \langle j \rangle \rangle\} & \mapsto & \sigma\{\{\langle i, \langle j \bullet i_p \rangle \rangle\}\} \\ \sigma\{\langle i, \langle j \rangle, \langle k \rangle \rangle\} & \mapsto & \sigma\{\{\langle i, \langle j \rangle, \langle k \bullet i_p \rangle \rangle\}\} \end{bmatrix}$

That is:
$\sigma\{\langle i \rangle\}[\![p]\!] = \sigma\{\{\langle i \bullet i_p \rangle\}\}$ – we update at 'matrix' level,
$\sigma\{\langle i, \langle j \rangle \rangle\}[\![p]\!] = \sigma\{\{\langle i, \langle j \bullet i_p \rangle \rangle\}\}$ – we update inside a conditional antecedent, and
$\sigma\{\langle i, \langle j \rangle, \langle k \rangle \rangle\}[\![p]\!] = \sigma\{\{\langle i, \langle j \rangle, \langle k \bullet i_p \rangle \rangle\}\}$ – we update inside a conditional consequent.

iii. $[\![\mathbf{if}]\!] = \begin{bmatrix} \sigma\{\langle i \rangle\} & \mapsto & \sigma\{\{\langle i, \langle \top \rangle \rangle\}\} \\ \sigma\{\langle i, \langle j \rangle \rangle\} & \mapsto & \sigma\{\{\langle i, \langle j, \langle \top \rangle \rangle \rangle\}\} \\ \sigma\{\langle i, \langle j \rangle, \langle k \rangle \rangle\} & \mapsto & \sigma\{\{\langle i, \langle j \rangle, \langle k, \langle \top \rangle \rangle \rangle\}\} \end{bmatrix}$

That is:
$\sigma\{\langle i\rangle\}[\![\mathbf{if}]\!] = \sigma\{\{\langle i, \langle\top\rangle\rangle\}\}$ – we start a conditional at 'matrix' level,
$\sigma\{\langle i, \langle j\rangle\rangle\}[\![\mathbf{if}]\!] = \sigma\{\{\langle i, \langle j, \langle\top\rangle\rangle\rangle\}\}$ – we start a conditional inside another conditional antecedent, and
$\sigma\{\langle i, \langle j\rangle, \langle k\rangle\rangle\}[\![\mathbf{if}]\!] = \sigma\{\{\langle i, \langle j\rangle, \langle k, \langle\top\rangle\rangle\rangle\}\}$ – we start a conditional inside another conditional consequent.

iv. $[\![\mathbf{then}]\!] = \left[\ \sigma\{\langle i, \langle j\rangle\rangle\} \quad\mapsto\quad \sigma\{\{\langle i, \langle j\rangle, \langle\top\rangle\rangle\}\}\ \right]$

That is:
$\sigma\{\langle i, \langle j\rangle\rangle\}[\![\mathbf{then}]\!] = \sigma\{\{\langle i, \langle j\rangle, \langle\top\rangle\rangle\}\}$ – if we have a conditional antecedent, we start a conditional consequent; otherwise undefined.

v. $[\![\mathbf{end}]\!] = \left[\ \sigma\{\langle i, \langle j\rangle, \langle k\rangle\rangle\} \quad\mapsto\quad \sigma\{\{\langle i \bullet (j \twoheadrightarrow k)\rangle\}\}\ \right]$

That is:
$\sigma\{\langle i, \langle j\rangle, \langle k\rangle\rangle\}[\![\mathbf{end}]\!] = \sigma\{\{\langle i \bullet (j \twoheadrightarrow k)\rangle\}\}$ – if we have both a conditional antecedent and a conditional consequent, we form a conditional and merge it with the 'matrix' info state; otherwise undefined.

b. Conjunction:
   i. $[\![\varphi;\psi]\!] = \left[\ \sigma \quad\mapsto\quad (\sigma[\![\varphi]\!])[\![\psi]\!]\ \right]$

That is:
$\sigma[\![\varphi;\psi]\!] = (\sigma[\![\varphi]\!])[\![\psi]\!]$

c. Truth:
   i. A formula φ is true in model $\mathfrak{M}$ relative to an input info state $i \in \mathbf{I}$ iff $\langle i\rangle[\![\varphi]\!] = \langle i\rangle$.
   ii. A formula φ is true in model $\mathfrak{M}$ *simpliciter* iff it is true relative to the input info state $\top$, i.e., iff $\langle\top\rangle[\![\varphi]\!] = \langle\top\rangle$.

Just like the sequence-based definition, this definition gives us the right result for the text in (4) above. But it also preserves the difference in truth-conditional / update status between the antecedent and the consequent of conditionals: the $[\![\mathbf{then}]\!]$ update is a partial function defined only on info histories in which the update contributed by a conditional antecedent is already present.

(65) $\langle\top\rangle[\![p;\mathbf{if};q;\mathbf{then};r;\mathbf{end}]\!] =$
$\langle\top \bullet i_p\rangle[\![\mathbf{if};q;\mathbf{then};r;\mathbf{end}]\!] =$
$\langle i_p, \langle\top\rangle\rangle[\![q;\mathbf{then};r;\mathbf{end}]\!] =$
$\langle i_p, \langle\top \bullet i_q\rangle\rangle[\![\mathbf{then};r;\mathbf{end}]\!] =$
$\langle i_p, \langle i_q\rangle, \langle\top\rangle\rangle[\![r;\mathbf{end}]\!] =$
$\langle i_p, \langle i_q\rangle, \langle\top \bullet i_r\rangle\rangle[\![\mathbf{end}]\!] =$
$\langle i_p \bullet (i_q \twoheadrightarrow i_r)\rangle$

4 The almost standard semantics for Dynamic Predicate Logic (DPL)

(66) **DPL semantics – almost standard version based on Groenendijk and Stokhof (1991)**

a. Atomic formulas:
   i. $[\![\bot]\!]^{\mathsf{DPL}} = \{\langle g, g\rangle : g \neq g\} = \varnothing$

ii. If π is an n-place predicate and $x_1, \ldots, x_n$ are variables, then $[\![\pi(x_1, \ldots, x_n)]\!]^{\mathsf{DPL}} = \{\langle g, g\rangle : \langle g(x_1), \ldots, g(x_n)\rangle \in [\![\pi]\!]^{\mathsf{DPL}}\}$.

iii. If x and y are terms, then $[\![x = y]\!]^{\mathsf{DPL}} = \{\langle g, g\rangle : g(x) = g(y)\}$.

b. Formulas (sentential connectives):

i. $[\![\varphi;\ \psi]\!]^{\mathsf{DPL}} = [\![\varphi]\!]^{\mathsf{DPL}} \bullet [\![\psi]\!]^{\mathsf{DPL}}$

ii. $[\![\varphi \to \psi]\!]^{\mathsf{DPL}} = [\![\varphi]\!]^{\mathsf{DPL}} \twoheadrightarrow [\![\psi]\!]^{\mathsf{DPL}}$
$\qquad = \{\langle g, g\rangle : g[\![\varphi]\!]^{\mathsf{DPL}} \subseteq \mathbf{Dom}([\![\psi]\!]^{\mathsf{DPL}})\}$

c. Formulas (random assignment):

i. $[\![[\upsilon]]\!]^{\mathsf{DPL}} = \{\langle g, h\rangle : g[\upsilon]h\}$,
where $g[\upsilon]h := \forall \upsilon' \neq \upsilon(g(\upsilon') = h(\upsilon'))$, i.e.,
g differs from h at most with respect to the value of υ.

d. Abbreviations:

i. $[\![\sim \varphi]\!]^{\mathsf{DPL}} = [\![\varphi \to \bot]\!]^{\mathsf{DPL}}$
$\qquad = \{\langle g, g\rangle : g \notin \mathbf{Dom}([\![\varphi]\!])\}$

ii. $[\![\exists\upsilon\varphi]\!]^{\mathsf{DPL}} = [\![[\upsilon];\ \varphi]\!]^{\mathsf{DPL}} = [\![[\upsilon]]\!]^{\mathsf{DPL}} \bullet [\![\varphi]\!]^{\mathsf{DPL}}$
$\qquad = \{\langle g, g\rangle : g[\![[\upsilon]]\!]^{\mathsf{DPL}} \cap \mathbf{Dom}([\![\varphi]\!]^{\mathsf{DPL}}) \neq \varnothing\}$

iii. $[\![\forall\upsilon\varphi]\!]^{\mathsf{DPL}} = [\![[\upsilon] \to \varphi]\!]^{\mathsf{DPL}} = [\![[\upsilon]]\!]^{\mathsf{DPL}} \twoheadrightarrow [\![\varphi]\!]^{\mathsf{DPL}}$
$\qquad = \{\langle g, g\rangle : g[\![[\upsilon]]\!]^{\mathsf{DPL}} \subseteq \mathbf{Dom}([\![\varphi]\!]^{\mathsf{DPL}})\}$

e. Truth:

i. A formula φ is true in model $\mathfrak{M}$ relative to an input assignment g iff there is an output assignment h such that $\langle g, h\rangle \in [\![\varphi]\!]^{\mathsf{DPL}}$, i.e., iff $g \in \mathbf{Dom}([\![\varphi]\!]^{\mathsf{DPL}})$.

5 Incremental DPL (IDPL): tree semantics for DPL

It is probably clear by now that the 'memory'-based solution proposed by Vermeulen (1994) for DPropL will extend to DPL, especially given the format of DPL semantics presented above and the particular formulation of DPropL semantics we introduced.

More specifically, the semantics of the existential quantifier $\exists x(\varphi)$, i.e., $[x]; \varphi$, is automatically associative because it simply involves dynamically conjoining two formulas. And as far as the non-associative semantics for the universal quantifier $\forall x(\varphi)$ is concerned, Vermeulen's solution for implication generalizes as soon as we observe that the universal is simply dynamic implication $[x] \to \varphi$, or using the Vermeulen (1994) syntax: **if**; $[x]$; **then**; φ; **end**.

(67) **Atomic info states.** We associate an info state, i.e., a binary relation over assignments, with every atomic formula. These info states are just the standard DPL denotations of the corresponding formulas:

a. $\mathcal{R}_{\pi(x_1,\ldots,x_n)} = \{\langle g, g\rangle : \langle g(x_1), \ldots, g(x_n)\rangle \in [\![\pi]\!]\}$

b. $\mathcal{R}_{x=y} = \{\langle g, g\rangle : g(x) = g(y)\}$

c. $\mathcal{R}_{[\upsilon]} = \{\langle g, h\rangle : g[\upsilon]h\}$

d. If we have a set of n atomic formulas $\varphi_1, \ldots, \varphi_n$, we will abbreviate the merge of the corresponding info states as $\mathcal{R}_{\varphi_1 \bullet \cdots \bullet \varphi_n} := \mathcal{R}_{\varphi_1} \bullet \cdots \bullet \mathcal{R}_{\varphi_n}$.

(68) **Incremental DPL (IDPL) semantics**

a. Atomic formulas:

i. $$[\![\bot]\!] = \begin{bmatrix} \sigma\{\langle \mathcal{R}\rangle\} & \mapsto & \sigma\{\{\langle \mathcal{R} \bullet \bot\rangle\}\} \\ \sigma\{\langle \mathcal{R}, \langle \mathcal{R}'\rangle\rangle\} & \mapsto & \sigma\{\{\langle \mathcal{R}, \langle \mathcal{R}' \bullet \bot\rangle\rangle\}\} \\ \sigma\{\langle \mathcal{R}, \langle \mathcal{R}'\rangle, \langle \mathcal{R}''\rangle\rangle\} & \mapsto & \sigma\{\{\langle \mathcal{R}, \langle \mathcal{R}'\rangle, \langle \mathcal{R}'' \bullet \bot\rangle\rangle\}\} \end{bmatrix}$$

ii. $$[\![\pi(x_1, \ldots, x_n)]\!] = \begin{bmatrix} \sigma\{\langle \mathcal{R}\rangle\} & \mapsto & \sigma\{\{\langle \mathcal{R} \bullet \mathcal{R}_{\pi(x_1,\ldots,x_n)}\rangle\}\} \\ \sigma\{\langle \mathcal{R}, \langle \mathcal{R}'\rangle\rangle\} & \mapsto & \sigma\{\{\langle \mathcal{R}, \langle \mathcal{R}' \bullet \mathcal{R}_{\pi(x_1,\ldots,x_n)}\rangle\rangle\}\} \\ \sigma\{\langle \mathcal{R}, \langle \mathcal{R}'\rangle, \langle \mathcal{R}''\rangle\rangle\} & \mapsto & \sigma\{\{\langle \mathcal{R}, \langle \mathcal{R}'\rangle, \langle \mathcal{R}'' \bullet \mathcal{R}_{\pi(x_1,\ldots,x_n)}\rangle\rangle\}\} \end{bmatrix}$$

iii. $[\![x = y]\!] = \left[\begin{array}{lll} \sigma\{\langle\mathcal{R}\rangle\} & \mapsto & \sigma\{\{\langle\mathcal{R} \bullet \mathcal{R}_{x=y}\rangle\}\} \\ \sigma\{\langle\mathcal{R}, \langle\mathcal{R}'\rangle\rangle\} & \mapsto & \sigma\{\{\langle\mathcal{R}, \langle\mathcal{R}' \bullet \mathcal{R}_{x=y}\rangle\rangle\}\} \\ \sigma\{\langle\mathcal{R}, \langle\mathcal{R}'\rangle, \langle\mathcal{R}''\rangle\rangle\} & \mapsto & \sigma\{\{\langle\mathcal{R}, \langle\mathcal{R}'\rangle, \langle\mathcal{R}'' \bullet \mathcal{R}_{x=y}\rangle\rangle\}\} \end{array}\right]$

iv. $[\![[v]]\!] = \left[\begin{array}{lll} \sigma\{\langle\mathcal{R}\rangle\} & \mapsto & \sigma\{\{\langle\mathcal{R} \bullet \mathcal{R}_{[v]}\rangle\}\} \\ \sigma\{\langle\mathcal{R}, \langle\mathcal{R}'\rangle\rangle\} & \mapsto & \sigma\{\{\langle\mathcal{R}, \langle\mathcal{R}' \bullet \mathcal{R}_{[v]}\rangle\rangle\}\} \\ \sigma\{\langle\mathcal{R}, \langle\mathcal{R}'\rangle, \langle\mathcal{R}''\rangle\rangle\} & \mapsto & \sigma\{\{\langle\mathcal{R}, \langle\mathcal{R}'\rangle, \langle\mathcal{R}'' \bullet \mathcal{R}_{[v]}\rangle\rangle\}\} \end{array}\right]$

v. $[\![\mathbf{if}]\!] = \left[\begin{array}{lll} \sigma\{\langle\mathcal{R}\rangle\} & \mapsto & \sigma\{\{\langle\mathcal{R}, \langle\top\rangle\rangle\}\} \\ \sigma\{\langle\mathcal{R}, \langle\mathcal{R}'\rangle\rangle\} & \mapsto & \sigma\{\{\langle\mathcal{R}, \langle\mathcal{R}', \langle\top\rangle\rangle\rangle\}\} \\ \sigma\{\langle\mathcal{R}, \langle\mathcal{R}'\rangle, \langle\mathcal{R}''\rangle\rangle\} & \mapsto & \sigma\{\{\langle\mathcal{R}, \langle\mathcal{R}'\rangle, \langle\mathcal{R}'', \langle\top\rangle\rangle\rangle\}\} \end{array}\right]$

vi. $[\![\mathbf{then}]\!] = \left[\begin{array}{lll} \sigma\{\langle\mathcal{R}, \langle\mathcal{R}'\rangle\rangle\} & \mapsto & \sigma\{\{\langle\mathcal{R}, \langle\mathcal{R}'\rangle, \langle\top\rangle\rangle\}\} \end{array}\right]$

vii. $[\![\mathbf{end}]\!] = \left[\begin{array}{lll} \sigma\{\langle\mathcal{R}, \langle\mathcal{R}'\rangle, \langle\mathcal{R}''\rangle\rangle\} & \mapsto & \sigma\{\{\langle\mathcal{R} \bullet (\mathcal{R}' \twoheadrightarrow \mathcal{R}'')\rangle\}\} \end{array}\right]$

b. Formulas (conjunction):

i. $[\![\varphi; \psi]\!] = \left[\begin{array}{lll} \sigma & \mapsto & (\sigma[\![\varphi]\!])[\![\psi]\!] \end{array}\right]$

c. Abbreviations:

i. $[\![\varphi \to \psi]\!] = [\![\mathbf{if}; \varphi; \mathbf{then}; \psi; \mathbf{end}]\!]$

ii. $[\![\sim \varphi]\!] = [\![\varphi \to \bot]\!]$

iii. $[\![\exists v\varphi]\!] = [\![[v];\ \varphi]\!]$

iv. $[\![\forall v\varphi]\!] = [\![[v] \to \varphi]\!]$

d. Truth:

i. A formula φ is true in model $\mathfrak{M}$ relative to an input info state $\mathcal{R} \subseteq \mathcal{G} \times \mathcal{G}$ iff there exists an output info state $\mathcal{R}' \subseteq \mathcal{G} \times \mathcal{G}$ that is non-empty (i.e., $\mathcal{R}' \neq \bot$) such that $\langle\mathcal{R}\rangle [\![\varphi]\!] = \langle\mathcal{R}'\rangle$.

IDPL preserves thr DPL equivalences that enable it to account for donkey anaphora. In particular, existentials have unlimited scope over conjuncts to the right and they can freely scope out of conditional antecedents:

(69)
$$\begin{array}{ll} (\exists v\varphi);\ \psi & \Leftrightarrow \\ ([v];\ \varphi);\ \psi & \Leftrightarrow \\ [v];\ (\varphi;\ \psi) & \Leftrightarrow \\ \exists v(\varphi;\ \psi) & \end{array}$$

(70)
$$\begin{array}{ll} (\exists v\varphi) \to \psi & \Leftrightarrow \\ ([v];\ \varphi) \to \psi & \Leftrightarrow \\ \mathbf{if}; [v]; \varphi; \mathbf{then}; \psi; \mathbf{end} & \Leftrightarrow \\ \mathbf{if}; [v]; \mathbf{then}; \mathbf{if}; \varphi; \mathbf{then}; \psi; \mathbf{end}; \mathbf{end} & \Leftrightarrow \\ [v] \to (\varphi \to \psi) & \Leftrightarrow \\ \forall v(\varphi \to \psi) & \end{array}$$

The typical donkey conditional and its IDPL translation are provided below. The resulting formula can be strictly incrementally interpreted and the correct truth conditions are derived.

(71) If a^x farmer owns a^y donkey, he$_x$ beats it$_y$.

(72) a. $\exists x(\text{FARMER}(x); \exists y(\text{DONKEY}(y); \text{OWN}(x, y))) \to \text{BEAT}(x, y)$

b. $\mathbf{if}; [x]; \text{FARMER}(x); [y]; \text{DONKEY}(y); \text{OWN}(x, y); \mathbf{then}; \text{BEAT}(x, y); \mathbf{end}$

(73)
$$\begin{array}{l}
\langle\top\rangle [\![\mathbf{if}; [x]; \text{FARMER}(x); [y]; \text{DONKEY}(y); \text{OWN}(x, y); \mathbf{then}; \text{BEAT}(x, y); \mathbf{end}]\!] = \\
\langle\top, \langle\top\rangle\rangle [\![[x]; \text{FARMER}(x); [y]; \text{DONKEY}(y); \text{OWN}(x, y); \mathbf{then}; \text{BEAT}(x, y); \mathbf{end}]\!] = \\
\langle\top, \langle\top \bullet \mathcal{R}_{[x]}\rangle\rangle [\![\text{FARMER}(x); [y]; \text{DONKEY}(y); \text{OWN}(x, y); \mathbf{then}; \text{BEAT}(x, y); \mathbf{end}]\!] = \\
\langle\top, \langle\mathcal{R}_{[x]} \bullet \mathcal{R}_{\text{FARMER}(x)}\rangle\rangle [\![[y]; \text{DONKEY}(y); \text{OWN}(x, y); \mathbf{then}; \text{BEAT}(x, y); \mathbf{end}]\!] = \\
\langle\top, \langle\mathcal{R}_{[x]} \bullet \mathcal{R}_{\text{FARMER}(x)} \bullet \mathcal{R}_{[y]}\rangle\rangle [\![\text{DONKEY}(y); \text{OWN}(x, y); \mathbf{then}; \text{BEAT}(x, y); \mathbf{end}]\!] = \\
\langle\top, \langle\mathcal{R}_{[x]} \bullet \mathcal{R}_{\text{FARMER}(x)} \bullet \mathcal{R}_{[y]} \bullet \mathcal{R}_{\text{DONKEY}(y)}\rangle\rangle [\![\text{OWN}(x, y); \mathbf{then}; \text{BEAT}(x, y); \mathbf{end}]\!] = \\
\langle\top, \langle\mathcal{R}_{[x]} \bullet \mathcal{R}_{\text{FARMER}(x)} \bullet \mathcal{R}_{[y]} \bullet \mathcal{R}_{\text{DONKEY}(y)} \bullet \mathcal{R}_{\text{OWN}(x,y)}\rangle\rangle [\![\mathbf{then}; \text{BEAT}(x, y); \mathbf{end}]\!] = \\
\langle\top, \langle\mathcal{R}_{[x]\bullet\text{FARMER}(x)\bullet[y]\bullet\text{DONKEY}(y)\bullet\text{OWN}(x,y)}\rangle, \langle\top\rangle\rangle [\![\text{BEAT}(x, y); \mathbf{end}]\!] =
\end{array}$$

$$\begin{aligned}
&\Big\langle \top, \Big\langle \mathcal{R}_{[x]\bullet\text{FARMER}(x)\bullet[y]\bullet\text{DONKEY}(y)\bullet\text{OWN}(x,y)} \Big\rangle, \Big\langle \top \bullet \mathcal{R}_{\text{BEAT}(x,y)} \Big\rangle \Big\rangle [\![\mathbf{end}]\!] = \\
&\Big\langle \top \bullet (\mathcal{R}_{[x]\bullet\text{FARMER}(x)\bullet[y]\bullet\text{DONKEY}(y)\bullet\text{OWN}(x,y)} \twoheadrightarrow \mathcal{R}_{\text{BEAT}(x,y)}) \Big\rangle = \\
&\Big\langle \mathcal{R}_{[x]\bullet\text{FARMER}(x)\bullet[y]\bullet\text{DONKEY}(y)\bullet\text{OWN}(x,y)} \twoheadrightarrow \mathcal{R}_{\text{BEAT}(x,y)} \Big\rangle = \\
&\Big\langle \{\langle g, g\rangle : g\mathcal{R}_{[x]\bullet\text{FARMER}(x)\bullet[y]\bullet\text{DONKEY}(y)\bullet\text{OWN}(x,y)} \subseteq \mathbf{Dom}(\mathcal{R}_{\text{BEAT}(x,y)})\} \Big\rangle = \\
&\langle \{\langle g, g\rangle : \text{ all pairs } \langle \alpha, \beta\rangle \text{ s.t. } \alpha \in [\![\text{FARMER}]\!], \beta \in [\![\text{DONKEY}]\!], \langle \alpha, \beta\rangle \in [\![\text{OWN}]\!] \\
&\qquad\qquad\qquad\qquad \text{are s.t. } \langle \alpha, \beta\rangle \in [\![\text{BEAT}]\!]\}\rangle
\end{aligned}$$

6 Conclusion

While our discussion of Incremental DPL semantics has been very terse (we hope to improve on it in the future), it accomplishes the main goal for this paper, namely motivating and fully defining a strictly incremental semantics for dynamic predicate logic. This semantics extends the incremental semantics for dynamic propositional logic introduced in Vermeulen (1994), borrowing central notions from Visser (2002).

The resulting logical system (Incremental DPL) can derive correct truth conditions for apparently non-incremental structures like donkey conditionals in a *strictly* incremental fashion. That is, the correct meanings for donkey conditionals are derived by means of a strictly left-to-right compositional procedure.

This is accomplished without having to type-shift the meanings of the individual words (as in Steedman 2001, for example), and with dynamic conjunction / sequencing as the only compositional operation.

References

Chater, Nick et al. (1995). "What is incremental interpretation?" In: *Incremental Interpretation (Edinburgh Working Papers in Cognitive Science)*. Ed. by David Milward and Patrick Sturt. Vol. 11. Edinburgh: Edinburgh University, pp. 1–23.

Groenendijk, Jeroen and Martin Stokhof (1991). "Dynamic Predicate Logic". In: *Linguistics and Philosophy* 14.1, pp. 39–100.

Heim, Irene (1982). "The semantics of definite and indefinite noun phrases (published 1988, New York: Garland)". PhD thesis. Amherst, MA: UMass Amherst.

Kamp, Hans (1981). "A Theory of Truth and Semantic Representation". In: *Formal Methods in the Study of Language*. Ed. by Jeroen Groenendijk et al. Amsterdam: Mathematical Centre Tracts, pp. 277–322.

Kamp, Hans and Uwe Reyle (1993). *From Discourse to Logic. Introduction to Model theoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Dordrecht: Kluwer.

Milward, David and Robin Cooper (1994). "Incremental interpretation: Applications, Theory, and Relationship to Dynamic Semantics". In: *The 15th International Conference on Computational Linguistics (COLING 94)*. Kyoto, Japan: COLING 94 Organizing Comm., pp. 748–754.

Steedman, Mark (2001). *The Syntactic Process*. Cambridge, MA: MIT Press.

Vermeulen, C.F.M. (1994). "Incremental Semantics for Propositional Texts". In: *Notre Dame Journal of Formal Logic* 35.2, pp. 243–271.

Visser, Albert (2002). "The Donkey and the Monoid". In: *Journal of Logic, Language and Information* 11, pp. 107–131.