# Computational Semantics, Type Theory, and Functional Programming

## II — Dynamic Montague Grammar and its Shortcomings

Jan van Eijck

CWI and ILLC, Amsterdam, Uil-OTS, Utrecht

LOLA7 Tutorial, Pecs

August 2002

## Summary

- Point of Departure: Dynamic Predicate Logic
- Extensions to Typed Logic
- Building a Montague Fragment from This
- States, Propositions, Transitions
- Implementation
- Chief Weakness

## Point of Departure: DPL

Dynamic Predicate Logic or DPL [GS91] is our point of departure.

Assume a first order model $M = (D, I)$ and a set of variables $V$. States for DPL are functions in $D^V$.

Let $M = (D, I)$ and let $s \in D^V$.

Term interpretation:

$[\![c]\!]_s^M = I(c)$ for constants $c$ and $[\![v]\!]_s^M = s(v)$ for variables $v$.

Define the relation

$$M \models_s Pt_1 \cdots t_n$$

by means of:

$$M \models_s Pt_1 \cdots t_n \ :\Leftrightarrow\ \langle [\![t_1]\!]_s^M, \ldots, [\![t_n]\!]_s^M \rangle \in I(P),$$

and the relation

$$M \models_s t_1 \doteq t_2$$

by means of:

$$M \models_s t_1 \doteq t_2 \ :\Leftrightarrow\ [\![t_1]\!]_s^M = [\![t_2]\!]_s^M.$$

If $x \in V$, $d \in D$ and $s \in D^V$, use $(x|d)s$ for the state $s'$ that differs from $s$ at most in the fact that $x$ gets mapped to $d$.

The DPL interpretation of formulas can now be given as a map in $D^V \to \mathcal{P}(D^V)$.

## DPL Semantics — Functional Version

$$[\![\exists x]\!](s) \ := \ \{(x|d)s \mid d \in D\}$$

$$[\![Pt_1 \cdots t_n]\!](s) \ := \ \begin{cases} \{s\} & \text{if } M \models_s Pt_1 \cdots t_n \\ \emptyset & \text{otherwise,} \end{cases}$$

$$[\![t_1 \doteq t_2]\!](s) \ := \ \begin{cases} \{s\} & \text{if } M \models_s t_1 \doteq t_2 \\ \emptyset & \text{otherwise,} \end{cases}$$

$$[\![\neg\varphi]\!](s) \ := \ \begin{cases} \{s\} & \text{if } [\![\varphi]\!](s) = \emptyset, \\ \emptyset & \text{otherwise,} \end{cases}$$

$$[\![\varphi;\psi]\!](s) \ := \ \bigcup\{[\![\psi]\!](s') \mid s' \in [\![\varphi]\!](s)\}$$

Note that predicates, identities and negations are interpreted as tests: if $s$ is the input state they either return $\{s\}$, in case the test succeeds, or $\emptyset$, in case the test fails.

The action of existential quantification is *not* a test.

Dynamic implication $\varphi \Rightarrow \psi$ can be defined in terms of $\neg$ and $;$ by means of $\neg(\varphi; \neg\psi)$. This is a test.

Universal quantification $\forall x \varphi$ is defined in terms of $\exists$, $\neg$ and $;$ as $\neg(\exists x; \neg\varphi)$. This is a test.

## DPL Semantics — Relational Version

$$M, s, s' \models \exists x \;:\Longleftrightarrow\; \text{there is a } d \in D \text{ with } s' = (x|d)s.$$

$$M, s, s' \models Pt_1 \cdots t_n \;:\Longleftrightarrow\; s = s' \text{ and } M \models_s Pt_1 \cdots t_n$$

$$M, s, s' \models t_1 \doteq t_2 \;:\Longleftrightarrow\; s = s' \text{ and } M \models_s t_1 \doteq t_2$$

$$M, s, s' \models \neg\varphi \;:=\; s = s' \text{ and there is no } s'' \text{ with } M, s, s'' \models \varphi$$

$$M, s, s' \models \varphi; \psi \;:=\; \text{there is an } s'' \text{ with } M, s, s'' \models \varphi \text{ and } M, s'', s' \models \psi.$$

## States as Carriers of Anaphoric Information

The advantage of the propagation of variable states is that they carry anaphoric information that can be used for the interpretation of subsequent discourse.

**1** *Some*$^1$ *man loved some*$^2$ *woman.*

The DPL rendering of (1) is $\exists u_1; Mu_1; \exists u_2; Wu_2; Lu_1u_2$.

This gets interpreted as the set of all maps $u_1 \mapsto e_1, u_2 \mapsto e_2$ that satisfy the relation ‘love’ in the model under consideration.

The result of this is that the subsequent sentence (2) can now use this discourse information to pick up the references:

**2** *He*$_1$ *kissed her*$_2$.

## Extensions to Typed Logic

Attempts to incorporate DPL stype dynamic semantics in mainstream Montague style natural language semantics [Mon73] can be found in [GS90, Chi92, Jan98, Mus95, Mus96, Mus94, Eij97, EK97, KKP96, Kus00].

The type hierarchy employed has basic types for entities ($e$), truth values($t$), and markers ($m$). The states themselves can be viewed as maps from markers to suitable referents, i.e., a state has type $m \rightarrow e$. We abbreviate this as $s$.

This can either be built into the type system from the start (see [Eij97]) or enforced by means of axioms (a kind of meaning postulates for proper state behaviour; see [Mus95, Mus94]).

Various set-ups of the encoding of DPL to type theory are possible. In the most straightforward approach, the meaning of a formula is no longer a truth value, but a state transition, i.e., the interpretations of formulas have type $s \longrightarrow s \longrightarrow t$.

Following [Eij97], we will take $(u|x)$ as a primitive operation of type $s \longrightarrow s$ that resets the value of $u$ to $x$.

Thus, $(u|x)a$ denotes the state $a'$ that differs from state $a$ at most in the fact that the value in $a'$ for marker $u$ is (the interpretation of) $x$.

Then the translation of an indefinite noun phrase *a man* becomes something like:

**3** $\lambda P \lambda a \lambda a'.\exists x(\textsf{\textit{man}}\ x \wedge P u_i\ (u_i|x)a\ a')$.

Here $P$ is a variable of type $m \longrightarrow s \longrightarrow s \longrightarrow t$ and $a, a'$ are variables of type $s$, so the translation (3) has type $(m \longrightarrow s \longrightarrow s \longrightarrow t) \longrightarrow s \longrightarrow s \longrightarrow t$.

## State Transitions

Note that $s \longrightarrow s \longrightarrow t$ is the type of a (characteristic function of) a binary relation on states, or, as we will call it, the type of a *state transition*.

In (the present version of) dynamic semantics, VPs are interpreted as maps from markers to state transitions.

Translation (3) introduces an anaphoric index $i$; as long as $u_i$ does not get reset, any reference to $u_i$ will pick up the link to the indefinite man that was introduced into the discourse.

## Encodings of Dynamic Operations in Typed Logic

Assume $\varphi$ and $\psi$ have the type of state transitions, i.e., type $s \to s \to t$, and that $a, a', a''$ have type $s$.

$$\boldsymbol{\exists} u_i \ := \ \lambda a a'.\exists x((u_i|x)a = a')$$

$$\boldsymbol{\neg}\varphi \ := \ \lambda a a'.(a = a' \wedge \neg\exists a''\varphi a a'')$$

$$\varphi \, \boldsymbol{;} \, \psi \ := \ \lambda a a'.\exists a''(\varphi a a'' \wedge \psi a'' a')$$

$$\varphi \Rightarrow \psi \ := \ \boldsymbol{\neg}(\varphi \, \boldsymbol{;} \, \boldsymbol{\neg}\psi)$$

It is also useful to define an operation $! : (s \to t) \to t$ to indicate that the state set $s \to t$ is not empty. Thus, ! serves as an indication of success. Assume $p$ to be an expression of type $s \to t$, the definition of ! is:

$$!p \ := \ \exists a.(pa).$$

Note that ¬ can now be defined in terms of !, as

$$\neg\varphi \ := \ \lambda aa'.(a = a' \wedge \neg!\varphi a)$$

## Lifting Lexical Meanings to the Discourse Level

We have to assume that the lexical meanings of CNs, VPs are given as one-placed predicates (type $e \to t$) and those of TVs as two-placed predicates (type $e \to e \to t$).

It makes sense to define blow-up operations for lifting one-placed and two-placed predicates to the dynamic level. Assume $A$ to be an expression of type $e \to t$, and $B$ an expression of type $e \to e \to t$); we use $r, r'$ as variables of type $m$, $a, a'$ as variables of type $s = m \to e$, and we employ postfix notation for the lifting operations:

$$\begin{array}{rcl} A^{\circ} & := & \lambda r \lambda a \lambda a'(a = a' \wedge A(ar)) \\ B^{\bullet} & := & \lambda r \lambda r' \lambda a \lambda a'(a = a' \wedge B(ar)(ar')) \end{array}$$

The encodings of the DPL operations in typed logic and the blow-up operations for one- and two-placed predicates are employed in the semantic specification of the fragment. The semantic specifications employ variables $P, Q$ of type $m \to s \to s \to t$, variables $u, u'$ of type $m$, and variables $a, a'$ of type $s$.

| | | | | | |
|---|---|---|---|---|---|
| **S** | $::=$ | **NP VP** | $X$ | $::=$ | $(X_1 X_2)$ |
| **S** | $::=$ | *if* **S** **S** | $X$ | $::=$ | $X_2 \Rightarrow X_3$ |
| **NP** | $::=$ | *Mary*$^n$ | $X$ | $::=$ | $\lambda P a a'.(P u_n a a')$ |
| **NP** | $::=$ | *Bill*$^n$ | $X$ | $::=$ | $\lambda P a a'.(P u_n a a')$ |
| **NP** | $::=$ | *PRO*$^n$ | $X$ | $::=$ | $\lambda P a a'.(P u_n a a')$ |
| **NP** | $::=$ | **DET CN** | $X$ | $::=$ | $(X_1 X_2)$ |
| **NP** | $::=$ | **DET RCN** | $X$ | $::=$ | $(X_1 X_2)$ |

$$
\begin{array}{lll}
\textbf{DET} & ::= & \textit{every}^n \\
X & ::= & \lambda PQ.(\boldsymbol{\exists} u_n \; \boldsymbol{;} \; Pu_n) \Rightarrow Qu_n \\[1ex]
\textbf{DET} & ::= & \textit{some}^n \\
X & ::= & \lambda PQ.\boldsymbol{\exists} u_n \; \boldsymbol{;} \; Pu_n \; \boldsymbol{;} \; Qu_n \\[1ex]
\textbf{DET} & ::= & \textit{no}^n \\
X & ::= & \lambda PQ.\boldsymbol{\neg}(\boldsymbol{\exists} u_n \; \boldsymbol{;} \; Pu_n \; \boldsymbol{;} \; Qu_n) \\[1ex]
\textbf{DET} & ::= & \textit{the}^n \\
X & ::= & \lambda PQ. \\
 & & \lambda aa'.a = a' \wedge \exists x \forall y (!(\boldsymbol{\exists} u_n \; \boldsymbol{;} \; Pu_n \; (u_n|y)a) \leftrightarrow x = y) \; \boldsymbol{;} \\
 & & \qquad \boldsymbol{\exists} u_n \; \boldsymbol{;} \; Pu_n \; \boldsymbol{;} \; Qu_n
\end{array}
$$

| | | | | | |
|---|---|---|---|---|---|
| **CN** | $::=$ | *man* | $X$ | $::=$ | $M^\circ$ |
| **CN** | $::=$ | *woman* | $X$ | $::=$ | $W^\circ$ |
| **CN** | $::=$ | *boy* | $X$ | $::=$ | $B^\circ$ |
| **RCN** | $::=$ | **CN** *that* **VP** | $X$ | $::=$ | $\lambda u.((X_1\ u)\ \mathbf{;}\ (X_3\ u))$ |
| **RCN** | $::=$ | **CN** *that* **NP TV** | $X$ | $::=$ | $\lambda u.((X_1\ u)\ \mathbf{;}\ (X_3(\lambda u'.((X_4\ u')u)))$ |
| **VP** | $::=$ | *laughed* | $X$ | $::=$ | $L^\circ$ |
| **VP** | $::=$ | *smiled* | $X$ | $::=$ | $S^\circ$ |
| **VP** | $::=$ | **TV NP** | $X$ | $::=$ | $\lambda u.(X_2\ \mathbf{;}\ \lambda u'.((X_1\ u')u))$ |
| **TV** | $::=$ | *loved* | $X$ | $::=$ | $L'^\bullet$ |
| **TV** | $::=$ | *respected* | $X$ | $::=$ | $R^\bullet$ |

## Comparison with Classical Montague Grammar

The types of the dynamic meanings are systematically related to the types of the earlier static meanings by a replacement of truth values (type $t$) by transitions (type $s \rightarrow s \rightarrow t$), and of entities (type $e$) by markers (type $m$).

The translation of the proper names assumes that every name is linked to an *anchored* marker (a marker that is never updated).

## Implementation – Basic Types

```
module LOLA2 where

import Domain
import Model
```

Apart from Booleans and Entities, we need basic types for (reference) markers. Reference markers are the dynamic variables that carry discourse information. For convenience, we also declare a type for indices, and a map from indices to markers.

```
data Marker = U0 | U1 | U2 | U3 | U4
            | U5 | U6 | U7 | U8 | U9
     deriving (Eq,Bounded,Enum,Show)

type Idx = Int

i2m :: Idx -> Marker
i2m i | i < 0 || i > fromEnum (maxBound::Marker)
        = error "idx out of range"
      | otherwise
        = toEnum i
```

## States, Propositions

The type of states is `Marker -> Entity`.

For purposes of implementation, we will represent maps from markers to entities as lists of marker/entity pairs.

```
type State = [(Marker,Entity)]
```

Propositions are collections of states. We will represent propositions as lists of states:

```
type Prop = [State]
```

## Transitions

Transitions are mappings from states to propositions, i.e., their type is `State -> Prop`.

```
type Trans = State -> Prop
```

## Applying a state to a marker

Next, we need a function for application of a state to a marker. An error message is generated if the marker is not in the domain.

```
apply :: State -> Marker -> Entity
apply [] m = error (show m ++ " not in domain")
apply ((m,e):xs) m’ | m == m’    = e
                    | otherwise = apply xs m’
```

## Updates

Updating a reference marker by mapping it to a new entity in a state (the implementation of the operation $(u|x)$):

```
update :: Marker -> Entity -> State -> State
update m e s = replace m e s where
     replace _ _ [] = error "undefined"
     replace m e ((m',e'):xs)
        | m == m'    = (m,e):xs
        | otherwise = (m',e'):(replace m e xs)
```

## Dynamic Negation

Dynamic negation is a test on a state $s$ that succeeds of the negated formula fails in that state, and succeeds otherwise.

```
neg :: Trans -> Trans
neg = \ phi s -> if (phi s == []) then [s] else []
```

## Dynamic Conjunction

Dynamic conjunction applies the first conjunct to the initial state, next applies the second conjuncts to all intermediate results, and finally collects all end results.

```
conj :: Trans -> Trans -> Trans
conj = \ phi psi s ->
       concat [ psi s’ | s’ <- (phi s) ]
```

## Dynamic Quantification

Dynamic existential quantification is defined in terms of `update`, making use of the fact that the domain of entities is bounded.

```
exists :: Marker -> Trans
exists = \ m s ->
           [ update m x s | x <- entities]
```

The universal quantifier is defined in terms of dynamic existential quantification, dynamic conjunction and dynamic negation.

```
forall :: Marker -> Trans -> Trans
forall = \ m phi ->
           neg ((exists m) `conj` (neg phi))
```

## Dynamic Implication

Dynamic implication is defined in terms of dynamic conjunction and dynamic negation.

```
impl :: Trans -> Trans -> Trans
impl = \ phi psi -> neg (phi ‘conj‘ (neg psi))
```

## Anchors for Proper Names

To get a reasonable treatment of proper names, we assume that some of the discourse markers are anchored to entities:

```
anchored :: Marker -> Entity -> Bool
anchored U6 A = True
anchored U7 M = True
anchored U8 B = True
anchored U9 J = True
anchored _  _ = False
```

## Syntax of the Fragment

The datatype declarations for syntax are almost as for classical Montague grammar. The main difference is that all noun phrases now carry index information. The index on proper names and pronouns is directly attached to the name or pronoun; the index information on a complex NP is attached to the determiner.

```
data S = S NP VP | If S S | Txt S S
     deriving (Eq,Show)

data NP = Ann Idx | Mary Idx | Bill Idx
        | Johnny Idx | PRO Idx
        | NP1 DET CN | NP2 DET RCN
     deriving (Eq,Show)

data DET = Every Idx | Some Idx | No Idx | The Idx
     deriving (Eq,Show)
```

```
data CN = Man | Woman | Boy | Person | Thing | House
     deriving (Eq,Show)

data RCN = CN1 CN VP | CN2 CN NP TV
     deriving (Eq,Show)

data VP = Laughed | Smiled | VP TV NP
     deriving (Eq,Show)

data TV = Loved | Respected | Hated | Owned
     deriving (Eq,Show)
```

## Lexical Meaning

We intend to use the information from the first order model in module `Model`. Thus, we assume that lexical VPs and CNs have a lexical meaning of type `Entity -> Bool`, which is subsequently blown up to a suitable discourse type. Similarly for two-placed predicates.

Mapping one-place predicates to functions from markers to transitions (or: discourse predicates) is done by:

```
blowupPred :: (Entity -> Bool) -> Marker -> Trans
blowupPred pred m s | pred (apply s m) = [s]
                    | otherwise        = []
```

Discourse blow-up for two-place predicates.

```
blowupPred2 :: (Entity -> Entity -> Bool) ->
                   Marker -> Marker -> Trans
blowupPred2 pred m1 m2 s
   | pred (apply s m1) (apply s m2) = [s]
   | otherwise                      = []
```

## Dynamic Interpretation

The interpretation of sentences now produces transitions (type `Trans`) rather than booleans:

```
intS :: S -> Trans
intS (S np vp)    = (intNP np) (intVP vp)
intS (If s1 s2)   = (intS s1) `impl` (intS s2)
intS (Txt s1 s2) = (intS s1) `conj` (intS s2)
```

In fact, we can get at the types of all the translation instructions by systematically replacing `Bool` by `Trans`, and `Entity` by `Marker`.

Interpretation of proper names: the code checks whether the proper names employ a suitably anchored marker, and generates an error message in case they are not.

```
intNP :: NP -> (Marker -> Trans) -> Trans
intNP (Ann i) p | anchored (i2m i) A = p (i2m i)
                | otherwise = error "wrong anchor"
intNP (Mary i) p | anchored (i2m i) M = p (i2m i)
                 | otherwise = error "wrong anchor"
intNP (Bill i) p | anchored (i2m i) B = p (i2m i)
                 | otherwise = error "wrong anchor"
intNP (Johnny i) p | anchored (i2m i) J = p (i2m i)
                   | otherwise = error "wrong anchor"
```

Interpretation of pronouns: use the anchor indicated by the index:

```
intNP (PRO i) p = p (i2m i)
```

Interpretation of complex NPs: use the appropriate recursion.

```
intNP (NP1 det cn) p = (intDET det) (intCN cn) p
intNP (NP2 det rcn) p = (intDET det) (intRCN rcn) p
```

Interpretation of lexical VPs uses the dynamic blow-up from the lexical meanings.

```
intVP :: VP -> Marker -> Trans
intVP Laughed subj = blowupPred laugh subj
intVP Smiled  subj = blowupPred smile subj
```

Complex VPs:

```
intVP (VP tv np) subj = intNP np phi
   where phi obj = intTV tv obj subj
```

Interpretation of TVs uses discourse blow-up of two-place predicates.

```
intTV :: TV -> Marker -> Marker -> Trans
intTV tv = blowupPred2 (lexTV tv)
  where lexTV Loved     = (flip . curry) love
        lexTV Respected = (flip . curry) respect
        lexTV Hated     = (flip . curry) hate
        lexTV Owned     = (flip . curry) own
```

Interpretation of CNs uses discourse blow-up of one-place predicates.

```
intCN :: CN -> Marker -> Trans
intCN Man    = blowupPred man
intCN Boy    = blowupPred boy
intCN Woman  = blowupPred woman
intCN Person = blowupPred person
intCN Thing  = blowupPred thing
intCN House  = blowupPred house
```

Code for checking that a discourse predicate is unique.

```
unique :: Marker -> Trans -> Trans
unique m phi s | singleton xs = [s]
               | otherwise    = []
    where singleton [x] = True
          singleton  _  = False
          xs = [x | x <- entities,
                    success (update m x s) phi]
          success s psi = phi s /= []
```

Discourse type of determiners: combine two discourse predicates into a transition.

```
intDET :: DET ->
   (Marker -> Trans) -> (Marker -> Trans) -> Trans
```

Interpretation of determiners in terms of the dynamic operators defined above.

```
intDET (Some i) phi psi =
   exists m `conj` (phi m) `conj` (psi m)
   where m = i2m i
intDET (Every i) phi psi =
   (exists m `conj` (phi m)) `impl` (psi m)
   where m = i2m i
intDET (No i) phi psi =
   neg (exists m `conj` (phi m) `conj` (psi m))
   where m = i2m i
intDET (The i) phi psi =
   (unique m (phi m)) `conj`
    (exists m) `conj` (phi m) `conj` (psi m)
    where m = i2m i
```

Interpretation of relativised common nouns: straightforward generalisation of the treatment in classical Montague grammar:

```
intRCN :: RCN -> Marker -> Trans
intRCN (CN1 cn vp) m =
   conj (intCN cn m) (intVP vp m)
intRCN (CN2 cn np tv) m =
   conj (intCN cn m) (intNP np (intTV tv m))
```

## Testing It Out

We need a suitable start state. Note that this start states respects the anchoring information for the proper names.

```
startstate :: State
startstate =
  [(U0,A),(U1,B),(U2,C),(U3,D),(U4,E),
   (U5,F),(U6,A),(U7,M),(U8,B),(U9,J)]
```

Evaluation starts out from the initial state and produces a list of states. An empty output state set indicates that the sentence is false, a non-empty output state set indicates that the sentence is true. The output states encode the anaphoric discourse information that is available for subsequent discourse.

```
eval :: S -> Prop
eval s = intS s startstate
```

```
LOLA2> eval (S (Johnny 10) Smiled)

Program error: idx out of range

LOLA2> eval (S (Johnny 8) Smiled)

Program error: wrong anchor

LOLA2> eval (S (Johnny 9) Smiled)
[]

LOLA2>  eval (S (Mary 7) Smiled)
[[(U0,A),(U1,B),(U2,C),(U3,D),(U4,E),
  (U5,F),(U6,A),(U7,M),(U8,B),(U9,J)]]
```

```
test4 = eval
  (S (NP1 (The 1) Boy)
     (VP Loved (NP1 (Some 2) Woman)))
```

```
LOLA2> test4
[[(U0,A),(U1,B),(U2,M),(U3,D),(U4,E),
  (U5,F),(U6,A),(U7,M),(U8,B),(U9,J)]]
```

```
test5 = eval
  (S (NP1 (Some 1) Man)
     (VP Loved (NP1 (Some 2) Woman)))
```

```
LOLA2> test5
[[(U0,A),(U1,B),(U2,M),(U3,D),(U4,E),
  (U5,F),(U6,A),(U7,M),(U8,B),(U9,J)],
 [(U0,A),(U1,J),(U2,M),(U3,D),(U4,E),
  (U5,F),(U6,A),(U7,M),(U8,B),(U9,J)]]
```

```
test6 = eval
  (S (NP1 (Some 1) Man)
     (VP Respected (NP1 (Some 2) Woman)))
```

```
LOLA2> test6
[[(U0,A),(U1,B),(U2,A),..],
 [(U0,A),(U1,B),(U2,C),..],
 [(U0,A),(U1,B),(U2,M),..],
 [(U0,A),(U1,D),(U2,A),..],
 [(U0,A),(U1,D),(U2,C),..],
 [(U0,A),(U1,D),(U2,M),..],
 [(U0,A),(U1,J),(U2,A),..],
 [(U0,A),(U1,J),(U2,C),..],
 [(U0,A),(U1,J),(U2,M),..]]
```

```
test7 = eval
  (S (NP1 (The 1) Man)
     (VP Loved (NP1 (Some 2) Woman)))
```

```
LOLA2> test7
[]
```

```
test8 = eval
  (S (NP1 (Some 1) Man)
     (VP Loved (NP1 (Some 2) Woman)))
```

```
LOLA2> test8
[[(U0,A),(U1,B),(U2,M),(U3,D),(U4,E),
  (U5,F),(U6,A),(U7,M),(U8,B),(U9,J)],
 [(U0,A),(U1,J),(U2,M),(U3,D),(U4,E),
  (U5,F),(U6,A),(U7,M),(U8,B),(U9,J)]]
```

```
test9 = eval
  (Txt (S (NP1 (Some 1) Man)
          (VP Loved (NP1 (Some 2) Woman)))
       (S (PRO 1) (VP Respected (PRO 2))))
```

```
LOLA2> test9
[[(U0,A),(U1,B),(U2,M),(U3,D),(U4,E),
  (U5,F),(U6,A),(U7,M),(U8,B),(U9,J)],
 [(U0,A),(U1,J),(U2,M),(U3,D),(U4,E),
  (U5,F),(U6,A),(U7,M),(U8,B),(U9,J)]]
```

```
test10 = eval
  (If (S (NP1 (Some 1) Man)
         (VP Loved (NP1 (Some 2) Woman)))
      (S (PRO 1) (VP Respected (PRO 2))))
```

```
LOLA2> test10
[[(U0,A),(U1,B),(U2,C),(U3,D),(U4,E),
  (U5,F),(U6,A),(U7,M),(U8,B),(U9,J)]]
```

```
test11 = eval
  (Txt (S (NP1 (Some 1) Man)
         (VP Respected (NP1 (Some 2) Woman)))
       (S (PRO 1) (VP Loved (PRO 2))))
```

```
LOLA2> test11
[[(U0,A),(U1,B),(U2,M),(U3,D),(U4,E),
  (U5,F),(U6,A),(U7,M),(U8,B),(U9,J)],
 [(U0,A),(U1,J),(U2,M),(U3,D),(U4,E),
  (U5,F),(U6,A),(U7,M),(U8,B),(U9,J)]]
```

```
test12 = eval
  (If (S (NP1 (Some 1) Man)
         (VP Respected (NP1 (Some 2) Woman)))
      (S (PRO 1) (VP Loved (PRO 2))))
```

```
LOLA2> test12
[]
```

## Chief Weakness

The chief weakness of DPL-based NL semantics is the need to supply indices for all noun phrases in advance.

Moreover, re-use of an index destroys access to the previous value of the corresponding marker.

This is a reflection of the fact that DPL has destructive assignment: quantification over $u$ replaces the previous contents of register $u$ by something new, and the old value gets lost forever.

## References

[Chi92] G. Chierchia. Anaphora and dynamic binding. *Linguistics and Philosophy*, 15(2):111–183, 1992.

[Eij97] J. van Eijck. Typed logics with states. *Logic Journal of the IGPL*, 5(5):623–645, 1997.

[EK97] J. van Eijck and H. Kamp. Representing discourse in context. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, pages 179–237. Elsevier, Amsterdam, 1997.

[GS90] J. Groenendijk and M. Stokhof. Dynamic Montague Grammar. In L. Kalman and L. Polos, editors, *Papers from the Second Symposium on Logic and Language*, pages 3–48. Akademiai Kiadoo, Budapest, 1990.

[GS91] J. Groenendijk and M. Stokhof. Dynamic predicate logic. *Linguistics and Philosophy*, 14:39–100, 1991.

[Jan98] Martin Jansche. Dynamic Montague Grammar lite. Dept of Linguistics, Ohio State University, November 1998.

[KKP96] M. Kohlhase, S. Kuschert, and M. Pinkal. A type-theoretic semantics for $\lambda$-DRT. In P. Dekker and M. Stokhof, editors, *Proceedings of the Tenth Amsterdam Colloquium*, Amsterdam, 1996. ILLC.

[Kus00] S. Kuschert. *Dynamic Meaning and Accommodation*. PhD thesis, Universität des Saarlandes, 2000. Thesis defended in 1999.

[Mon73] R. Montague. The proper treatment of quantification in ordinary English. In J. Hintikka e.a., editor, *Approaches to Natural Language*, pages 221–242. Reidel, 1973.

[Mus94] R. Muskens. A compositional discourse representation theory. In P. Dekker and M. Stokhof, editors, *Proceedings 9th Amsterdam Colloquium*, pages 467–486. ILLC, Amsterdam, 1994.

[Mus95] R. Muskens. Tense and the logic of change. In U. Egli et al., editor, *Lexical Knowledge in the Organization of Language*, pages 147–183. W. Benjamins, 1995.

[Mus96] R. Muskens. Combining Montague Semantics and Discourse Representation. *Linguistics and Philosophy*, 19:143–186, 1996.