Evaluation of Previous Work

The FRACAS Consortium

Robin Cooper, Dick Crouch, Jan van Eijck,
Chris Fox, Josef van Genabith, Jan Jaspars, Hans Kamp,
David Milward, Manfred Pinkal, Massimo Poesio, Steve Pulman

FraCaS

A Framework for Computational Semantics
LRE 62-051

Deliverable D13

January 15, 1996

A Framework for Computational Semantics

CWI Amsterdam

University of Edinburgh
Centre for Cognitive Science and Human Communication Research Centre

Universität des Saarlandes
Department of Computational Linguistics

Universität Stuttgart
Institut für Maschinelle Sprachverarbeitung

SRI Cambridge

For copies of reports, updates on project activities and other FRACAS-related information, contact:

The FRACAS Project Administrator
University of Edinburgh
Centre for Cognitive Science
2 Buccleuch Place
Edinburgh EH8 9LW, Scotland, UK
fracas@cogsci.ed.ac.uk

Copies of reports and other material can also be accessed via anonymous ftp from `ftp.cogsci.ed.ac.uk`, directory `pub/FRACAS`.

Contents

Evaluation of Previous Work

The FRACAS Consortium

Robin Cooper, Dick Crouch, Jan van Eijck,
Chris Fox, Josef van Genabith, Jan Jaspars, Hans Kamp,
David Milward, Manfred Pinkal, Massimo Poesio, Steve Pulman

Abstract

In this document we discuss and evaluate some relevant recent research which aims at providing strategies for a general framework for computational semantics. The approaches considered can be broadly classified into three groups: *operators*, *formal specification* and *semantic metatheory*.

Chapter 1

Operators

1.1 Introduction

Briefly and informally the (*semantic* or *syntactic*) *operators* approach is based on the following idea: unlike in the traditional setup where syntactic and semantic representations are related to one another in a fairly *direct* fashion (e.g. in terms of associating syntactic composition rules with corresponding semantic rules) in the operator approach syntactic and semantic representations are related *indirectly* in terms of constructor operators. The operations associated with the operators then define the particulars of a syntax-semantics interface while the operators themselves are hoped to capture the more abstract and general properties of an interface or interfaces in general. This manner of indirection allows one to go in one of two ways: either a single syntactic representation is related to several semantic representations in different semantic formalisms or a single semantic representation is related to a number of syntactic representations in different syntactic formalisms. The potential or claimed benefits of such approaches are: first, achievement of greater modularity in grammar engineering; second, a factorization between what is generic and what is particular in syntax-semantics interfaces; third, a handle on the comparison between different semantic (and syntactic) formalisms. Below we concentrate on the work by [Johnson and Kay, 1990] relating different semantic representations to a single syntactic representation, i.e. treating semantics as an abstract data type (ADT) from the point of view of syntax, and the work by Pinkal & Millies on relating a single semantic representation to different syntactic representations, i.e. treating syntax as an ADT from the point of view of semantics, and evaluate the approaches in question.

1.2 Semantic Operators

In this section we discuss the semantic operator approach developed by [Johnson and Kay, 1990]. In this approach a single syntactic representation with Montague-type-theory-style semantic decorations is related to Predicate Logic, DRT and Situation Semantic (inspired) representations in terms of a set of semantic constructor operators. The set of operators includes "primitives" like `compose` and `subordinate` constructors where the former introduces ordering (precedence) information while the latter introduces quantificational and anaphoric subordination relations. In the case of the Situation Semantics and DRT representations the constructor operations implement the threading approach discussed in [Johnson and Klein, 1986].
In order provide some of the background that is assumed in the [Johnson and Kay, 1990] paper we first discuss the threading approach to constructing DRSs developed in [Johnson and Klein, 1986] and briefly and informally relate it to Dynamic Predicate Logic [Groenendijk and Stokhof, 1991].
We then present present the semantic operators approach and evaluate its potential usefulness with respect to establishing a general framework for computational semantics.

1.2.1 Johnson & Klein, 1986

The fundamental tenet in dynamic semantics is that the meaning of a linguistic expression is explicated in terms of a relation between preceding and following context.

(1.1) *Preceding Context* | α | *Following Context*

Superficially, dynamic semantic theories vary considerably with respect to how this intuition is spelled out: context can be modelled as a semantic object (e.g. sets of *world-sequence* pairs) or as syntactic representations (e.g. DRSs). In any case, dynamic meaning is defined in terms of the change that is brought about by some linguistic constituent to whatever form the preceding context takes to yield a following context.[1]

[1] Syntactic and semantic representations of context can usually be set into correspondence via the interpretation of the syntactic representations. Take e.g. the original formulation of the DRS construction algorithm. The central idea there is to transform a given context DRS K into a new context K' through incorporation of a new sentence s. Formally this can be described as associating with each sentence s a relation $\mathcal{C}$ (or if you want a function $\mathcal{C}$ which is both partial and multi-valued) which transforms DRSs into DRSs. $\mathcal{C}$ can then be paired with a relation $\mathcal{C}'$ from semantic objects to semantic objects (say sets of world-assignment pairs to sets of world-assignment pairs) such that if X and Y are such sets, then $\mathcal{C}'(s)(X) = Y$ iff there are DRSs K and K' such that K represents X, K' represents Y and $\mathcal{C}(s)(K) = K'$. In a computational setting, it goes without saying, we need to work with syntactic representations.

In the original formulation of DRT ([Kamp, 1981],[Kamp and Reyle, 1993]), e.g., the dynamics is defined in terms of a DRS construction procedure which takes an incoming context in the form of a DRS and successively transforms it into a DRS representation resulting from the incorporation of some linguistic constituent into the context. The resulting representation then serves as the preceding context to the next linguistic constituent. Here context update is unpacked in terms of a computational metaphor as operations on context representations.
The approach to DRS construction presented in [Johnson and Klein, 1986] is based on a similar computational metaphor. It differs from the original DRS construction procedure in that it is specified declaratively and in that it does not involve destructive operations. In order to achieve this the basic left-to-right dependencies encoded in the DRS construction procedure have to be explicitly coded in the grammar. In the case at hand this is implemented in terms of the threading technique from logic programming where difference list representations are threaded through syntactic representations. The difference list representations separate incoming (i.e. preceding) from outgoing (following) context representations and the *difference* between the two encodes the semantic contribution of some constituent.
On a naive view of discourse phenomena the approach can be illustrated as follows:

(1.2) $\emptyset$ A representative $\{f\}$ called $\{f\}$ a meeting. $\{f,n\}$ She $\{f,n\}$ chaired $\{f,n\}$ it. $\{f,n\}$

Initially we are presented with the null-context $\emptyset$ and after processing the first indefinite NP the null-context has been updated with a discourse referent f which is made available for further reference by anaphoric elements to the right. The semantic contribution of the indefinite NP can be characterised in terms of the difference between the incoming null-context $\emptyset$ and the outgoing context $\{f\}$, informally $\{f\} - \emptyset = \{f\}$. The second indefinite NP introduces a further discourse referent n and we have $\{f,n\} - \{f\} = \{n\}$. f and n can then be picked up by the personal pronouns in the second sentence.
An evolving discourse context can be modelled by a series of equations relating incoming and outgoing contexts:

(1.3) $C \mid$ a representative $\mid C \cup \{f\}$

$$phon(x) = \langle a\ representative \rangle \wedge con_{in}(x) = C \wedge con_{out}(x) = C \cup \{f\}$$

$$\begin{bmatrix} \textsc{phon} & \langle a\ representative \rangle \\ \textsc{con}_{in} & C \\ \textsc{con}_{out} & C \cup \{f\} \end{bmatrix}_x$$

Anaphors require that some suitable discourse referent be available in the incoming discourse

(1.4) $C \mid \text{she} \mid C$ iff $f \in C$

$$(phon(x) = \langle she \rangle \wedge con_{in}(x) = con_{out}(x)) \leftrightarrow f \in con_{in}(x)$$

$$\begin{bmatrix} \text{PHON} & \langle she \rangle \\ \text{CON}_{in} & C \\ \text{CON}_{out} & C \end{bmatrix}_x \text{ iff } f \in C$$

The basic idea is that a discourse is assigned a sequence of discourse contexts which can be viewed as a stream of discourse markers where the equations relating incoming and outgoing contexts act as operators on the stream. A discourse is well-formed if all the equations determining the discourse contexts admit of a simultaneous solution (or solutions in case of ambiguous discourses). Sets of discourse referents are unstructured objects. In order to account for anaphoric and quantificational structure context is modelled not by sets of reference makers but entire DRSs which are threaded through syntactic representations. Superordination relations ([Kamp and Reyle, 1993]) are represented in terms of list structures where depth of embedding[2] corresponds to superordination:

(1.5) `[K1,K2,K3,K4]`

where for each $\texttt{K}_i$ and $\texttt{K}_j$ such that $i \leq j$ $\texttt{K}_j$ is superordinate to $\texttt{K}_i$ etc. and where discourse referents in *complex* conditions in $\texttt{K}_j$ are anaphorically opaque to discourse referents in $\texttt{K}_i$.
Context change is effected in terms of updates on representations. Updates may

- open or close spaces
- add reference makers and conditions to spaces
- simply pass on the context unchanged (e.g. in the case of anaphors subject to the condition that the context furnishes an antecedent of the required type)

In the following we will briefly illustrate each type of update. The lexical entries associated with the determiners *a* and *every* are[3]

[2]Note that lists are recursive objects of the form `.(K1,.(K2,.(K3,.(K4,[]))))`.

[3]Note that we slightly rewrote the entries from the original PrAtt (Prolog with Attributes) into the more familiar PATR-II notation.

(1.6)
```
det(Det) --> [a],
        {Det:sem:in === Det:sem:res:in,
         Det:sem:res:out === Det:sem:scope:in,
         Det:sem:scope:out === Det:sem:out}.

det(Det) --> [every],
        {Det:sem:in === DetSemIn,
         Det:sem:res:in  ===[[]|DetSemIn],
         Det:sem:res:out === DetSemResOut,
         Det:sem:scope:in  === [[]|DetSemResOut],
         Det:sem:scope:out === [Scope,Res|[Current|Super]],
         Det:sem:out === [[(Res ==> Scope)|Current]|Super]}.
```

Determiners *determine* quantificational and anaphoric structure. The lexical entry for the indefinite *a* simply passes on the incoming discourse context as the in-context of its restrictor, takes the out-context of the restrictor and passes this on as in-context of its scope and finally equates the out-context of the scope with the out-context of the indefinite. This means that updates of the incoming discourse context in the restrictor and scope parts of the determiner are simply passed on and hence transparent to the outgoing discourse.
The entry for the universal *every* introduces two new subspaces into the representation: a restriction space and a scope space. Initially these subspaces are empty `[]`. The restriction space is superordinate to the scope space. Later they are picked up and added to the resulting representation as an externally quantificationally and anaphorically opaque structure `(Res ==> Scope)`.
Lexical entries for nouns and verbs simply add conditions to the currently open space:

(1.7)
```
n(N) --> [meeting],
        {N:syn:index === n,
         N:sem:in  === [Current|Super],
         N:sem:out === [[n,meeting(n)|Current]|Super]}.
```

while anaphors check superordinate spaces for suitable antecedents and provided some suitable antecedent can be established in the discourse context pass on the context unchanged:

(1.8)
```
np(NP) --> [she],
        {NP:sem:in === SemIn,
         member(Space,SemIn),
         member(f,Space),
         NP:sem:in  === NP:sem:scope:in,
         NP:sem:out === NP:sem:scope:out}.
```

Note that there are some striking parallels between this way of *syntactically* modeling discourse context in a computational setting and *non-representational* approaches that unpack discourse context in term of semantic objects as in e.g. Dynamic Predicate Logic (DPL) [Groenendijk and Stokhof, 1991]. Both

are based on the relational view of meaning as a relation between preceding and following context. In the syntactic approach meaning is conceived of as a relation between input and output DRS representations. In DPL $[\![\cdot]\!]$ is a subset of the cross product of the set of total assignments:

$$\mathcal{M} = \langle \mathcal{U}, \Im \rangle$$

$$\mathcal{G} = \{g | g : Var \rightarrow \mathcal{U}\}$$

$$[\![\cdot]\!] \subseteq \mathcal{G} \times \mathcal{G}$$

The interpretation of a formula φ is defined in terms of a set of pairs of assignments g and h such that $\langle g, h \rangle \in [\![\varphi]\!]$ iff h is a possible output of φ evaluated with g as input assignment.
To give a few examples: the existential quantifier is externally dynamic, i.e. it binds occurrences of the variable quantified over beyond its syntactic scope. This is achieved in terms of passing on the input context in the form of the assignment g plus any changes (here changes with respect to g) effected by the quantifier and what is in the scope of the quantifier, i.e. φ, as output assignment h to subsequent formulae:

$$[\![\exists x \varphi]\!] = \{\langle g, h \rangle | \exists k : k[x]g \ \& \ \langle k, h \rangle \in [\![\varphi]\!]\}$$

Note that this corresponds exactly to what happens in the syntactic version of discourse modeling in (1.6) above where in the case of the indefinite the input context in the form of a DRS is passed on through the restrictor and scope parts of the quantifier where it may be updated, i.e. where it may collect further discourse referents and conditions, and it is then the such updated version that is passed on to subsequent linguistic constituents.
The dynamic universal quantifier is externally static and internally dynamic.

$$[\![\forall x \varphi]\!] = \{\langle g, h \rangle | g = h \ \& \ \forall k : k[x]g \ \exists j : \langle k, j \rangle \in [\![\varphi]\!]\}$$

The input context g is simply passed on as output context $g = h$ and any internal updates $k[x]g$ and $\langle k, j \rangle \in [\![\varphi]\!]$ are thus anaphorically and quantificationally opaque to subsequent constituents. Again this is exactly what happens in the syntactic version in (1.6) above. There we simply add `Res` and `Scope` spaces to the currently active DRS space and in last two equations of the lexical entry for *every* retrieve those spaces and ensure that they (plus, of course, the input DRS representation) are passed on to subsequent constituents in such a way that the `(Res ==> Scope)` structure is anaphorically opaque to subsequent constituents.
To conclude this section we give reformulations of the example entries in (1.6), (1.7) and (1.8) above as simply Prolog clauses, as before in DCG notation but

this time without the equations in curly brackets, and with difference list `In-Out` representations of incoming and outgoing contexts. This facilitates comparison with the [Johnson and Kay, 1990] paper and shows even more clearly that what the [Johnson and Klein, 1986] paper presents is in fact a definite clause axiomatisation of context modeling and DRS construction.

(1.9)
```
det(sem: SemIn - SemOut,
    res: SemIn - SemResOut,
    scope: SemResOut - SemOut)
    --> [a].

det(sem: SemIn - [[(Res ==> Scope)|Current]|Super],
    res: [[]|SemIn] - SemResOut,
    scope: [[]|SemResOut] - [Scope,Res|[Current|Super]])
    --> [every].

n(sem: [Current|Super] - [[n,meeting(n)|Current]|Super])
    --> [meeting].

np(sem: SemIn - SemOut
   scope: SemIn - SemOut)
    --> [she],
        {member(Space,SemIn),
         member(f,Space)}.
```

A complete listing of the program is given in the appendix.

1.2.2 Johnson & Kay, 1990

A number of approaches have been pursued in the construction of syntax-semantics interfaces. The more prominent ones are probably:

- the rule-to-rule setup where each syntactic composition rule is paired with a corresponding semantic composition rule which for every local tree determines the semantics of the mother constituent by composing the semantic representations of its immediate daughter constituents
- type-driven composition where the composition process for a set of semantic representations of (not necessarily immediate) daughter constituents is dictated by the type requirements of the constituents involved
- construction algorithms like for example the DRS construction algorithm which transforms local or non-local subtrees (non-local subtrees are trees which do not exactly correspond to a single syntactic composition rule application) into semantic representations
- the use of so called *glue-languages* like e.g. resource sensitive logics to model semantic composition

Note that often these seemingly different types of interfaces are in fact not mutually exclusive: a construction algorithm defined on local trees can be translated into a rule-to-rule setup; implicative premisses in resource sensitive glue-language proofs function in a way similar to types in type-driven composition schemes etc. Anyway, the semantic operator approach presented in [Johnson and Kay, 1990] is probably best seen as a variation on the traditional rule-to-rule setup where unlike in the standard setup where each syntactic composition rule is paired with a semantic composition rule expressed in terms of the semantic representation language of choice (e.g. type theory) now each syntactic composition rule is paired with a semantic composition rule expressed in terms of (supposedly) general semantic constructor operators. This manner of indirection allows us to pair different operations with the operators and thus e.g. to map a single syntactic representation into corresponding representations in different semantic representation formalisms. In the case at hand three sets of operations associated with the operator set map into Predicate Logic (PL), Discourse Representation Theory (DRT) and Situation Semantic (SitSem) representations. In a sense, in this approach semantic representations are viewed as abstract data types (ADTs) from the point of view of syntax where the construction operators provide limited and indirect access to the resulting representations. The potential or claimed benefits of such approaches are: first, achievement of greater modularity in grammar engineering; second, a factorization between what is generic and what is particular in syntax-semantics interfaces; third, a handle on the comparison between different semantic (and syntactic) formalisms.

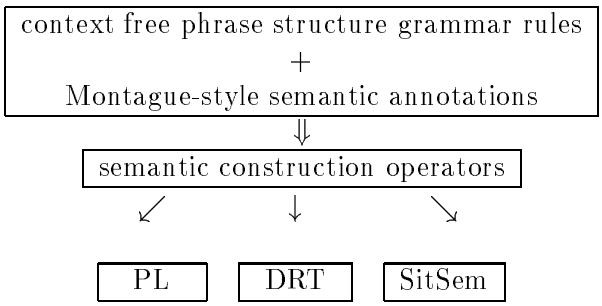
Above we said that the semantic operator approach by [Johnson and Kay, 1990] can be seen as a variation of the simple rule-to-rule setup where syntactic rules are paired with semantic rules composed of operator expressions. Actually, the picture is slightly more complicated: in addition to syntactic phrase structure rules we also have Montague-style semantic annotations for those rules. The annotations are given in the manner of the semantic composition rules in [Pereira and Shieber, 1987] where λ-abstraction and β-conversion are approximated[4] in terms of Prolog variables and unification. Types are implicit and intensionality is not accounted for. The semantic annotations are mainly responsible to link the semantic argument positions of e.g. verbs with the variables or constants associated with the semantic representations of their syntactic arguments and

[4] λ-abstraction and β-conversion are only *approximated* because in the Prolog representations variables are quantified universally with wide scope over a clause and hence an approach that does not involve copying of variables during β-reduction will not be able to correctly model a situation like

$$(\lambda x.(xc_1)(xc_2))(\lambda y.y)$$

which should reduce to c_1c_2 where c_1, c_2 are constants such that $c_1 \neq c_2$.

restrictor and scope parts of quantifiers to semantic representations of the respective syntactic restrictor and scope contributions etc. They also provide the hooks into which the semantic operator expressions link. The basic picture we get is thus:



For the simple fragment presented in [Johnson and Kay, 1990] the authors employ a set of seven constructor operators. No claims are made about whether this set is sufficient for general purposes. We will first briefly present the operators and then discuss the 3 sets of operations (PL, DRT, SitSem) associated with those operators.
The first two operators are fairly trivial. The first one relates some internal representation (e.g. one that is cluttered by threading information) to a printable output representation. The second one links Montague-style lexical semantic annotations to whatever semantic formalism is used:

- `external(S,SF)`: maps an internal semantic representation `S` into some external representation `SF`
- `atom(S,Prop)`: maps an atomic proposition `S` into the required semantic representation `Prop`

Two operators *merge* semantic representations (conjunctively): the first operator is not sensitive to sequential ordering information while the second is:

- `conjoin(S1,S2,S12)`: `S12` is constructed by conjoining `S1` with `S2`
- `compose(S1,S2,S12)`: `S12` is `S1` followed by `S2`

The `subordinate/3` operator introduces quantificationally and anaphorically subordinate subspaces into the representation:

- `subordinate(Sub,SubName,S)`:`Sub` is `S` plus a new quantificationally and anaphorically subordinate expression `SubName`

Finally, two operators deal with basic anaphoric phenomena. The first introduces an antecedent index expression into the representation; the second implements tracking of accessible antecedents:

- `new_index(I,S)`: `I`, a new referential index for non-anaphoric NPs, is introduced into `S`
- `accessible_index(I,S)`: `S` contains accessible index `I`

Three sets of constructor operations are associated with the operators listed above. The definitions for the DRS-constructor operations are:

(1.10)
```
external([[]]-[S],S).

atom(P,[B|Bs]-[[P|B]|Bs]).

conjoin(P1,P2,P) :-
    compose(P1,P2,P).

compose(B0s-B1s,B1s-B2s,B0s-B2s).

subordinate([[]|B0s]-[B|B1s],B,B0s-B1s).

new_index(Index,C) :-
    atom(i(Index),C).

accessible_index(Index,Bs-Bs) :-
    member(B,Bs),
    member(i(Index),B).
```

The `external/2` operation sets up an internal difference list representation to implement threading of input and output contexts `In-Out` as in [Johnson and Klein, 1986] discussed above. Initially the input context is empty `[]`. The output context contains the representation `S` which constitutes the external representation. The `atom/2` operation adds an atomic proposition `P` to the currently active space `B` of an input context DRS `[B|Bs]` resulting in a `[[P|B]|Bs]` output context. `conjoin/3` is defined in terms of `compose/3` and `compose/3` is simply difference list concatenation. The `subordinate/3` constructor takes an input-output representation `B0s-B1s` and adds a new empty context `[]` to the input `B0s` and picks it up as `B` in the corresponding output `[B|B1s]`. `new_index/2` updates the input-output pair with a new index `i(Index)` while `accessible_index/2` requires that for the input context to be equal to the output context it contain an accessible index `i(Index)`.
The SitSem-constructor operations are variations on the DRS-constructor operations:

(1.11)
```
external(@([Sit],[],Is), Sit:Is) :-
```

```
    gensym(Sit).

atom(P,@([Sit|_],Is,[(Sit:P)|Is])).

conjoin(I1,I2,I12) :-
    compose(I1,I2,I12).

compose(@(Ss,I0s,I1s),
        @(Ss,I1s,I2s),
        @(Ss,I0s,I2s)).

subordinate(@([Sit|Sits],I0s,I1s),Sit,@(Sits,I0s,I1s)) :-
    gensym(Sit).

new_index(Index,S) :-
    atom(i(Index),S).

accessible_index(Index,@(Ss,Is,Is)) :-
    member(Sit:i(Index),Is),
    member(Sit,Ss).
```

Internal representations `@(Sits,InfonsIn,InfonsOut)` are constructed with a `@` functor taking a list of situations (represented by integers[5]), a list of input infons representing input context and a list of output infons representing output context. The `atom/2` constructor takes a situation `Sit` of the list of situations and adds the infon `(Sit:P)` constructed from the atomic proposition `P` to the input context `Is` to construct an output context `[(Sit:P)|Is]`. The other definitions are the same as in case of the DRS-constructor operation definitions modulo the `@(Sits,InfonsIn,InfonsOut)` representations. Note that this way of setting things up simply buys the declarative specification of the DRT dynamics originally developed in the [Johnson and Klein, 1986] paper discussed above into the Situation Semantics account.
The PL-operations are mostly degenerate definitions[6] and are simply listed without further comment:

(1.12)
```
external(P,P).

atom(Prop,Prop).

conjoin(P,Q,P&Q).

compose(P,P,P).

subordinate(Sub,Sub,_).

new_index(_,_).
```

[5] Actually, by the result of a `gensym/1` operation.
[6] Note that here we have removed two typos in the original paper.

```
accessible_index(_,_).
```

Finally we will take a look at how the constructor operators are actually employed in the grammar. First we discuss lexical entries for nouns, pronouns and quantifiers just as we did in (1.6), (1.7), (1.8) and (1.9) above. Entries for nouns are listed as facts specifying category, string and Montague-style semantics information:

(1.13)
```
noun(meeting,X^meeting(X)).
noun(representative,X^representative(X)).
```

The lexical production expanding nouns is paired with an expression composed of semantic operators:

(1.14)
```
n(X^S) --> [Noun],
    {noun(Noun,X^Pred),
     new_index(X,S1),
     atom(Pred,S2),
     compose(S1,S2,S)}.
```

Assume that the DRS-construction operations are in force and that we expand `n(X^S)` into the string `meeting`. The `new_index/2` operator will construct a difference list representation `S1 = [B0|B0s]-[[i(X)|B0]|B0s]` which contains the index `i(X)` coindexed with the λ-variable in the lexical entry for the noun in (1.13) above. The `atom/2` constructor instantiates `S2` to `[B1|B1s]-[[meeting(X)|B1]|B1s]`. Finally the `compose/3` operator will merge `S1` and `S2` into `[B0|B0s]-[[meeting(X),i(X)|B0]|B0s]`. Thus (1.14) *partially evaluated* with respect to the lexical entry for `meeting` and the semantic operator expressions results in

(1.15)
```
n(X^[B0|B0s]-[[meeting(X),i(X)|B0]|B0s]) --> [meeting].
```

Note that modulo index representation this corresponds exactly to the Prolog term reformulation of the [Johnson and Klein, 1986] representation as given in (1.9) above.
The corresponding Situation Semantics inspired representation would be something like:

(1.16)
```
n(X^@([S1|Sits],[I0|I0s],[[S1:meeting(X),S1:i(X)|I0]|I0s])
    --> [meeting].
```

The lexical entries for quantifiers are

(1.17)
```
determiner(a,Res^Scope^S) :-
    conjoin(Res,Scope,S).
determiner(every,Res0^Scope^S) :-
    compose(S1,S2,S),
    subordinate(Res,ResName,S1),
    compose(Res0,Res1,Res),
```

```
    subordinate(Scope,ScopeName,Res1),
    atom(ResName ==> ScopeName,S2).
```

Partial evaluation of the clause for the indefinite with respect to its body and the DRT operator definitions yields:

(1.18)
```
determiner(a,B0s-B1s^B1s-B2s^B0s-B2s).
```

which can easily be seen as a variant of the corresponding entry in (1.9) here repeated as (1.19) which implements threading through restrictor and scope representations

(1.19)
```
det(sem: SemIn - SemOut,
    res: SemIn - SemResOut,
    scope: SemResOut - SemOut)
    --> [a].
```

Now it will not come as a big surprise that the complicated and at least at first sight fairly non-transparent operator annotations in the body of the clause defining the universal quantifier in (1.17) above amount to a notational variant of the corresponding entry in (1.9) here repeated as (1.20):

(1.20)
```
det(sem: SemIn - [[(Res ==> Scope)|Current]|Super],
    res: [[]|SemIn] - SemResOut,
    scope: [[]|SemResOut] - [Scope,Res|[Current|Super]])
    --> [every].
```

The constructor definitions for the universal in (1.17) require that `S` be the composition (i.e. difference list concatenation) of `S1` with `S2` where `Res` is `S1` plus a subordinate structure `ResName` where `Res` itself is the composition of `Res0` with `Res1` such that `Scope` is `Res1` plus the new subordinate structure `ScopeName`. Finally, the two subordinate structures `ResName` and `ScopeName` are collected and `S2` is updated with `ResName ==> ScopeName`. Partial evaluation yields:

(1.21)
```
determiner(every,[[]|B0s]-B1s
                 ^[[]|B1s]-[ScopeName,ResName,B|Bs]
                 ^B0s-[[ResName==>ScopeName|B]|Bs]).
```

As was the case in the [Johnson and Klein, 1986] paper, the grammar and the operator definitions constitute a definite clause axiomatization of a system with PL, DRT an SitSem inspired semantic representations which can be directly executed by the Prolog proof procedure. If the grammar is not interpreted directly by the Prolog proof procedure, if e.g. the grammar is interpreted by a bottom-up left-corner inference procedure special coroutining techniques (e.g. the SICStus `freeze/2`) may be employed to ensure termination of anaphor tracking in the `member/2` calls in `accessible_index/2` in (1.10) and (1.11):

(1.22)
```
accessible_index(Index,Bs-Bs) :-
```

```
    member(B,Bs),
    member(i(Index),B).

accessible_index(Index,@(Ss,Is,Is)) :-
    member(Sit:i(Index),Is),
    member(Sit,Ss).

member(X,[Y|_]) :-
    freeze(Y,X=Y).
member(X,[_|Y]) :-
    freeze(Y,member(X,Y)).
```

A complete listing of the grammar and the operator definitions[7] is given in the appendix. [Johnson and Kay, 1990] furthermore show how a basic Cooper-store mechanism to handle quantifier scope effects can be integrated into the grammar in a simple way which is independent off and compatible with the PL, DRT and SitSem representations. This version is also listed in the appendix.

1.2.3 Conclusion

The potential or claimed benefits of the semantic operator approach are: first, achievement of greater modularity in grammar engineering; second, a factorization between what is generic and what is particular in syntax-semantics interfaces; third, a handle on the comparison between different semantic (and syntactic) formalisms.
We will discuss each in turn: first, effectively Johnson & Kay treat *semantics* as an ADT from the perspective of syntax. However, it seems that their attempt does not insulate the grammar writer from the details of the possible semantic target representations and also internal representations like for example the context representation in terms of threading of difference lists, about which s/he must still know a great deal. Consider again the following example showing the lexical entry for *every*:

(1.23)
```
determiner(every,Res0^Scope^S) :-
    compose(S1,S2,S),
    subordinate(Res,ResName,S1),
    compose(Res0,Res1,Res),
    subordinate(Scope,ScopeName,Res1),
    atom(ResName ==> ScopeName,S2).
```

The calls to the `compose/3` operator decompose semantic representations and set up the difference list threading in the case of the DRT and SitSem representations. The `subordinate/3` operators hook into the decomposed representations and the `atom/2` call fixes the ouput semantic representation. In our view it seems very unlikely that a grammar writer will come up with a correct definition

[7] Some typos are removed from the code given in the original paper.

of (1.23) solely on the basis of the informal presentations of the operators in the description of the Johnson & Kay approach given above without a detailed knowledge of the desired target representations, such as e.g. the internal DRT representation:

(1.24)
```
determiner(every,[[]|B0s]-B1s
                 ^[[]|B1s]-[ScopeName,ResName,B|Bs]
                 ^B0s-[[ResName==>ScopeName|B]|Bs]).
```

As far as a productive division of labour is concerned, it also seems wrong to require the grammar writer to know several semantic theories in detail. What the Johnson & Kay approach does show, however, is that certain tools like quantifier scope mechanisms or simple dynamic effects can be integrated with different semantic theories in a modular fashion.

The Johnson & Kay operator approach constitutes an interesting approach towards a factorization between what is generic and what is particular in syntax-semantics interfaces. The `conjoin`, `compose`, `subordinate`, `new_index` and `accessible_index` operators do seem to capture general and fairly transparent relations in syntax-semantics interfaces. What is problematic is that some of the combinations of the constructor relations in the rule annotations as e.g. in (1.23) this transparency is almost completely lost and the rule annotations are largely motivated in terms of the desired target representations which is precisely what they should be abstracting from.

We also feel that in some cases the particular choice of operators and associated operations seem to be on the wrong level expressing generalizations to the worst case. In (1.23) the two calls to the `subordinate/3` constructor are needed to introduce some hierarchical structure into the representation, which is used by DRT and imported into SitSem (but not by PL) for the purposes of accessibility constraints. Note also that one has to introduce names for the restriction and scope of the quantifier (`ResName`,`ScopeName`) because Situation Semantics distinguishes a propositional content from the situation in which it is true (although PL does not).

The operator approach could provide some handle on the comparison between different semantic (and syntactic) formalisms. Johnson & Kay do not pursue this matter other than indirectly by showing that different formalisms are compatible with certain tools (e.g. Cooper store, context threading).

The general problem with the Johnson & Kay solution seems to be a somewhat inappropriate view of the division of labour between syntax and semantics. Semantic construction, e.g., the combination of constituent representations into larger units, is still basically taken to be part of the grammar. Since grammar has to bear the burden of directly generating target representations of different kinds, a considerable amount of semantic knowledge is necessary to design the correct grammar rules, contrary to Johnson and Kay's intentions. More

recent developments in the area of semantic formalisms like λ-DRT ([Millies and Pinkal, 1993], [Kuschert, 1995]), CDRT ([Muskens, 1994]), and STDRT ([Barwise and Cooper, 1993]) suggest a different solution, i.e., a modularization within semantics: logic, especially one or the other form of the λ-calculus, can do the task of semantic construction. The semantic operations used (like function application and quantifier storage) can be given a general definition independent of the target representation, which may be Predicate Logic, DRT, or a form of Situation Semantics while the target formalism is encoded in the lexical representations. If the grammar writer is supposed to provide semantic annotations to the syntactic rules, as in the Johnson & Kay scenario, s/he gets the (denotationally interpreted) semantic operators for free; their application is straightforward and really independent of specific semantic formalisms.

1.3 Syntactic Operators

1.3.1 Pinkal & Millies, 1993

The semantic interpretation system SCOLD (Syntax-sensitive Computation of Logical Descriptions) of [Millies and Pinkal, 1993] makes use of the modified concept of semantic operators briefly outlined in the closing section of our comments on the Johnson & Kay proposal above. It uses a small set of semantic operators, among them *generalized functional application*, which is closely related to functional composition, and allows a straightforward analysis of different phenomena which are problem cases for standard functional application. The operators allow a general formulation of semantic interpretation independent of the semantic target representations. The semantic representation language which is actually used in the implemented system is λ-DRT.
The essential feature of the approach is its special conception of the syntax-semantics interface. Syntax does neither contain semantic information nor instructions to build semantic representations. Semantic representations are built by an interpretation component which inspects the syntax to extract information necessary for the selection of the appropriate semantic operation. Two levels are separated:

- a core mechanism for semantic construction which is largely independent of the grammar formalism used in the syntax
- a set of abstract syntax interface predicates, which are defined differently for different grammar formalisms.

The reason to have the abstract interface predicates is to specify the functional role of syntax in the process of building semantic representations while hiding

the details of how the requisite information is realized in a separate interface component. Thus, the system is flexible: it allows to extract arbitrary information from syntax, encoded in a variety of different possible notations, and it does so in a controlled way through the interface.
Semantic operators work on semantic objects which are associated with particular syntactic objects, e.g. nodes in a phrase structure tree in GB ([Chomsky, 1981]), values of certain syntactic features in HPSG ([Pollard and Sag, 1994]), or parts of f-structure in LFG ([Kaplan and Bresnan, 1982]). The application of the semantic operators to their arguments is sensitive to the particular syntactic environment of the associated node. For example, there are three basic semantic operators called *compose*, *predication*, and *abstract*, which turn out to be sufficient for local semantic interpretation. Semantic interpretation proceeds by collecting the meanings of some set of nodes (the current *local domain*) and freely applying the operators to them. In this process, the operators "consume" their arguments and "produce" a result, until only one semantic object is left, which is then assigned to some node. The local domain is given by the syntax-interface predicate *local-domain** (predicates from the syntax interface are written in italics and marked by the diacritic "*"). The final result will be associated with the root of the domain, if this is a *compatible-type-assignment** to that node. A syntactic node may be compatible with a range of types.
Non-local semantic phenomena like quantifier scope or anaphora are also treated. Scope readings are generated by a version of Nested Cooper Storage ([Keller, 1988]): there are semantic constructors *store* and *quantify-in* and interface predicates *non-locally-interpretable**, *quant-in-permissible**, and *constraints-on-scope**. The meaning of any constituent declared non-locally interpretable may be stored, and later quantified into a meaning associated with a node where this is permissible, provided that the scope constraints hold. Anaphoric binding is implemented as the unification of discourse markers in semantic structure, provided that the nodes from where the markers originate fulfill *constraints-on-binding**.
The core interpretation component simply consists in a recursive postorder traversal of the root domain and assignment of the result of the evaluation to the root, followed by enumerating binding possibilities. Its Prolog implementation looks as follows (`Lob` is mnemonic for "linguistic object"):

(1.25)
```
semantics(Lob, Store) :-
  evaluation(Lob, Store),
  binding(Lob).

evaluation(Lob, Store) :-
  local_domain(Lob, Domain0),
  traversal(Domain0, Domain),
  evaluate(Domain, Lob, Store),
```

```
    compatible_type_assignment(Lob).

traversal([], []).
traversal([Lob|Domain1], [Lob/Store|Domain]) :-
    evaluation(Lob, Store),
    traversal(Domain1, Domain).
```

The definitions of the interfaces to some grammar formalisms are given in more detail, below.

The GB Interface The GB-interface is written so as to enable direct interpretation of S-structure. (LF input is likewise possible; in fact, the semantic construction task is simpler since no storage mechanism is required. In GB the local domain contains the immediate subconstituents of a node, and as a consequence, the operators can only apply to semantic objects associated with sister nodes:

(1.26)
```
local_domain(Lob, Domain) :-
    feature(dtrs, Lob, Domain).
```

Binary branching trees are assumed. The system non-deterministically selects one element from the domain as the functor, the other as the argument, returning an empty rest domain.

(1.27)
```
find_fun_arg_pair([Fun, Arg], [], Fun, Arg).
find_fun_arg_pair([Arg, Fun], [], Fun, Arg).
```

Complements have raised type, so that normally complements are functors over heads, with the exception of empty functional heads. Because semantic objects are typed, only one of the combinations is possible in each case. The result of composing functor and argument will be assigned to the mother node. An application of the move-α transformation (leaving behind a co-indexed trace) will correlate in the semantics with an abstraction operation: when a moved functor is composed with an argument, first the argument is abstracted over with the referential index of the functor, which is also accessed by an interface function. Referential indices in the syntax correspond to semantic variables, so that syntactic co-indexing guarantees that the correct variable is abstracted. For the treatment of quantifier scope, the notion of scope-bearing elements is defined in syntactic terms. Every element that takes scope may go in storage. As for retrieval, different kinds of nodes can be determined to allow quantifying-in. Adding NP to this class of constituents takes care of quantifying into term phrases:

(1.28)
```
non_locally_interpretable(Lob) :-
    scope_inducing(Lob).

quant_in_admissible(Lob) :-
```

```
    (ip(Lob) ; np(Lob)).
```

We assume that the constraint on scoping uses subjacency. As for binding, the nodes from which pronoun and antecedent originate must agree in certain features, and one must c-command the other.

(1.29)
```
constraints_on_quant(Quant, Scope) :-
  subjacent(Quant, Scope).

constraints_on_binding(Binder, Bindee) :-
  c_command(Binder, Bindee) ,
  agree_markers(Binder, Bindee).
```

In order to implement these relations, every semantic object bears a pointer to its home syntactic node. The syntactic tests involved are the same as those used in syntax, there is no duplication of syntactic information for semantic purposes.

The HPSG Interface The local domain of interpretation in HPSG is not the set of all daughters, as in GB, but the set of all daughters except the filler-daughter. Fillers are reconstructed by HPSG into their D-structure (or NP-structure) position, either by postulating a trace, or by transforming the lexical semantics of the verb. Therefore, they are never interpreted in their surface position. Complements are type-raised in semantics, and because of the absence of empty functional heads in our HPSG grammar, complements and adjuncts alike are uniformly treated as functors over heads. As a further difference to GB, multiply branching structures are allowed.

(1.30)
```
local_domain(Lob, []) :-
  lexical(Lob).

local_domain(Lob, [Head|Domain]) :-
  phrasal(Lob),
  head_dtr(Lob, Head),
  comp_dtrs(Lob, Comps),
  adj_dtrs(Lob, Adjuncts),
  append(Comps, Adjuncts, Domain).

find_fun_arg_pair([Arg,Fun|FLobs], Flobs, Fun, Arg).
```

The interface predicates for generating different scopings are the same as in GB, except that no syntactic constraints are assumed (since HPSG does not offer a theory for it). The interface predicate for binding employs the non-configurational notion of o-command instead of the tree-based notion of c-command. For the purpose of computing o-command, every semantic object again bears a pointer to its origin, which in this case is a position on a SUBCAT-value.

(1.31)
```
constraints_on_quant(_, _).

constraints_on_binding(Binder, Bindee) :-
  o_command(Binder, Bindee),
  agree_refos(Binder, Bindee).
```

The LFG Interface The idea for an LFG-interface is to decompose f-structures into their constituent parts. The local domain is the union of all grammatical functions governed by the PRED-value except non-thematic functions, where set-valued functions contribute all the elements in the set.

(1.32)
```
local_domain(Lob, []) :-
  semantic_form(Lob).

local_domain(Lob, Domain) :-
  pred_of(Lob, Pred),
  thematic_gfs(Lob, GFs),
  mods_of(Lob, Mods),
  union([Pred|GFs], Mods, Domain).
```

This domain is reduced non-deterministically, relying on the types of the semantic objects to drive the composition. The result is assigned to the f-structure as a whole. There is one additional complication: in order to ensure that semantic representations for the complements of a head are applied in the correct order, the semantic objects in the domain will be indexed by grammatical function. Subcategorized functions must be applied first, and in the order in which they appear in the semantic value of the PRED function. Quantifier scope will be treated analogously to GB; binding will work with the notion of f-command. The relative "flatness" of f-structures (as opposed to c-structure trees) may turn out to be an additional problem for the formulation of scope and binding constraints.

1.3.2 Conclusion

Millies & Pinkal propose a semantic interpretation system which interacts with syntax by accessing syntactic information flexibly, but in a controlled way. The semantic interpretation module can be used with different grammar formalisms (or grammatical theories) by re-defining only the syntax-interface. This supports the theoretical perspicuity of the semantic interpretation process and the syntax-semantics interaction, facilitates porting of the semantics to different grammar systems, and supports the distributed development of large NL-systems. The interface concept developed in SCOLD has been applied in the Verbmobil system ([Bos *et al.*, 1994b; Bos *et al.*, 1994a]).

Finally, however, we point out that semantics never is completely neutral with respect to the syntax formalism. At least three types of problems can be distinguished which, taken together, make syntax-semantics interdependence a much more complex field than it may appear at a first look.

- The syntactic analysis of some particular phenomenon may be radically different in the various syntax formalisms. For example, in control constructions GB theory postulates an empty element PRO in an embedded clause, whereas HPSG embeds an unsaturated verbal complement. The crucial difference is that in HPSG, a semantically relevant object occurs outside the domain of recursion. In order to accommodate the HPSG analysis to a semantic framework where control verbs take propositions as arguments, one would have to add a *control-adjustment rule* to the HPSG interface. Thus, while in the one case a semantic analysis follows naturally from a syntactic decision, in the other case an otherwise unmotivated stipulation has to be made.

- A formalism may simply not provide the syntactic information necessary to implement some constraint on interpretation. The difficulty may be "accidental" - there is no essential reason why HPSG should not be able to come up with a constraint on quantifier scope, for example. But the difficulty may also be fundamental, as is the case with the problem of re-entrancy in LFG. In order to derive the correct representations for f-structures involving e.g. control constructions, it is necessary to distinguish the controller from the controlled element. But f-structure simply does not contain this information. It is hidden in the syntax-rule annotations to which the semantic component has no access.

- The SCOLD system assumes a monostratal syntactic analysis. In the GB case, one can straightforwardly adapt the interface to interpret LF instead of S-structure, but one cannot exploit syntactic information on several levels of representation simultaneously. This may or may not be considered a sensible constraint on syntactic theory, in any case it is a major limitation on the generality of the approach.

Chapter 2

Formal Specification

2.1 Introduction

Formal specification techniques for programs are well-developed in the computer science literature. It is only recently, however, that such techniques have been considered as a possibility for making precise the relationship between different linguistic theories.
The most prominent approaches are probably algebraic approaches and approaches involving constructive type theories.
Here we will first give a brief overview of [Holt, 1993] who uses multi-sorted algebras. We will then give an example of what might be involved in giving a detailed logical treatment of predication in our own terms. We will conclude that this approach, while potentially very fruitful is also very complicated and that the returns within the life of the FraCaS project make it not feasible for us to pursue it at this stage.
We will then turn to constructive type theories which provide a means of giving formal specifications and derivations of programs [Martin-Löf, 1982; Mohring, 1986; Henson, 1989; Turner, 1991], for example. They can also be used as a formalism in which to develop theories of natural language semantics [Sundholm, 1989; Ranta, 1991; Davila-Perez, 1994; Ahn and Kolb, 1990], for example. Thus, they seem to offer the potential of allowing us to give a formal specification of a semantic theory which can then be used to derive programs that implement that theory. There are, however, some drawbacks to this approach. Constructive theories are sometimes at odds, philosophically, with the classical frameworks in which many semantic theories are developed. Also, there appears to be a significant additional over-head in devising semantic theories within constructive frameworks which might negate any benefits they have in providing a formal specification and derivation of computational implementations.

2.2 Algebraic Specification Languages

2.2.1 Overview of Holt's work

Holt first presents the case for semantic abstraction, by which he means the extraction of general systems of metatheoretical statements which could be applied to all or a subset of semantic approaches. On this view a metatheoretical specification of a particular semantic approach could be seen as inheriting from more abstract levels of specification which delineate more general characteristics which are shared by different semantic approaches. In the first part of the thesis Holt employs a formalism which derives from the algebraic specification language ASL [Sannella and Wirsing, May 1983; Sannella and Tarlecki, 1988; Sannella and Tarlecki, 1992]. A specification defines a multi-sorted algebra in terms of sorts, operations on those sorts, predicates on them and axioms which the elements of the algebra must obey. Holt define his algebras over semantic representation languages rather than the semantic objects themselves. Here is Holt's specification of a proposition algebra with an operator *prop* which forms propositions from formulae:

PROP =
sorts Form, Prop
opns truth: $\rightarrow$ Form
not: Form $\rightarrow$ Form
and: Form, Form, $\rightarrow$ Form
prop: Form $\rightarrow$ Prop
eqprop: Prop, Prop $\rightarrow$ Form
preds seq: Form, Form
axioms $\forall f \forall g$ seq(truth, eqprop(prop(f), prop(g))) $\Rightarrow$
seq(f, g) $\wedge$ seq(g, f)

Here *eqprop* expresses that two propositions are identical and *seq* holds between two formulae if the second is a logical consequence of the first. Various additional axioms can be added to this general specification to give different theories of propositions. Holt gives examples where you require commutativity and identity under logical equivalence:

PROP_COMM_CONJ = PROP +
axioms $\forall f \forall g$ seq(truth, eqprop(prop(and(f, g)), prop(and(g, f))))

PROP_COARSE = PROP +
axioms $\forall f \forall g$ seq(f, g) $\wedge$ seq(g, f) $\Rightarrow$
seq(truth, eqprop(prop(f), prop(g)))

Holt adds variables, entities and properties to the specification by introducing an abstraction operation that creates property expressions from formulae:

ABS = PROP +
sorts Var,Ent,Pty
subsorts Var $\subseteq$ Ent
opns abs: Var, Form $\rightarrow$ Pty
eqpty: Pty, Pty $\rightarrow$ Form
axioms $\forall f \forall g \forall x \forall y$ seq(truth, eqpty(abs(x, f), abs(y, g))) $\Rightarrow$
seq(f, g) $\wedge$ seq(g, f)

¿From this you can build up various specifications of the different relationships between properties and propositions according to the different semantic approaches.
In later parts of the thesis, Holt applies this approach, though not using the exact ASL type formalism, to Johnson and Kay's work on semantic operators [Johnson and Kay, 1990] and to a general specification of quantification and anaphora.

2.2.2 An algebraic view of predication

In considering whether we would take this approach to building a framework, we thought of attempting to characterize core notions of semantic theory in terms of a collection of algebras which can be related to each other. Whereas Holt does his specification at the level of semantic or logical representations, we considered the possibility of doing it at the level of semantic universes. We provide below an example of what this might look like for predication, the area which is probably the easiest to approach in this kind of way. This general presentation is derived from the more specific presentation of situation theory presented in deliverable D8.
The structures we will use will be of the form $<A$, *Sorts*, *Rlns*$>$ where

1. A is a set
2. *Sorts* is a set of unary relations on (subsets of) A, the set of *sorts* introduced in the structure
3. *Rlns* is a set of relations on A

We will refer to A as the set of OBJECTS of the structure. We will assume that the members of *Sorts* do not overlap unless explicitly stated to the contrary.
These structures can be regarded as relational structures of the standard kind, i.e. $<A$, *Sorts*$\cup$*Rlns*$>$

A predication structure has three sorts: predicates, roles for the predicates and the sort of objects which are the result of predication – we'll call these last predication objects. There is a role assignment function which tells you which roles are associated with each predicate and an appropriateness relation which tells you which assignments to the roles of a predicate are appropriate for that predicate. The main use of the appropriateness relation is to allow for the possibility that it is not the case that arbitrary assignments of objects can be made to a predicate's roles. This enables us to avoid paradoxes. Another possible use of the appropriateness relation is to handle sortal restrictions. Finally, a predication structure includes a predication operation which maps any predicate and appropriate assignment to that predicate to a predication object. Different systems make different claims about the nature of the object that results from a predication.
A *predicate structure* $< A, \{Pred, Roles, PredObj\}, \{ \Sigma, appr, * \}>$ is such that:

1. Σ:*Pred*$\rightarrow$Pow(*Roles*), the *role assignment function*, is a function which assigns a set of roles to each predicate in *Pred*.

2. *appr*, the APPROPRIATENESS RELATION, $\subseteq \bigcup_{r \in Pred}(\{r\} \times A^{\Sigma(r)})$, i.e. for the predicate r, *appr* will relate assignments to r's roles to r.

3. $*$, the PREDICATION OPERATION is a partial binary operation on A with domain

$$\{< r, f >| \; r \in Pred, appr(r, f)\}$$

To do predication in predicate calculus we need to require

1. for each $r \in Pred$, for some natural number i, r is a set of i-place sequences of members of A

2. *Roles* is the set of natural numbers

3. *PredObj* is a pair of TRUTH VALUES, represented by $\{T, F\}$

4. Σ assigns to any predicate a finite initial segment of the set of natural numbers

5. if r is a set of i-place sequences then $\mid \Sigma(r) \mid = i$

6. for any $r \in Pred$, and assignment f meeting the requirements on an appropriateness relation of a predicate structure, $appr(r, f)$ iff ran(f) is disjoint from all the sorts. I.e., an appropriate assignment is any one of the right arity which does not assign predicates, roles or truth-values as the arguments to the predicates

7. for any $r \in Pred$, r is a subset of the set of assignments which are appropriate to it.

8. for an $r \in Pred$ and appropriate f, $r * f = T$ if $f \in r$ and $r * f = F$ otherwise.

¿From this we can extract out the property of using natural numbers for the roles since this is a (rather trivial) property which is shared by a good number of approaches. Thus we can talk of a PREDICATION STRUCTURE WITH ROLE INDEXING BY THE NATURAL NUMBERS which is a predication structure meeting the following requirements.

1. *Roles* is the set of natural numbers

2. Σ assigns to any predicate a finite initial segment of the set of natural numbers

A PROPERTY THEORETIC PREDICATION STRUCTURE might then be characterized as a predication structure with role indexing by the natural numbers which meets the following requirements.

1. *PredObj* is a set of propositions. (This places no constraints on the predication structure as such but shows how it is to be connected to other structures in property theory, in particular the proposition structure with connectives and the set of true propositions.)

2. Perhaps general constraints on appropriateness which should be added here.

Appropriateness can be ARGUMENT INDEPENDENT. This means that we can define a function **type** whose domain is $\{< r, i > |\ r \in Pred, i \in \Sigma(r)\}$ such that appr(r, f) iff for each $i \in \Sigma(r)$, $f(i) \in$ type(r, i).
A predication structure with indexing by natural numbers and argument independent appropriateness can be curried.[1] We do this by adding an additional sort of predicate functions *PredFns* where there is a one-one mapping μ from *Pred* to *PredFns* such that for each predicate r in *Pred* with roles $\{0, \ldots, n-1\}$, $\mu(r) \in PredObj^{\text{type}(r,0)\ldots\text{type}(r,n-1)}$. We then require that the predicate operation $*$ obey the following additional axiom.

- For any predicate r and appropriate assignment f, $r * f = \mu(r)(f_{n-1}) \ldots (f_0)$

[1] There are presumably more general notions of order independence which do not require indexing by natural numbers and argument independent appropriateness, but it is convenient to assume them.

Predication can be structured. For this we require the following additional axiom on $*$ to be obeyed.

- $r * f = r' * f'$ iff $r = r'$ and $f = f'$

In DRT the kind of predication depends on whether you look at the models for DRSs or the DRSs themselves. In the models the standard PC kind of predication is used. In the DRS predication is used to construct conditions which are structured objects. Hence it would seem appropriate to say that DRT uses a predication structure with indexing by natural numbers, argument independent appropriateness and the structure axiom.
Situation theory uses predication with the structure axiom like DRSs, though it does not require indexing by natural numbers or argument independent appropriateness. ST has the unusual feature of using two predication structures simultaneously. One is like Property Theory in that the predication objects are propositions which take part in a proposition structure in the same way as they do in Property Theory. The predicates in this structure are called types in situation theory. The other predication structure in situation theory has infons (a kind of situation type) as its predication objects and relations (properties) as its predicates. This means that the predication objects in the infon predication structure serve as predicates in the proposition predication structure. While the formulation of DRT is in terms of single predication like the other systems, the syntactic conventions for the notation of DRSs suggests a dual predication system like ST and one might try to argue that DRT should be formulated in this way. Thus in DRT one has infon-like conditions such as

$$\text{like}(x, y)$$

and proposition-like conditions where the infon-like conditions might appear to be predicated of an eventuality (situation):

$$e : \text{run}(x)$$

In actual fact DRT treats the eventuality as an argument to the predicate. However, it is not clear that this Davidsonian approach to predication of eventualities will handle the general case (cf. the discussion of "Negative Events", D10, p.117).

2.3 Constructive Type Theory as A Framework for Computational Semantics

2.3.1 Reasons for Examining Constructive Type Theories for a Framework

If we are to find a framework for computational natural language semantics, one paradigm that suggests itself is that of constructive type theories such as Martin-Löf's Type Theory (MLTT) [Martin-Löf, 1982; Martin-Löf, 1984] and the Calculus of Constructions [Coquand and Huet, 1985; Coquand, 1990]. This is largely for two reasons. First, it is clear that we can go some way towards developing semantic theories within constructive type theories [Sundholm, 1989; Ranta, 1991; Davila-Perez, 1994; Ahn and Kolb, 1990]. This work has shown that constructive type theories' dependent types can be used to analyse some of the dynamic aspects of meaning. Second, such theories can be viewed as providing the means to give formal specifications of programs, and mechanisms to derive programs from proofs. Therefore, as such theories can be used both to provide a semantic theory for natural language, and a notion of computation, they have a *prima facea* case for being consider as a framework for computational semantics.

2.3.2 Some Possible Drawbacks

There are, however, some reasons why one might be cautious of promoting constructive type theories, *per se* as a framework for computational semantics, if such a framework is intended to provide a *logical space* in which to explore and compare different semantic theories.
Some constructive frameworks come with philosophical baggage that may be at odds with the semantic theories. For example, MLTT supports the view that propositions really are types. A true proposition is then a type with an element (or *witness*). That element can be considered to be a proof of that proposition. Two consequences follow: (i) false propositions are equated; (ii) sets (or classes) and propositions become indistinguishable in kind. Both of these points seem undesirable for the purposes of natural language semantics. Of course, it should be noted that not all type theories adopt such as extreme position on these issues as MLTT.
In general, constructive frameworks seem to be pitched at the wrong level of abstraction for everyday use in building and working with natural language semantic theories. When using the derivation rules, not only must you manipulate the type which expresses the proposition, but you must also be able to provide a suitable element that is a member of that type. This usually means manipulating program-like objects written in something like λ-calculus, in addition to

the work of devising definitions and axioms required in a classical framework. It seems unlikely that a classical natural language semanticist would be prepared to work at this level without significant motivation. It would perhaps be better to be able to hide away some or all of these details whilst developing a semantic theory, and to recover them when viewing the resultant theory as a program specification. Existing constructive type theories do not, as far as we are aware, allow for this possibility.
Finally, the formal aspects of theories of the power of MLTT can be captured within existing formalisms. For example, MLTT can be encoded in Property Theory [Smith, 1984; Turner, 1990; Turner, 1992] which offers a classical meta-theory, without the constraints imposed by the constructivist paradigm. Because of this, it is hard to see how constructive type theories such as MLTT can provide a *uniquely* appropriate framework for computational semantics.

2.4 Conclusion

As regards algebraic specification languages our treatment of predication has shown that, although it ultimate may be a very productive and revealing way to go, it would be fiendishly difficult to get exactly right and general even for the kind of predication that has been presented here, let alone a general treatment which covers a reasonable amount of core semantic phenomena.
It would be impossible to obtain a useable result within the life of the project and it is quite likely that any results that were obtained would give the impression of being an odd and complicated way of doing things that are otherwise quite familiar to us. While in the long term, doing this properly might set computational semantics on a very firm foundation, in the short term we would be better advised to concentrate on more useable and intuitive tools in order to achieve tangible results within the life of the project.
As regards constructive type theory our assessment should not be misinterpreted as stating that research in the constructive semantics of natural language is not a worthy occupation. Rather that, on the whole, additional motivation would be needed for classical natural language semanticists to adopt a constructive framework.

Chapter 3

Semantic Metatheory

3.1 Situation Semantics as Semantic Metatheory

3.1.1 Introduction

We will examine an approach towards framework building which has its roots in some of the work on comparison and integration of different semantic approaches conducted in the DYANA-2 project.[1] The idea there was to use the tools developed in situation theory to cast different approaches in the same general theory so that they can be explicitly compared and so that they can enrich each other.
We will consider the kind of computational semantic formalism which is suggested by this DYANA work and the situation theory described in Deliverable D8. There are three important features which are introduced there which make a semantic formalism based on situation theory distinct from other semantic formalisms. These are:

- Austinian propositions
- simultaneous abstraction
- restrictions

We will first give an overview of these three features and indicate how they are used in the situation theoretic versions of Montague semantics and DRT which were created in the DYANA project, as well as in the situation semantics presented in the first phase of the FraCaS project. We will then look in some

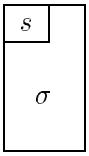
[1] This section contains some revised material from a document called 'Towards a general semantic framework' presented as part of deliverable R2.1.A in the ESPRIT BR Project 6852, DYANA-2. An earlier version of that paper was presented at the Dagstuhl Seminar "Semantic Formalisms in Natural Lanugage Processing", 22nd-26th February, 1993.

more detail at the treatments of Montague semantics and DRT proposed in DYANA.

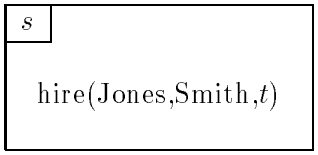
3.1.2 Overview

Let us first review these three features mentioned in the previous section and consider them from the perspective of developing a semantic representation language.

3.1.2.1 Austinian propositions

We will allow ourselves proposition terms of the form $(s : \sigma)$ (represented graphically as 

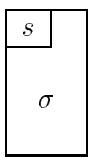
) where s is a situation and σ is an infon. An example is given in (1).

(1) 

This represents the proposition that Jones hired Smith at time t in situation s. The infon

hire(Jones,Smith,t)

is a persistent type of situations, that is, it is a type that will hold of any situation of which s is a part. Thus the proposition is a predication. It represents a type predicated of a situation.
In addition to terms such as this we allow formulas in our language which assert that s is of a type σ. We write this as $s : \sigma$ (or, following the standard tradition of situation theory $s \models \sigma$). The relationship between the proposition terms and these formulas is expressed by (2).

(2) $true($ 

$) \leftrightarrow s \models \sigma$

Austinian propositions in Montague semantics Austinian propositions are used in the recreation of Montague's semantics to solve problems of granularity. In Montague's original traditional possible worlds treatment, propositions are treated as functions from world-time pairs to truth-values, or equivalently as sets of world-time pairs. This meant that any two sentences which were logically equivalent would correspond to the same Montague proposition. Austinian propositions can be equivalent without being identical and thus do not suffer from the problem of the traditional possible worlds approach. We can capture Montague's idea of making propositions dependent on worlds and time by abstracting over the s and t in a proposition like (1). Thus the reconstruction of a Montague proposition in the situation theoretic treatment is an object which takes a situation/world and a time and returns an Austinian proposition. This is to be compared with Montague's original proposition which took a world and a time and returned a truth-value. The details of the situation-theoretic treatment are discussed in section 3.1.3.

Austinian propositions in DRT Austinian propositions are used in the DRT reconstruction to represent conditions in DRSs that represent constraints on events or states. Consider, for example, the discourse in (3) discussed by [Kamp, 1990]

(3) Last month a whale was beached near San Diego. Three days later it was dead.

His DRS for this makes crucial use of a discourse referent for the event of the whale being beached and another discourse referent for the state of the whale being dead and there is a condition relating the temporal occurrence of the two. In standard DRS notation the relevant conditions are as in (4).

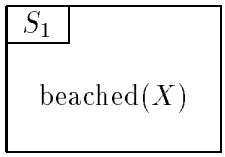
(4) a. e : $\boxed{\text{beached}(x)}$

b. s : $\boxed{\text{dead}(y)}$

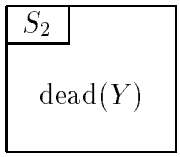
These conditions are represented in the situation theoretic reconstruction of DRT as parametric Austinian propositions (the parameters corresponding to discourse referents). The Austinian propositions are represented in (5).[2]

[2]Here we do not distinguish between those situations which are event and those which are states, though it is possible to do this.

(5) a.

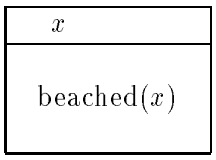


b.

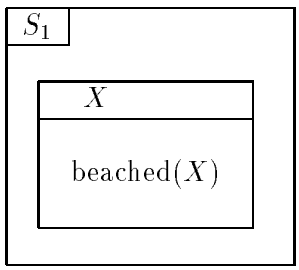


Treating these conditions as Austinian propositions makes a certain prediction which seems to be borne out at least by the treatment presented in [Kamp and Reyle, 1993]. Austinian propositions are constructed from a situation and a situation-type, in this particular case a persistent situation type or infon, this places certain restrictions on the kind of DRS that can be to the right-hand-side of the colon in the DRS condition. Only DRSs without domains correspond to infons. Thus the condition in (6a), while potentially allowable according to DRS syntax does not make any sense when converted into the corresponding situation theoretic representation (6b).

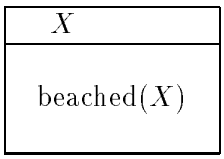
(6) a. e :



b.



(6b) is not well-formed because



is a property of individuals rather than a type of situations.

Another important difference concerns the treatment of negative events. Whereas the original DRT is based on a Davidsonian event semantics, a situation theoretic treatment allows for a treatment of negative events in terms of negative infons. Thus if we are representing the sentence (7a), example 128 of

D5, discussed on p. 116 of deliverable D10, it is natural to use an Austinian proposition like that in (7b).

(7) a. Smith did not travel by air

b. $\boxed{\begin{array}{|l} S \\ \hline \neg\text{travel\_by\_air}(X) \end{array}}$

(7b) gives us an event S which can be referred to subsequently in the discourse, for example by a follow-up sentence like (8).

(8) It was a terrible journey.

In principle it would be possible to formulate DRS syntax so as to allow us to construct a corresponding negative DRS condition, as in (9).

(9) $e : \neg \boxed{\text{travel\_by\_air}(x)}$

However, it is difficult to see how to construe this in terms of the Davidsonian semantics which has been proposed for traditional DRT. Consider first the positive case. The DRS condition (10a) corresponds to the Davidsonian semantics (10b).

(10) a. $e : \boxed{\text{travel\_by\_air}(x)}$

b. $\text{travel\_by\_air}(e, x)$

(10b) is to be read intuitively as "e is an event of travelling by air by x". When we consider the negative condition (repeated in (11a)), it appears that the only possibility for representing the negation in Davidsonian terms is (11b).

(11) a. $e : \neg \boxed{\text{travel\_by\_air}(x)}$

b. $\neg\text{travel\_by\_air}(e, x)$

(11b) says "e is not an event of travelling by air by x". But what the DRS suggests and the particular Austinian proposition which corresponds to it gives us is "e is an event of not travelling by air by x". This inner scope of the negation is made possible by the fact that situation theory allows negative infons. It is impossible to obtain on a Davidsonian approach because there is nothing corresponding to an infon to be negated.
The issue, then, is whether we need to have the internal negation or not. The answer is by no means clear. On one view at least, we seem to need a theory of events which allows us to conclude that Smith's not travelling by air refers to an event which has a positive type which precludes Smith travelling by air, such as being an event of Smith's travelling by train or staying home to get some work done. To have just the wide scope negation does not allow us to prescribe this inference. An event which is not an event of Smith travelling by air could be one of him eating lunch, or indeed of somebody else eating lunch, neither of which preclude it being an event of him travelling by air, except, possibly, in special circumstances.

Austinian propositions in situation semantics We isolate two uses of Austinian propositions here: resource situations and event structure. The idea behind resource situations is that we can use Austinian propositions to provide a situation associated with a particular use of a noun-phrase in order to restrict the range of the quantifier expressed by the noun-phrase (or determine a referent for the noun-phrase in the case of a definite). Arguments are provided in [Cooper, 1996]) that resource situations should be distinct from the situations described by sentences.
Austinian propositions are the vehicle in situation semantics for discussing event structure since they involve predication of situations (including events). In the early situation semantics literature the discussion of event structure centres around naked infinitive perception complements and we have been working on extending this to the analysis of event structure that is required for the analysis of aspect and event anaphora.

3.1.2.2 Simultaneous abstraction

The DYANA work we are reporting on here uses the notion of simultaneous abstraction as developed by [Aczel and Lunnon, 1991] and presents it in the context of situation theory. However, there are now other versions of simultaneous abstraction, notably that developed by Peter Ruhrberg [Ruhrberg, 1994]. While the Aczel-Lunnon version of simultaneous abstraction was developed originally to support situation theory it is independent of it and the central ideas of simultaneous abstraction can be seen as independent of Aczel and Lunnon's

particular development.
On the Aczel-Lunnon approach, abstracts are regarded as a particular kind of object in a structured universe. This contrasts with the standard[3] view of abstracts received from Montague's semantics as functions or rather λ-expressions which are interpreted as functions. While this is important for the situation theoretic approach there are two other features of this kind of abstraction which are perhaps of more general importance for a computational semantic formalism:

simultaneous abstraction Any number of parameters in a parametric object may be abstracted over simultaneously. While in standard λ-notations one may have expressions such as

$$\lambda x, y, z[\phi(x, y, z)]$$

this is to be construed as an abbreviation for

$$\lambda x[\lambda y[\lambda z[\phi(x, y, z)]]]$$

In Aczel-Lunnon abstraction, however, it is the set which is abstracted over. Thus arguments to the abstract can be supplied simultaneously and there is no required order.

indexing This feature is closely related to the previous one. Since abstraction over parameters results in an object in which those parameters do not occur[4], we have to have some way of determining how arguments are to be assigned to the abstract in the case where more than one parameter has been abstracted over. Aczel and Lunnon achieve this by defining the abstraction operation in terms of indexed sets of parameters, i.e. one-one mappings from some domain ("the indices") to the parameters being abstracted over. An important aspect of this for us is that we can use any objects in the universe as the indices.

Simultaneous abstraction in Montague semantics Simultaneous abstraction is not exploited in the situation theoretic reconstruction of Montague semantics since Montague employs the unary abstraction of the standard λ-calculus. It is used in the particular treatment presented for the abstraction corresponding to Montague's abstraction over world-time pairs, although this is not a crucial use of simultaneous abstraction.

[3] at least for formal semanticists

[4] This is important in order to achieve α-equivalence, i.e. $\lambda x[\phi(x)] = \lambda y[\phi(y)]$

Simultaneous abstraction in DRT The proposal for reconstructing DRSs in terms of situation theory is that they are abstracts which are either relations or types. (See below for more detailed discussion.) The idea is that a DRS such as (12a) can be represented as an abstract such as (12b).

(12) a.

x,y
man(x) donkey(y) own(x, y)

b.

$i \rightarrow X, j \rightarrow Y$
man(X) donkey(Y) own(X, Y)

(12b) in linear notation is given in (13).

(13) $\lambda[\, i \rightarrow X, j \rightarrow Y](\text{man}(X) \wedge \text{donkey}(Y) \wedge \text{own}(X, Y))$

The parameters abstracted over correspond to the universe of the DRS. The fact that we use simultaneous abstraction means that we can have a single abstraction corresponding to the DRS and that this can be embedded in further levels of abstraction for compositional purposes in the style of Montague. Thus we can have the effect of λ-DRT without having to introduce special DRS objects into our semantic universe. The fact that we are using arbitrary indices enables us to compose DRSs and use coindexing to achieve discourse anaphora. By using different indices for the "DRS"-abstractions and the "compositional"-abstractions we are able to treat both DRSs and Montague style compositionality within a unified abstraction framework.

Simultaneous abstraction in situation semantics We use simultaneous abstraction in situation semantics to treat the variable adicity of meanings. The basic format for a meaning is given in (14).

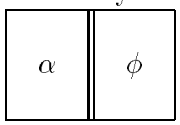
(14)

context roles
content

This follows the Montague-Kaplan view that a meaning is applied to a context (modelled as an assignment to the roles of an abstract) to obtain a content (corresponding to what Montague would call an intension). The use of simultaneous abstraction allows meanings for different phrases to have different numbers of context roles and indeed for the same phrase to be ambiguous with respect to the number of roles which depend on context, without this having as a consequence that the different meanings are different types (or sorts) or objects or that any ordering is implied among the roles. The context roles are related to the notion of universe in a DRS and in a similar way to the treatment of DRT we are able to gather all the context roles together "in one bag" while still using different levels of abstraction within the same object for the "compositional" abstracts in the style of Montague. This view of the techniques makes the situation semantics and DRT approaches look very similar.

3.1.2.3 Restrictions

Restrictions provide a way of backgrounding information and of expressing presuppositions. In terms of a language of semantic representation the idea behind restriction is as follows:

- if α is any kind of term and ϕ is a proposition term then $\alpha \upharpoonright \phi$ (graphically, ) is a term
- $\alpha \upharpoonright \phi = \alpha$ if ϕ is true
- $\alpha \upharpoonright \phi$ is undefined if ϕ is false

In the case where ϕ contains parameters it will be neither true nor false, but given an assignment to its parameters it can be anchored to become true or false. Thus in the parametric case the restriction can be seen as representing a restriction on parameters returning α under an assignment just in case ϕ under the assignment is a true proposition.
Restrictions obey various identity conditions mainly having to do with distribution. Some examples are

$$(\alpha \upharpoonright \phi) \upharpoonright \psi = \alpha \upharpoonright (\phi \wedge \psi)$$

$$(\alpha \upharpoonright \phi) \wedge \beta = (\alpha \wedge \beta) \upharpoonright \phi$$

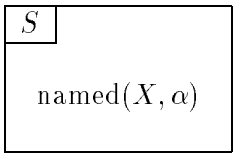
And similarly for the other connectives

The only place where restrictions do not distribute is when they contain parameters within the scope of a λ. In this case the restrictions cannot distribute beyond the scope of the λ and they represent appropriateness conditions on assignments to the abstract represented.

Restrictions in Montague semantics The situation theoretic reconstruction of Montague semantics that was presented in the DYANA project was exactly Montague's treatment of the PTQ fragment. Thus there was no treatment of presupposition or sortal restrictions and restrictions were thus not used. It would be an interesting exercise to add restrictions to some basic extension of the PTQ fragment and see what treatment of presuppositions emerges.

Restrictions in DRT In the treatment of situation theoretic DRT presented in DYANA proper names α are treated as introducing a restriction of the form (15).

(15)



Because of the distribution laws these restrictions percolate up to the level where the X is abstracted over, i.e. the level of the topmost DRS.[5] Thus by making the naming condition a restriction it can be introduced arbitrarily low down in the compositional semantics and "rises" to the appropriate place determined by the scope of the parameter to which it is attached.
There are other potential uses for restrictions in a situation theoretic version of DRT but they are not exploited in the fragment presented in this work.

Restrictions in situation semantics Restrictions in situation semantics are used to express presuppositions and appropriateness conditions. In the fragment defined in deliverable D9, they are mainly used for appropriateness conditions on parameters associated with context roles, such as the naming restriction associated with proper names as discussed for DRT above, and various constraints on the discourse situation. Restrictions are a useful tool for expressing constraints which originate on particular constituents and become constraints associated with a whole utterance because of their distributive properties required by the situation theory.

3.1.2.4 Summary

Here is a summary table indicating the uses which the three features we have discussed have been put to in the DYANA work we are discussing.

[5] This is not strictly true, although it ought to be. Given the way the reduction rules are implemented a restriction will not rise above any intermediate abstraction even if it does not bind the parameters in the restriction. This needs some further consideration.

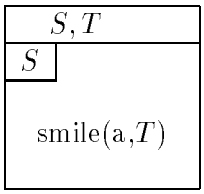
	Montague	*DRT*	*SS*
Austinian propositions	granularity problems	conditions of the form $e : K$	resource situations, event structure
simultaneous abstraction	not crucially used, though employed for abstraction over world(situation)-time pairs	DRSs treated as abstracts	variable adicity of meanings and context dependence
restrictions	not used	restrictions associated with proper names percolate up to first abstraction	presupposition, appropriateness conditions

In the next two sections we will give a slightly more detailed review of the treatments of Montague semantics and DRT presented in the DYANA work.

3.1.3 Montague's semantics

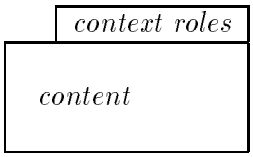
The basic idea behind the reconstruction of Montague's semantics is that we treat Montague's propositions as situation theoretic types, abstracts which when applied to an appropriate assignment return a situation theoretic proposition. Here we shall think of them as simultaneous abstracts over situations or possible worlds and times as represented in (16).

(16)



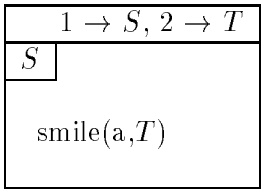
Let us compare this object with the kind of meanings that we introduced in D9 in the treatment of situation theoretic grammar (STG). There the idea was that a MEANING should be an abstract which requires an assignment (modelling a context). When a meaning of an utterance is applied to a context assignment it yields a CONTENT of that utterance. Graphically, this can be represented as in (17).

(17)



In STG the context roles associated with different utterances can vary and it was therefore useful to have role indices, usually uniquely identified for the particular utterance. In Montague semantics the context roles are always the same. In PTQ there are just two, one for a possible world and one for a time. Because of this there is no need to create an elaborate system of role indices. We shall use the natural numbers. In (17) we are making use of a notational convention introduced in [Barwise and Cooper, 1991] which suppresses the role indices in an abstract when they are the natural numbers beginning with 1. Thus (17) is really an abbreviation for (18).

(18)



Note that in this abbreviatory notation the order of the S and T in the tab in (16) is significant. We introduce a similar abbreviatory notation for assignments. Thus (19a) is an abbreviation for (19b).

(19) a. $[S, T]$

b. $\begin{bmatrix} 1 \rightarrow S \\ 2 \rightarrow T \end{bmatrix}$

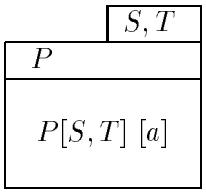
We shall assume that the structure of time is treated here in the same way as Montague treats it in PTQ. It becomes an interesting issue whether the first role, for which we have used the parameter S, should be for possible worlds, in Montague's sense, or situations (possible or actual? partial or total?). We will return to this issue after we have discussed the choice of situation theoretic propositions (in contrast to Montague's truth-values) as the result of applying the meaning of a sentence to a context. But first let us establish concretely that the use of Aczel-Lunnon abstraction and structured objects does not preclude the use of abstraction for Montague's compositional techniques.

3.1.3.1 Compositionality

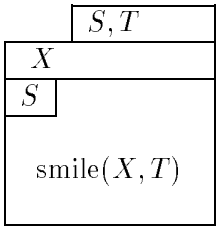
In order to indicate how the Montague style combinatorial machinery would work we give a very small sample grammar in (20).

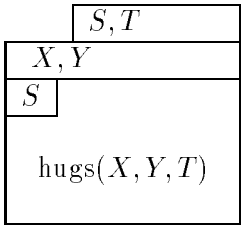
(20) NP – {Anna, Claire, Maria}
V_i – {smiles}
V_t – {hugs}

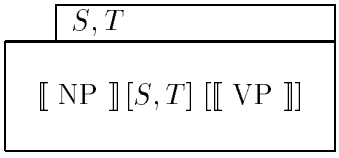
S → NP VP
VP → V_i
VP → V_t NP

$[\![$ Anna $]\!] =$ 

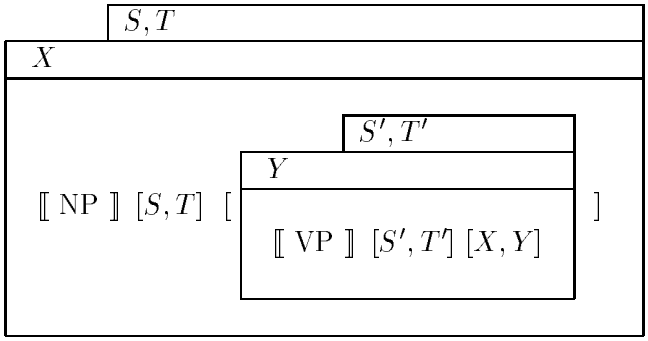
And similarly for the other proper names.

$[\![$ smiles $]\!] =$ 

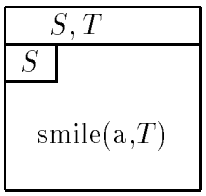
$[\![$ hugs $]\!] =$ 

$[\![$ [NP VP]$_S$ $]\!] =$ 

$[\![$ [V NP]$_{VP}$ $]\!] =$



Using this grammar and several applications of β-conversion we can show

$[\![$ Anna smiles $]\!] =$ 

as desired.

Note that the semantic composition here is going on at the level of what Montague would call "intension" (and what we are calling meaning[6]). We let $[\![\alpha]\!]$ be the meaning ("intension") of α rather than its content ("extension"). In Montague's PTQ style of presentation the latter is the case and the normal case of application involves the '^'-operator in intensional logic, e.g. $\alpha(\beta)$. If α and β represent meanings ("intensions") as they do in our presentation then the normal case of application would be represented in terms of Montague's intensional logic as $[\check{\alpha}](\beta)$ where $\check{\alpha}$ represents the "extension" of α, i.e. the result of applying it to the world and time of evaluation. We represent this application explicitly as $\alpha[S,T]$. The "intensional" mode of presentation is more convenient for us because we are representing the abstractions over worlds and times explicitly rather than in the metatheory of the interpretation of a language as Montague does in PTQ. The '^'-operator relies on this metatheoretical definition. But the effect for compositional interpretation is precisely equivalent.

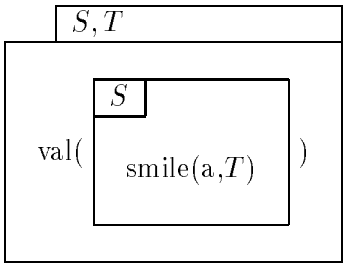
3.1.3.2 Propositions or truth values?

Why did we choose to make the result of applying a sentence meaning to a context a proposition rather than a truth value as in Montague's original? Part of the reason is that there is no such thing as a truth-value conceived of as an object in the universe in situation theory. There are objects which are either true or false. These are the propositions. Situation theory, as designed for use in situation semantics, does not appear to need truth-values as objects in addition. As in property theory, truth and falsity are properties of propositions, not objects in their own right.

Perhaps this means that in order to accurately model Montague's semantics in situation theory we should enrich it with truth-values and an operation **val** which maps a proposition to **True** if it is true and **False** otherwise (i.e. if it is false – recall that propositions in situation theory are classically bivalent). Then we could say that the meaning of *Anna smiles* is (21).

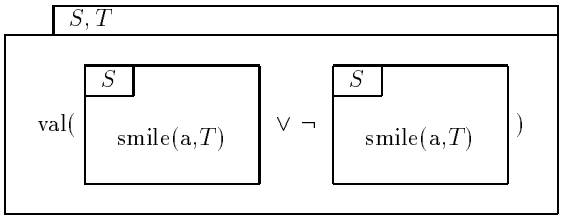
[6] We will discuss the relationship between our notion of meaning and Montague's below.

(21)

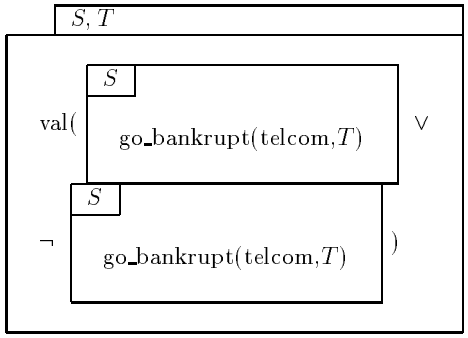


Since **val** is a kind of operation that you do not normally find in situation theory in that it does not preserve any of the structure of what it operates on and yet it produces different results depending on what value for S and T you choose, it is not very clear what kind of abstract in an Aczel-Lunnon universe (21) could denote. Let us stipulate for the sake of argument that this notation represents a function (in the set-theoretic sense) from worlds/situations and times to truth values. This move would indeed bring us closer to Montague's original in that sentence meanings would now be functions from worlds and times to truth-values. However, it would bring back the problem that Montague's semantics has of not quite finely individuating propositions enough. Expressed in terms of situation theory, the problem would be this: if you have two propositions parametric on situations/worlds and times which for each anchoring of situations/worlds and times produce a proposition with the same value, then you would not be able to distinguish the types obtained as the meanings of the sentences. Let us consider a simple example of tautologies to make this concrete. In situation theory, just as in classical logic, the disjunction of a proposition with its negation is always true. This means that (22a) and (22b) will be identical objects.

(22) a.



b.



If on the other hand we take the option without **val** and return to types we see that the structure axioms of situation theory guarantee that the types and the propositions resulting from applications to contexts are distinct (although they are all true). The temptation to avoid the unwelcome complication to the situation theory and solve this problem is just too strong, so we have been slightly unfaithful in recreating Montague's semantics.
The solution to the "logical equivalence" problem that we have just discussed is, of course, essentially similar to that presented by property theory and other approaches where propositions are taken to be objects in the universe which can be distinct even if equivalent (e.g. [Thomason, 1980], [Church, 1951]). This approach differs in that the propositions are conceived of as structured objects. It is this fact that makes it straightforward for us to treat sentence meanings as abstracts with roles for situation and times. Because the propositions are structured objects we can introduce parameters available for abstraction corresponding to any constituent[7] of the structured object. We believe that this is not so straightforward using property theory where propositions are not structured.

3.1.3.3 Situations or worlds?

There are a number of choices we can make as to what we make appropriate to the situation role in these meanings:

1. possible worlds

[7] Taken hereditarily, e.g. constituents of constituents ...

2. possible situations

3. actual situations

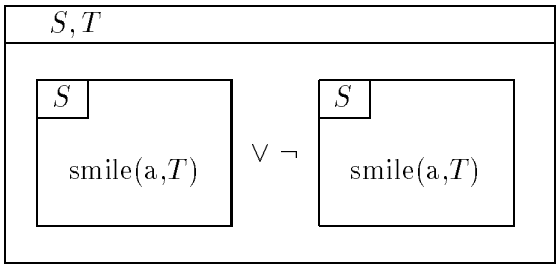
We will claim that

- it does not matter which of these choices you make if you use types which yield situation theoretic propositions when applied to contexts
- possible situations will give us the finest grain if we think of the meanings as being functions yielding truth-values. However, this will fail to make distinctions which are made using propositions rather than truth-values

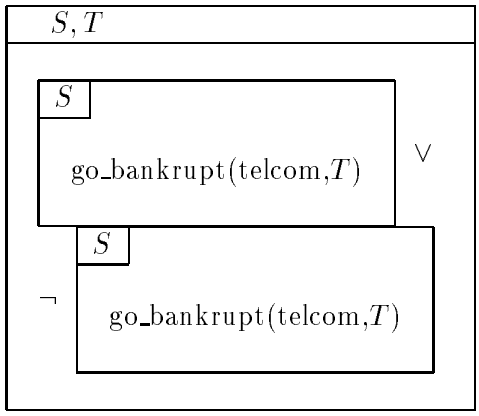
We will discuss the three options in turn.

Possible worlds Imagine that we want to import a set of classical possible worlds as objects into our situation-theoretic universe. Possible worlds, like situations, have things true in them, i.e. that is they belong to the kinds of types that are represented by infons in situation theory. So we would want worlds to "support" infons just as situations do. The difference would be that for each basic infon σ a possible world should support either that infon or its dual, but not both. But then we see that worlds just are maximal consistent situations, what are called complete and coherent situations in the jargon of situation theory. If we assume that our situation theory allows all possible non-actual situations (nothing in situation theory prevents this, although it does not require it either), then there is no need to import possible worlds. They are already present.
Now let us suppose that the situation-role in our sentence meanings is restricted to possible worlds and let us consider the case of the tautologies we discussed before in their "type form" as in (23).

(23) a.



b.



Clearly these types will have the same extension, that is, they will produce true propositions for exactly the same set of worlds and times (the universal set, in fact). Nevertheless, they are different types which yield different propositions when applied to these worlds and times. The use of possible worlds has no effect on the identity of propositions as long as we do not use the possible worlds to identify the propositions. Of course, if we take the function to truth values option, the problem reappears – the two represent identical functions.

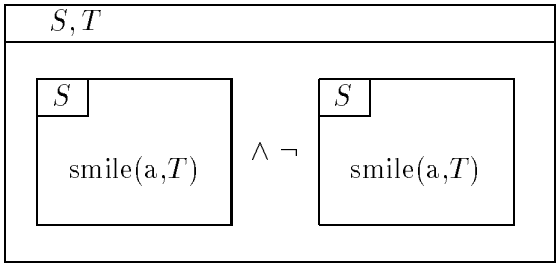
Possible situations Suppose now that we extend the range of the situation role to include not only the complete and coherent situations (the worlds) but also the partial ones as well. On the type view our tautologies are still distinguished. On the function view we get a distinction which we didn't get before. Since we have now introduced partiality, we have some situations which do not support either the relevant infons or their duals and some which support only one of them. Hence the introduction of partial situations enables us to distinguish these two tautologies without requiring the use of situation theoretic (or property theoretic) propositions. This is essentially the proposal that is made by [Muskens, 1989].

Actual situations Suppose now that we restrict the range of situations down again, but with a different cut this time. We allow only situations which are actual, both complete and partial ones and throw away all of the ones that are

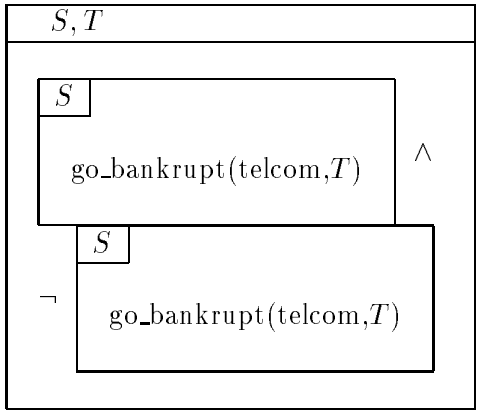
possible but not actual. The extensions of the types will, of course, be severely cut down by this restriction, but no distinctions between the types will be lost since the types are not identified in terms of their extensions. Funnily enough, on the function view, our two tautologies will be distinguished as well. But the fact that the meaning of all false sentences will be identical (the constant function from situations and times to False) makes this an unrealistic choice.
It seems, then, that we have two main choices: either the type view or the function view together with possible situations. Is there some way to distinguish between them in terms of their fine-grainedness of meaning? We will now try to argue that the type view is more fine-grained and furthermore that the additional fine-grain it provides is desirable. We will consider two cases which we consider to be important. The first involves contradictions and the other involves local constraints on situations.

Contradictions We have looked so far at an example of distinguishing two tautologies. Now let us consider distinguishing two contradictions, say (24).

(24) a.



b.



On the type view (24a) and (24b) are distinct objects, even though they have identical extensions. The story is no different for contradictions than it is for tautologies. On the function view, however, the fact that we are dealing with contradictions makes a difference. Since situations are consistent, none of them

will support any contradiction. Hence on the function view, even using possible situations will not enable us to distinguish two contradictions. This can be solved by giving up the requirement that possible situations are all consistent, by allowing "partial impossible worlds". This is the route that Muskens would follow. Other people may feel that it is a philosophically too high price to pay.

Local constraints We may want to place constraints on our situations which say that if they are of one type then they are also of another. A good candidate for this is a by now rather hackneyed example from early situation semantics (Barwise and Perry, 1983). It concerns kissing and touching. One may well want to say that any situation where a kisses b is also a situation where a touches b. This is stronger than saying that a's touching b follows from a's kissing b. To claim that the facts hold in the same situation means that you cannot perceive a kissing b without also perceiving a touching b. The extent to which such tight contraints are desirable is a matter of debate but it seems unlikely that we would want our semantic machinery to rule out the possibility. Even if we agree that this constraint on kissing and touching holds we would not want that to have the consequence that the meanings of *a kissed b* and *a kissed and touched b* could not be distinguished. On the type view they would be distinct meanings, although with the same extension. On the function view with possible situations, they would fall together. In order to fix things on the function view we would again have to import impossible situations that violate the constraint.
In defining the fragment here, we have taken what appears to be the more conservative choice which allows us to solve Montague's problem without introducing either possible or impossible situations. We will return to the issue of whether possible situations are necessary when we discuss Montague's treatment of modality in the next section. It is important to notice, though, that on this view the solution to the logical equivalence problem is driven by the use of propositions and types. Once we have made this decision it is independent of whether we use situations or (im)possible worlds.

3.1.3.4 Modality

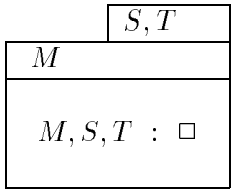
In this section we will discuss two ways of handling Montague's treatment of necessity in PTQ within our situation theoretic version of Montague's semantics. One way involves introducing possible worlds into the situation theory and treating necessity as universal quantification over these worlds. We shall call this the "Lewis" view of modality, because of Lewis's view of possible worlds as real alternative objects. The other way involves treating modality in terms of "ways that the world might be", i.e. types of worlds. We shall call this the

"Stalnaker/Cresswell" view of modality, because of their view of possible worlds as ways that the world might be. We will come to the conclusion that a basic treatment of modality does not require the introduction of possible worlds into situation theory (and that therefore the Stalnaker/Cresswell view is the more conservative one) and leave open the question as to whether a more modern treatment of modality does require possible worlds.
Recall that Montague translates the English word *necessarily* as $\lambda p[\Box p]$ where p is a variable over Montague's propositions and $\Box$ is an operator in intensional logic. This means that in our terms *necessarily* takes as arguments what we have been calling sentence meanings or Montague propositions, i.e. situation-time types. We will treat the symbol $\Box$ as denoting a ternary type taking Montague propositions, situations and times as its argument. Thus we will write

(25) $m, s, t \ : \ \Box$

to mean that m, s, t are of type $\Box$. On the Lewis view we will say that this holds true if for every world s' and time t' accessible from s and t respectively, $s', t' \ : \ m$, i.e. s', t' are of the type m which is to say the same as that $m[s', t']$ is a true proposition. On the Stalnaker/Cresswell view, we will say that (25) holds true essentially just in case for every time t' $\lambda[S](m[S, t'])$ is informationally subsumed by any coherent and complete infon, i.e. there is no way we could have total information relating to t' and not have m turn out to be true for t' and a complete and coherent situation. The two definitions we will give will say bascially the same thing, except that the first will commit us to the existence of alternative possible worlds whereas the second will just talk of types of those worlds.
In order to treat *necessarily* as something which "maps from a Montague proposition to a Montague proposition" we will give it the meaning in (26).

(26)



For the Lewis view we assume that there are non-actual possible situations and that there is an accessibility relation **acc** in $(S \times T)^2$. (In accordance with Montgue's assumption of S5 modal logic, we will assume that this is the universal relation, i.e. any situation-time pair is accessible from any other situation-time pair.) We distinguish some of the situations as being complete and coherent (i.e., possible worlds). We define these notions in (27).

(27) a. A situation s is COMPLETE, compl(s), iff for all basic infons σ $s \models \sigma$ or $s \models \overline{\sigma}$

b. A situation s is COHERENT, coh(s), iff there is no basic infon σ such that $s \models \sigma$ and $s \models \overline{\sigma}$

Now we can say what it means for a Montague proposition m to be necessary with respect to a situation s and time t.

(28) $m, s, t \ : \ \Box$ iff $\forall s', t'(\text{acc}(< s, t >, < s', t' >) \wedge \text{coh}(s') \wedge \text{compl}(s')) \rightarrow s', t' : m)$.

This says that any s', t' accessible from s, t will be of the type m if $m, s, t \ : \ \Box$ is the case.
For the Stalnaker/Cresswell view we will not introduce possible non-actual situations but rather treat the modality in terms of types of situations. The types that we will use are the persistent types which we call infons. To facilitate our definitions we will assume the identity in (29).

(29) $\lambda[S](S : \sigma) = \sigma$

In order to have situation types which will correspond to possible worlds we need to have notions of completeness and coherence for situation types and in order to have these notions we need a subsumption relation holding between two types. We shall talk of one type being informationally subsumed by another, σ is subsumed by τ, $\sigma \leq \tau$ meaning intuitively that τ contains the information contained in σ. We define this relation in the standard way for infons.[8]

(30) **Subsumption for infons**

a. $\sigma \leq \sigma$

b. $\sigma \leq \sigma \wedge \tau$

c. $\sigma \vee \tau \leq \sigma$

Now it is straightforward to define completeness and coherence for types in a similar manner to our definition for possible situations above.

(31) a. An infon τ is COMPLETE, compl(τ), iff for all basic infons σ $\sigma \leq \tau$ or $\overline{\sigma} \leq \tau$

b. An infon τ is COHERENT, coh(τ), iff there is no basic infon σ such that $\sigma \leq \tau$ and $\overline{\sigma} \leq \tau$

[8] To make this subsumption relation be intuitively correct we need to make standard assumptions about commutativity and associativity of $\wedge$ and $\vee$.

The basic idea behind the Stalnaker/Cresswell view is that a Montague proposition is necessarily true if for any time the infon resulting from evaluating the Montague proposition with respect to that time is subsumed by any complete and coherent infon. This basic intuition does not immediately leave room for accessibility, and indeed for the S5 version of modal logic assumed by Montague, we have no need of an explicit accessibility relation. We can thus leave it out and give the definition in (32).

(32) $m, s, t : \Box$ iff $\forall \sigma, t'(\mathrm{coh}(\sigma) \wedge \mathrm{compl}(\sigma)) \rightarrow \lambda[S](m[S, t']) \leq \sigma$.

One way to build accessibility back in is to have a relation between situation-time pairs and infons and give the definition in (33).

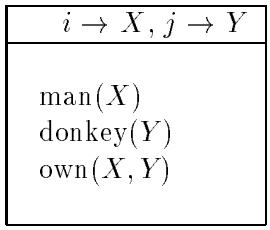
(33) $m, s, t : \Box$ iff $\forall \sigma, t'(\mathrm{acc}(< s, t >, \sigma) \wedge \mathrm{coh}(\sigma) \wedge \mathrm{compl}(\sigma)) \rightarrow$
$\lambda[S](m[S, t']) \leq \sigma$.

This restricts the previous definition by requiring in addition that complete and coherent type σ be accessible from s and t.
The notion of coherence that we have used here has just been in terms of not allowing both infons and their duals. In a more realistic treatment we should require that coherent situations/infons respect certain constraints, for example, corresponding to Montague's meaning postulates in PTQ.

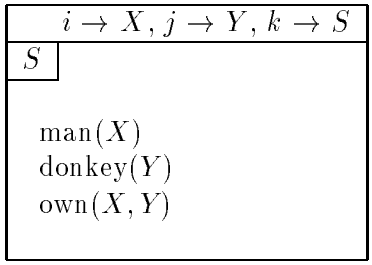
3.1.4 Discourse representation theory

In our treatment of discourse representation theory the simultaneous nature of the abstraction coupled with indexing using arbitrary indices will be crucial. The leading idea is that we model discourse representation structures as abstracts which from the situation theoretic perspective are predicates. If we are working in situation theory, modelling DRSs as predicates gives us two options. They can be either relations or types. The choice is illustrated in (34) with respect to DRS corresponding to *a man owns a donkey* (ignoring matters of tense).

(34) a. *Relation*



b. *Type*



The difference is that in the relation there is no role for a situation whereas this is the case in the type.
I think that ultimately DRSs need to be modelled as types in order to deal with the DRT analyses of tense and attitudes. It is natural within situation theory to think of DRS conditions where DRS's are used to classify events and states as corresponding to propositions concerning those events and states (i.e. situations) supporting infons. Consider, once again, the discourse in (35) discussed by [Kamp, 1990].

(35) Last month a whale was beached near San Diego. Three days later it was dead.

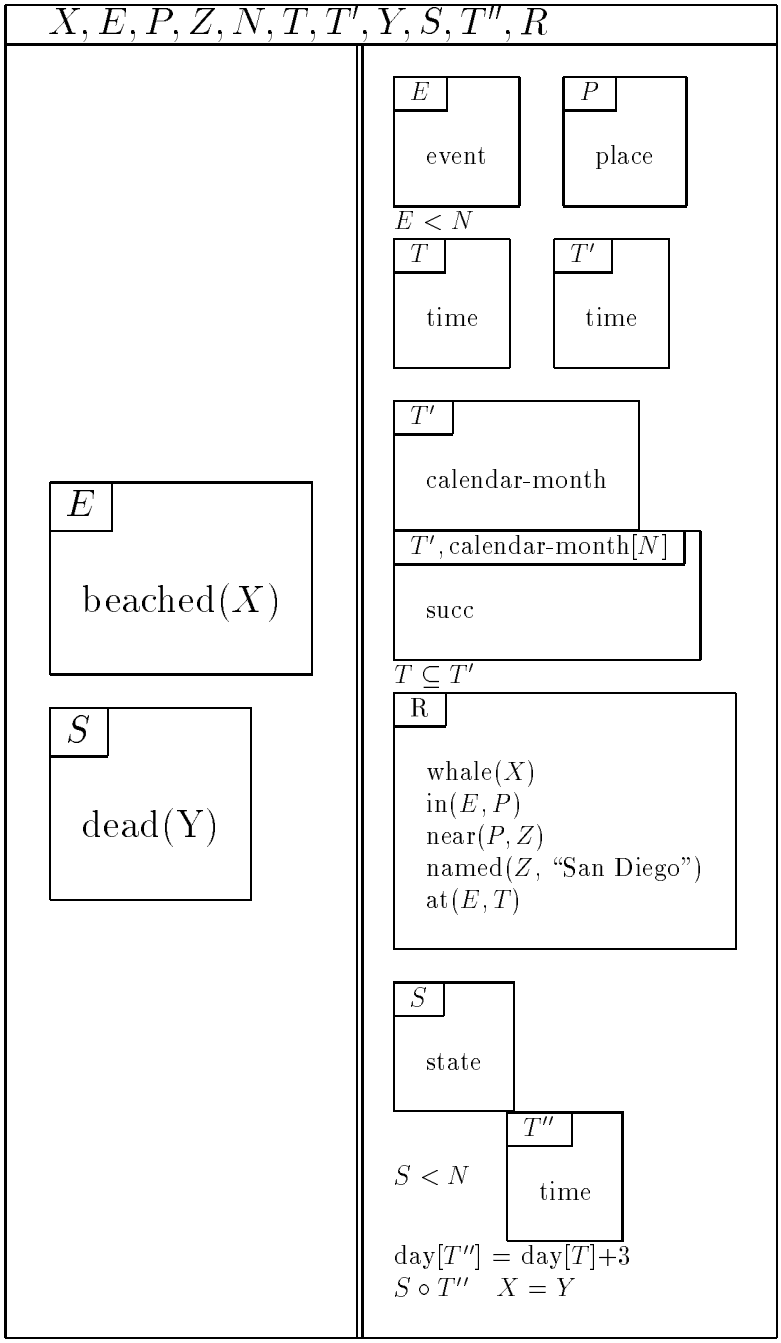
The DRS for this makes crucial use of a discourse referent for the event of the whale being beached and another discourse referent for the state of the whale being dead and there is a condition relating the temporal occurrence of the two. The relevant conditions look as in (36).

(36) a. $e \ldots$ beached(x)

b. $s \ldots$ dead(y)

Often in DRS notations ':' is used instead of '...'. In (37) we give a rough reconstruction of Kamp's DRS for the discourse as a situation theoretic type. We have made many arbitrary decisions here, for example, concerning which information is backgrounded as restrictions and the exact representation of temporal relations. Our only aim here is to illustrate the relationship between DRT's ':' or '...' and the situation theoretic notion of a situation supporting an infon.

(37)



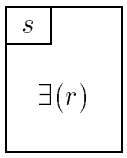
However, even though we think that in a situation theoretic approach DRSs should ultimately be treated as types, it is nevertheless very attractive to treat simple DRSs that do not involve conditions with ':' or '...' as relations, pre-

cisely because this emphasizes that it is not the situations which are central to the modelling of DRSs but rather the Aczel-Lunnon abstraction. This then offers us the possibility of building on the Aczel-Lunnon abstraction but moving in a direction other than situations to obtain a complete treatment of DRSs, if such a move should be considered desirable. For the remainder of this paper we will talk of DRSs as relations.

This means that we will take (34a) as the relation corresponding to the DRS for *a man owns a donkey*. How now do we get the effect of non-selective existential quantification that is obtained when traditional DRSs are interpreted in a model? We cannot use interpretation in a model since our DRSs are not syntactic objects. However, abstracts are the kind of thing that can be quantified over using a variant of the same technique that is used for the introduction of quantification into the λ-calculus as represented in (38).

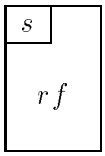
(38) $\exists(\lambda x[\phi(x)])$

We introduce a distinguished property of relations "instantiated" or "realized" represented by $\exists$. This holds of a relation just in case there is some assignment to the roles of the relation which yields something that holds true when the relation is applied to the assignment. We can make this precise in situation theoretic terms as in (39), though, of course, one could choose to do it a different way if one wished to avoid the committment to situations.

(39) 

is true implies

there is some assignment f appropriate to r such that

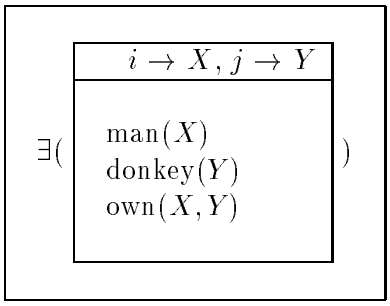


is true

If desired (39) could be strengthened to a biconditional, though we do not believe that this is necessary.

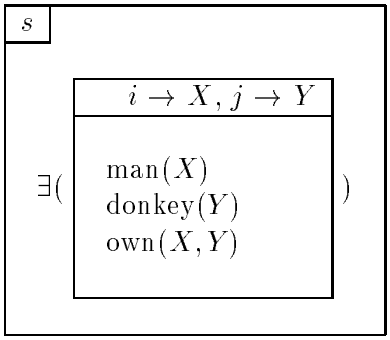
This shows us that, given the relation corresponding to the DRS for *a man owns a donkey*, we can construct an infon where this relation is existentially quantified.

(40)



Notice the important effect of unselective binding here. Since we are using simultaneous abstraction we have simultaneous quantification. This is one important ingredient which enables us to capture the classical DRT analysis of donkey anaphora.
Now that we have a quantified infon it is straightforward to use this to construct a proposition which might correspond to the interpretation of a DRS in classical DRT. If one does this in situation theoretic terms one such proposition is (41), though, of course, one might wish to use a different notion of proposition.

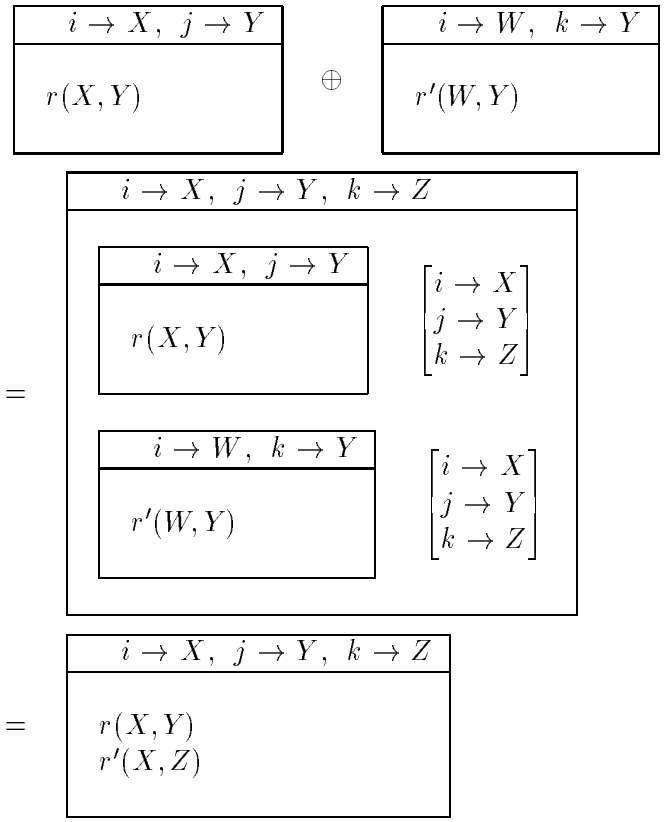
(41)



But how are you going to achieve the effect of discourse anaphora if the DRS that you construct for a single sentence is modelled as an abstract where everything is already bound? It is here that the second feature of Aczel-Lunnon abstraction that we highlighted comes into play. The use of arbitrary role indices allows us to bind parameters but at the same time uniquely identify the roles in the abstract and identify roles across different abstracts. In designing grammars the strategy that we have been using for the incrementation of discourse representation is to assign a predicate corresponding to a DRS to each new sentence of the discourse and then integrate that predicate with the one obtained for the discourse so far. Basically the integration is predicate conjunction where roles that have the same index are merged. We define an operation of predicate conjunction, $\oplus$, which will be the central tool used in the incrementation of one DRS with another DRS (corresponding to the next sentence in the discourse).

The idea is best illustrated first by an example.

(42)



In (42) we have two binary predicates which are conjoined by $\oplus$ to form a ternary predicate. The roles indexed by i in the two original predicates are merged in the result. If there had been no overlap in the indices the result would have been a quaternary predicate and if the roles of both binary predicates had been indexed by i and j then the result would have been a binary predicate. Thus even though the parameters are bound, the indices are freely available and can be used to encode anaphoric relations. Note that it is important here that we are allowing arbitrary indices rather than, say, always using an initial segment of the natural numbers to do our indexing. It is the fact that we are allowed to use arbitrary indices which will give us the freedom to use them to encode discourse anaphoric relations.
Given the machinery for Aczel-Lunnon abstraction we have sketched here it is quite straightforward to give a general definition of $\oplus$.

(43) **Definition of $\oplus$**

If ζ is a predicate with role indices $r_{\zeta 1}, \ldots, r_{\zeta n}$, ξ is a predicate with role indices $r_{\xi 1}, \ldots, r_{\xi m}$ and f is an assignment whose domain $\{r_1, \ldots, r_k\} = \{r_{\zeta 1}, \ldots, r_{\zeta n}\} \cup \{r_{\xi 1}, \ldots, r_{\xi m}\}$ which assigns a unique parameter X_i (which is distinct from any free parameter in ζ or ξ) to each r_i in its domain then

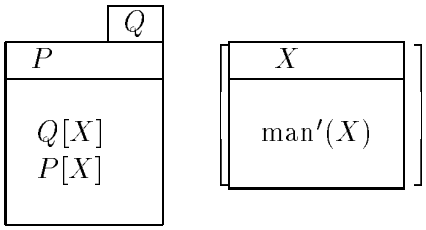
$\zeta \oplus \xi =$

$r_1 \rightarrow X_1, \ldots, r_k \rightarrow X_k$
ζf ξf

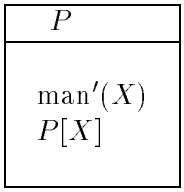
The DYANA work characterized a basic DRT fragment based on part of the fragment defined in [Kamp and Reyle, 1993] including donkey anaphora, quantified sentences and relative clauses. It exploited the fact that the use of Aczel-Lunnon abstraction allows us to use abstracts not only to recreate DRSs but also to combine that with the use of abstraction for compositional interpretation as in Montague's semantics. By way of example we give a sketch of the derivation of *a man loves a woman* according to the grammar.

(44) **A man loves a woman**

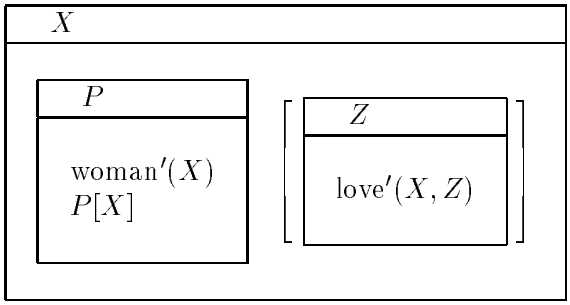
a man $\Longrightarrow$



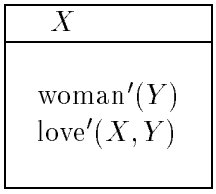
$=$



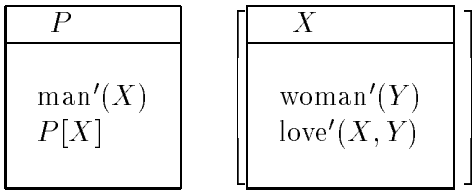
loves a woman $\Longrightarrow$



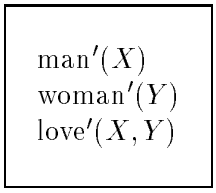
$=$



a man loves a woman $\Longrightarrow$



$=$



Note that what we have obtained in the derivation as represented in (44) is just a parametric infon and not the abstract that we will actually use to model the DRS. In order to obtain the abstract we need to use a rule of discourse interpretation which will abstract over the free parameters. In more complex sentences such as conditionals parameters corresponding to indefinites are abstracted over before the root sentence is reached. Parameters introduced by proper names are always abstracted over at the root sentence level.
This fragment has been implemented by Julian Day and Phil Kime and documented in DYANA deliverables. There are clear similarities to λ-DRT, although it is not clear at the moment that the precise details of the λ-DRT treatments can be encoded in this formalism. This needs further investigation.

3.1.5 Conclusion

We have reviewed work conducted on the DYANA project using situation theory to formulate both Montague semantics and DRT and to put the two together. The contributions of the present review have been to examine in some more detail what contribution the situation theoretic approach might make to a general computational semantic formalism and we isolated three features which seem to distinguish such a formalism from more standard ones: Austinian propositions, simultaneous abstraction and restrictions. We have explained some of the uses to which these features are put. The STDRT implementation of the DRT fragment by Day and Kime can be seen in part as an implementation of such a formalism including the main inference rules that are important for reductions in compositional semantics.

Appendix A

Program Listings

A.1 [Johnson & Klein,1986]

A.1.1 A Toy Grammar

```
%%% john_klein.pl
%%% Johnson & Klein
%%% COLING 86

%%% reformulated with Prolog terms

%%% FraCaS

?- op(500,xfy,:).
?- op(300,xfx,==>).            % complex Drs condition

%%% grammar

dis(SemIn-SemOut) -->
  s(SemIn-SemOut).

dis(SemIn-SemOut) -->
  s(SemIn-SemMid),
  dis(SemMid-SemOut).

s(Sem) -->
  np(Arg1,ScopeSem,Sem),
  vp(Arg1,ScopeSem).

s(SemIn-[[(Ante ==> Cons)|Current]|Super]) -->
  [if],
  s([[]|SemIn]-S1SemOut),
  s([[]|S1SemOut]-[Cons,Ante|[Current|Super]]).
```

```
np(Index,ScopeSem,Sem) -->
  det(ResSem,ScopeSem,Sem),
  n(Index,ResSem).

vp(Arg1,Sem) -->
  vt(Arg1,Arg2,ScopeSem),
  np(Arg2,ScopeSem,Sem).

vp(Arg1,Sem) -->
  vi(Arg1,Sem).

n(w,[Current|Super]-[[w,woman(w)|Current]|Super]) --> [woman].
n(m,[Current|Super]-[[m,man(m)|Current]|Super])   --> [man].

vt(Arg1,Arg2,
  [Current|Super]-[[see(Arg1,Arg2)|Current]|Super])  --> [saw].
vt(Arg1,Arg2,
  [Current|Super]-[[like(Arg1,Arg2)|Current]|Super]) --> [liked].
vi(Arg1,
  [Current|Super]-[[leave(Arg1)|Current]|Super])     --> [left].

det(SemIn-ResOut,ResOut-SemOut,SemIn-SemOut) --> [a].

det([[]|SemIn]-ResOut,
  [[]|ResOut]-[Scope,Res|[Current|Super]],
  SemIn-[[(Res ==> Scope)|Current]|Super])   --> [every].

np(w,SemIn-SemOut,SemIn-SemOut) --> [her],
  {member(Space,SemIn),
   member(w,Space)}.

np(w,SemIn-SemOut,SemIn-SemOut) --> [she],
  {member(Space,SemIn),
   member(w,Space)}.

np(m,SemIn-SemOut,SemIn-SemOut) --> [him],
  {member(Space,SemIn),
   member(m,Space)}.

np(m,SemIn-SemOut,SemIn-SemOut) --> [he],
  {member(Space,SemIn),
   member(m,Space)}.

member(X,[X|_]).
member(X,[_|Y]) :-
  member(X,Y).
```

A.2 [Johnson & Kay,1990]

A.2.1 A Toy Grammar

```
%%% john_kay.pl
%%% Johnson & Kay
%%% Coling 90
%%% some typos removed
%%% FraCaS

%%% :- ['pred_logic_constructors.pl'].
%%% :- ['drt_constructors.pl'].
%%% :- ['set_of_infons_constructors.pl'].

:- op(950,xfy,^).
:- op(300,xfx,==>).

%%% top level

parse(String,ExtSem) :-
    external(IntSem,ExtSem),
    dis(IntSem,String,[]).

%%% grammar

dis(D) -->
  s(S),
  dis(RestD),
  {compose(S,RestD,D)}.
dis(D) --> s(D).

s(S) -->
  np(VP^S),
  vp(VP).

np(NP) -->
  det(N1^NP),
  n1(N1).

n1(N) -->
  n(N).
n1(X^S) -->
  n(X^S1),
  rc(X^S2),
  {conjoin(S1,S2,S)}.

vp(X^S) -->
  v(X^VP),
  np(VP^S).

rc(VP) -->
```

```
  [that],
  vp(VP).

v(X^Y^S) --> [Verb],
  {verb(Verb,X^Y^Pred),
   atom(Pred,S)}.

n(X^S) --> [Noun],
  {noun(Noun,X^Pred),
   new_index(X,S1),
   atom(Pred,S2),
   compose(S1,S2,S)}.

det((X^Res)^(X^Scope)^S) --> [Det],
  {determiner(Det,Res^Scope^S)}.

np((X^S1)^S) --> [Pronoun],
  {pronoun(Pronoun),
   compose(S1,S2,S),
   accessible_index(X,S2)}.

%%% lexicon

pronoun(he).
pronoun(she).
pronoun(him).
pronoun(her).
pronoun(it).

verb(likes,X^Y^like(X,Y)).
verb(saw,X^Y^saw(X,Y)).
verb(beats,X^Y^beat(X,Y)).
verb(owns,X^Y^own(X,Y)).

noun(woman,X^woman(X)).
noun(man,X^man(X)).
noun(donkey,X^donkey(X)).

determiner(a,Res^Scope^S) :-
  conjoin(Res,Scope,S).
determiner(every,Res0^Scope^S) :-
  compose(S1,S2,S),
  subordinate(Res,ResName,S1),
  compose(Res0,Res1,Res),
  subordinate(Scope,ScopeName,Res1),
  atom(ResName ==> ScopeName,S2).

member(X,[Y|_]) :-
  freeze(Y,X=Y).
member(X,[_|Y]) :-
  freeze(Y,member(X,Y)).
```

A.2.2 DRT Constructors

```
%%% drt_constructors.pl

atom(P,[B|Bs]-[[P|B]|Bs]).

conjoin(P1,P2,P) :-
  compose(P1,P2,P).

new_index(Index,C) :-
  atom(i(Index),C).

accessible_index(Index,Bs-Bs) :-
  member(B,Bs),
  member(i(Index),B).

compose(B0s-B1s,B1s-B2s,B0s-B2s).

subordinate([[]|B0s]-[B|B1s],B,B0s-B1s).

external([[]]-[S],S).
```

A.2.3 SitSem Constructors

```
%%% set_of_infons_constructors.pl
:- op(900,xfx,:).

atom(P,@([Sit|_],Is,[(Sit:P)|Is])).

conjoin(I1,I2,I12) :-
  compose(I1,I2,I12).

new_index(Index,S) :-
  atom(i(Index),S).

accessible_index(Index,@(Ss,Is,Is)) :-
  member(Sit:i(Index),Is),
  member(Sit,Ss).

compose(@(Ss,I0s,I1s),
  @(Ss,I1s,I2s),
  @(Ss,I0s,I2s)).

subordinate(@([Sit|Sits],I0s,I1s),Sit,@(Sits,I0s,I1s)) :-
  gensym(Sit).

external(@([Sit],[],Is), Sit:Is) :-
  gensym(Sit).

gensym(Sit) :-
  numbervars(Sit,0,_).
```

A.2.4 PL Constructors

```
%%% pred_logic_constructors.pl

:- op(300,xfy,&).

atom(Prop,Prop).

conjoin(P,Q,P&Q).

new_index(_,_).

accessible_index(_,_).

compose(P,P,P).

subordinate(Sub,Sub,_).

external(P,P).
```

A.2.5 Grammar with Cooper Storage

```
%%% john_kay_quants.pl
%%% Coling 90
%%% some typos removed FraCaS

%%% :- ['pred_logic_constructors.pl'].
%%% :- ['drt_constructors.pl'].
%%% :- ['set_of_infons_constructors.pl'].

:- op(950,xfy,^).
:- op(300,xfx,==>).

%%% top level

parse(String,ExtSem) :-
  external(IntSem,ExtSem),
  s(IntSem,[],String,[]).

%%% grammar

s(S,Qs) -->
  np(VP^S1,Qnp),
  vp(VP,Qvp),
  {shuffle(Qnp,Qvp,Q1s),
   apply_some(Q1s,S1,Qs,S)}.

np(NP,Qnp) -->
  det(N1^NP,Qdet),
  n1(N1,Qn1),
  {append(Qdet,Qn1,Qnp)}.
```

```
n1(N,Qn) -->
  n(N,Qn).
n1(X^S,Qn1) -->
  n(X^S1,Qn),
  rc(X^S2,Qrc),
  {conjoin(S1,S2,S),
   shuffle(Qn,Qrc,Qn1)}.

vp(X^S,Qvp) -->
  v(X^VP,Qv),
  np(VP^S,Qnp),
  {shuffle(Qv,Qnp,Qvp)}.

rc(X^S2,Qrc) -->
  [that],
  vp(X^S1,Qvp),
  {apply_some(Qvp,S1,Qrc,S2)}.

v(X^Y^S,[]) --> [Verb],
  {verb(Verb,X^Y^Pred),
   atom(Pred,S)}.

n(X^S,[]) --> [Noun],
  {noun(Noun,X^Pred),
   new_index(X,S1),
   atom(Pred,S2),
   compose(S1,S2,S)}.
det((X^Res)^(X^Scope)^Scope,[Quant]) --> [Det],
   determiner(Det,Res^Quant)}.

np((X^S1)^S,[]) --> [Pronoun],
  {pronoun(Pronoun),
   accessible_index(X,S2),
   compose(S1,S2,S)}.

%%% lexicon

pronoun(he).
pronoun(she).
pronoun(him).
pronoun(her).
pronoun(it).

verb(likes,X^Y^like(X,Y)).
verb(saw,X^Y^saw(X,Y)).
verb(beats,X^Y^beat(X,Y)).
verb(owns,X^Y^own(X,Y)).

noun(woman,X^woman(X)).
noun(man,X^man(X)).
```

```
noun(donkey,X^donkey(X)).

determiner(a,Res^Scope^S) :-
  conjoin(Res,Scope,S).
determiner(every,Res0^Scope^S) :-
  compose(S1,S2,S),
  subordinate(Res,ResName,S1),
  compose(Res0,Res1,Res),
  subordinate(Scope,ScopeName,Res1),
  atom(ResName ==> ScopeName,S2).

%%% utility predicates

append([],L,L).
append([H1|T1],L2,[H1|T3]) :-
  append(T1,L2,T3).

member(X,[Y|_]) :-
  freeze(Y,X=Y).
member(X,[_|Y]) :-
  freeze(Y,member(X,Y)).

shuffle([],[],[]).
shuffle([Q|Q1s],Q2s,[Q|Q3s]) :-
  shuffle(Q1s,Q2s,Q3s).
shuffle(Q1s,[Q|Q2s],[Q|Q3s]) :-
  shuffle(Q1s,Q2s,Q3s).

apply_some(Qs,P,Qs,P).
apply_some([P^Qp|Qs],P,Q1s,P1) :-
  apply_some(Qs,Qp,Q1s,P1).
```

Bibliography

[Aczel and Lunnon, 1991] Aczel, P. and Lunnon, R. 1991. Universes and parameters. In Barwise, J.; Gawron, J. M.; Plotkin, G.; and Tutiya, S., editors 1991, *Situation Theory and its Applications, volume II.* CSLI and University of Chicago, Stanford. chapter 1, 3–24.

[Ahn and Kolb, 1990] Ahn, R. and Kolb, H. 1990. Discourse representation theory meets constructive mathematics. ITK Research Report Number 16, Tilburg University.

[Barwise and Cooper, 1991] Barwise, J. and Cooper, R. 1991. Simple situation theory and its graphical representation. In Seligman, J., editor 1991, *Partial and Dynamic Semantics III.* DYANA ESPRIT Basic Research Project.

[Barwise and Cooper, 1993] Barwise, J. and Cooper, R. 1993. Extended Kamp notation. In Aczel, P.; Israel, D.; Katagiri, Y.; and Peters, S., editors 1993, *Situation Theory and its Applications, v.3.* CSLI. chapter 2, 29–54.

[Bos *et al.*, 1994a] Bos, J.; Mastenbroek, E.; McGlashan, S.; Millies, S.; and Pinkal, M. 1994a. A compositional DRS-based formalism for NLP-applications. In *International Workshop on Computational Semantics*, Tilburg.

[Bos *et al.*, 1994b] Bos, J.; Mastenbroek, E.; McGlashan, S.; Millies, S.; and Pinkal, M. 1994b. The VERBMOBIL semantic formalism. VM-Report 6, Universität des Saarlandes.

[Chomsky, 1981] Chomsky, N. 1981. *Lectures on Government and Binding.* Foris, Dordrecht.

[Church, 1951] Church, A. 1951. A formulation of the logic of sense and denotation. In Henle, P.; Kallen, H.M.; and Langer, S.K., editors 1951, *Structure, Method and Meaning.* Liberal Arts Press. 3–24.

[Cooper, 1996] Cooper, R. 1996. The role of situations in generalized quantifiers. To appear in Handbook of Contemporary Semantic Theory, ed. by S. Lappin.

[Coquand and Huet, 1985] Coquand, T. and Huet, G. 1985. A theory of constructions. In *Semantics of Data Types*. Springer-Verlag.

[Coquand, 1990] Coquand, T. 1990. Metamathematical investigations of the calculus of constructions. In Odifreddi, P., editor 1990, *Logic and Computer Science*. Academic Press.

[Davila-Perez, 1994] Davila-Perez, Rogelio 1994. Constructive type theory and natural language. Computer Science Memorandum 206, University of Essex.

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

[Henson, 1989] Henson, M.C. 1989. Program development in the constructive set theory TK. *Formal Aspects of Computing* 1:173–192.

[Holt, 1993] Holt, L. 1993. *Abstracting over Semantic Theories*. Ph.D. Dissertation, University of Edinburgh.

[Johnson and Kay, 1990] Johnson, M. and Kay, M. 1990. Semantic abstraction and anaphora. In *COLING 90, Proceedings of the Conference, Helsinki, Finland*. 17–27.

[Johnson and Klein, 1986] Johnson, M. and Klein, E. 1986. Discourse, anaphora and parsing. In *COLING 86, Proceedings of the Conference, Bonn, Germany*. 669–675.

[Kamp and Reyle, 1993] Kamp, H. and Reyle, U. 1993. *From Discourse to Logic*. Kluwer, Dordrecht.

[Kamp, 1981] Kamp, H. 1981. A theory of truth and semantic representation. In Groenendijk, J. and others, , editors 1981, *Formal Methods in the Study of Language*. Mathematisch Centrum, Amsterdam.

[Kamp, 1990] Kamp, H. 1990. Prolegomena to a structural account of belief and other attitudes. In Anderson, C. A. and Owens, J., editors 1990, *Propositional Attitudes—The Role of Content in Logic, Language, and Mind*. University of Chicago Press and CSLI, Stanford. chapter 2, 27–90.

[Kaplan and Bresnan, 1982] Kaplan, R.M. and Bresnan, J. 1982. Lexical functional grammar. In Bresnan, J., editor 1982, *The mental representation of grammatical relations*. MIT Press, Cambridge Mass. 173–281.

[Keller, 1988] Keller, W.R. 1988. Nested Cooper storage. In Reyle, U. and Rohrer, C., editors 1988, *Natural Language Parsing and Linguistic Theory.* D. Reidel, Dordrecht, The Netherlands. 432–447.

[Kuschert, 1995] Kuschert, S. 1995. Eine erweiterung des λ-kalküls um diskursrepräsentationsstrukturen. Master's thesis, Universität des Saarlandes.

[Martin-Löf, 1982] Martin-Löf, P. 1982. Constructive mathematics and computer programming. In Cohen, ; Los, ; Pfeiffer, ; and Podewski, , editors 1982, *Logic, Methodology and Philosophy of Science VI.* North Holland. 153–179.

[Martin-Löf, 1984] Martin-Löf, P. 1984. *Intuitionistic Type Theory.* Bibliopolis.

[Millies and Pinkal, 1993] Millies, S. and Pinkal, M. 1993. Linking one semantic interpretation system to different syntactic formalisms. Extended Abstract of the Dagstuhl Colloquium on Semantic Formalisms in Natural Language Processing.

[Mohring, 1986] Mohring, C. 1986. Algorithm development in the calculus of constructions. In *Proceedings of the IEEE Symposium on Logic in Computer Science*, New York. IEEE.

[Muskens, 1989] Muskens, R. 1989. *Meaning and Partiality.* Ph.D. Dissertation, University of Amsterdam.

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

[Pereira and Shieber, 1987] Pereira, F.C.N. and Shieber, S.M. 1987. *Prolog and Natural Language Analysis*, volume 10 of *CSLI Lecture Notes.* CSLI, Stanford. Distributed by University of Chicago Press.

[Pollard and Sag, 1994] Pollard, C. and Sag, I. 1994. *Head-Driven Phrase Structure Grammar.* CSLI Lecture Notes. CSLI, Stanford. Distributed by University of Chicago Press.

[Ranta, 1991] Ranta, Aarne 1991. Intuitionistic categorial grammar. *Linguistics and Philosophy* 14:203–239.

[Ruhrberg, 1994] Ruhrberg, P. 1994. A simultaneous abstraction calculus. In Cooper, R. and Groenendijk, J., editors 1994, *DYANA-2 Deliverable R2.1.B, Part II.* The DYANA-2 Project Administrator, ILLC, University of Amsterdam. 65–82.

[Sannella and Tarlecki, 1988] Sannella, D. and Tarlecki, A. 1988. Toward formal development of programs from algebraic specifications: Implementations revisited. *Acta Informatica* 25:233–281.

[Sannella and Tarlecki, 1992] Sannella, D. and Tarlecki, A. 1992. Toward formal development of programs from algebraic specifications: Model-theoretic foundations. In *Proceedings of the Nineteenth International Colloquium on Automata, Languages and Programming*. Springer Verlag. 32–38.

[Sannella and Wirsing, May 1983] Sannella, D. and Wirsing, M. 1983. A kernel language for algebraic specification and implementations. Technical report, Department of Computer Science, University of Edinburgh.

[Smith, 1984] Smith, Jan M. 1984. An interpretation of Martin-Löf's Type Theory in a type-free theory of propositions. *Journal of Symbolic Logic* 49.

[Sundholm, 1989] Sundholm, G. 1989. Constructive generalised quantifiers. *Synthese* 79:1–12.

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

[Turner, 1990] Turner, R. 1990. *Truth and Modality for Knowledge Representation*. Pitman.

[Turner, 1991] Turner, R. 1991. *Constructive Foundations for Functional Languages*. McGraw-Hill.

[Turner, 1992] Turner, R. 1992. Properties, propositions and semantic theory. In Rosner, M. and Johnson, R., editors 1992, *Computational Linguistics and Formal Semantics*. CUP, Cambridge.