CHAPTER III

INTENSIONAL LOGIC

ABSTRACT

In this chapter the language of intensional logic is introduced; this language is a useful tool for representing meanings of e.g. English. The semantic interpretation of intensional logic is defined by a translation into the language Ty2 of two-sorted type theory. Several properties of intensional logic are explained using this translation into Ty2.
1. TWO FACETS

1.1. Introduction

Our aim is to associate in a systematic way the expressions of a language with their meanings. Hence we need a method to represent meanings. The most convenient way to do so, is to use some suitable logical language. Once the interpretation of that language has been defined, it can further be used to represent meanings. The language we will use in this book, is the language of intensional logic, henceforth IL. This language is especially suitable for representing the intended meanings because it 'wears its interpretation upon its sleeves' (Van Benthem, pers.comm.).

In chapter I some consequences of the principle of compositionality are discussed. Here I will pay special attention to two of them.

I) The meanings associated with expressions of a natural language or a programming language are intensions, i.e. functions on a domain consisting of a set of 'indices'. The indices formalize several factors which influence the meaning of an expression.

II) The meanings form a many sorted algebra which is similar to the syntactic algebra. Hence we have for each category in the syntactic algebra a corresponding sort in the semantic algebra: the semantic model is 'typed'.

In the light of the close connection between IL and its models, it is not surprising that these two facets of meaning are reflected in IL. This language contains operators connected with indices (e.g. tense operators), as well as operators reflecting the typed structure of the semantic domain (e.g. λ abstraction). This means that IL can be considered as the amalgamation of two kinds of languages: type logic and modal tense logic. With this characterization in mind, many properties of IL can be explained. This will be done in the sequel.

1.2. Model - part I

The set of indices plays an important role in the formalization of the notion 'meaning' since meanings are functions with indices as their domain. The definition of the model will not say much about what indices are: they are defined as an arbitrary set. This level of abstraction has the advantage that the meanings of such different languages as English and Algol can be described by them. But one might like to have an intuitive understanding of what indices are, what they are a formal counterpart of, and which degree of reality they have. Several views on these issues are possible, and I will
mention some of them. Thereafter I will give my personal opinion.

1. Our semantic theory gives a model of how the reality is, or might have been. An index represents one of these possibilities. In application to natural language this means that an index represents a possible state of affairs of the reality. In application to programming languages this means that an index represents a possible internal state of the computer.

2. Our semantical theory gives a model of a psychologically acceptable way of dealing with meanings. In this conception an index formalizes a perceptually possible state of affairs (cf. PARTEE 1977b).

3. Languages describe concepts, and users of a language are equipped with a battery of identification procedures for such concepts. An index represents a class of possible outcomes of such procedures (cf. TICHY 1971).

4. Our semantic theory describes how we deal with data. An index represents a maximal, non-contradictory set of data (cf. VELTMAN 1981).

5. 'In order to say what meaning is, we may first ask what a meaning does, and then find something that does that.' (LEWIS 1970). We want meanings to do certain things (e.g. formalize implication relations among sentences), we define meanings in an appropriate way, and indices form a technical tool which is useful to this purpose. Indices are not a formalization of something; they are just a tool.

Conception 1 is intuitively very appealing, and most widespread in the literature. But the interpretations 2, 3, and 4 are also intuitively appealing. The reader is invited to choose that conception he likes best. An intuitively conceivable interpretation might help him to understand how and why everything works. But the reader should only stick to his interpretation as long as it is of use to him. For the simple cases indices can probably be considered as an adequate formalization of his intuitions. But once comes the day that his intuition does not help him any more. Then he should switch to conception 5; no interpretation but a technical tool. Such a situation arises, for instance, with the treatment of questions. Do you have an idea of what the meaning of a question should be in the light of conception 1, 2, 3, or 4? For instance the treatment of indirect questions given in GROENENDIJK & STOKHOF (1981) cannot be explained on the basis of the first four conceptions. They have chosen as meanings those semantic entities which do what they wanted them to do: the indices play just a technical role.
1.3. Model - part II

In the model theory of type-logic the models are constructed from a few basic sets by adding sets of functions between already available sets. Two kinds of models can be distinguished, depending on how many functions are added. In the so called 'standard models', the addition clause says that if A and B are sets in the model, then the set $A^B$ of all functions from B to A also is a set in the model. In the so called 'generalized models' one needs not to take this whole set, but one may take some subset. There is a condition on the construction of models which guarantees that not too few elements are added to the model: every object that can be described in the logic should be incorporated in the model.

The laws of type logic which hold in standard models are not axiomatizable. In order to escape this situation, the generalized models were introduced (HENKIN 1950). By extending the class of possible models, the laws were restricted to an axiomatizable class: the more models the more possible counter examples, and therefore the fewer laws.

What kind of models will be used for the interpretation of intensional logic? I mention four options.

1. the class of all standard models
2. a subclass of the standard models
3. the class of all generalized models
4. a subclass of the generalized models.

Which choice is made, depends on the application one has in mind, and what conception one has about the role of the model (see section 1.2). If one intends to model certain psychological insights, then one might argue that the generalized models with countably many elements are the best choice (cf. PARTEE 1977b, and the discussion in chapter 7). If the model is used for dealing with the semantics of programming languages then a certain subset of the generalized models is required (see chapter 4). In the application of Montague grammar to natural language, one works with option 2. A subclass of the standard models is characterized by means of meaning postulates which give restrictions on the interpretation of the constants of the logic. I will follow this standard approach in the sequel.

1.4. Laws

Most of the proof-theoretic properties of IL can be explained by considering IL as the union of two systems: type logic and modal tense logic. The modal laws of IL are the laws of the modal logic S5. Many of the laws
of type logic are laws of IL, exceptions are variants of the laws which are not valid in modal logic. The laws of type logic (i.e. those which hold in all standard models) are not axiomatizable. Since IL has (on sorted) type logic as a sublanguage, IL is not axiomatizable either. For modal logic there is an axiomatization of the laws which hold in all generalized models. This is expressed by saying that type logic has the property of generalized completeness. By combining these two completeness results, the generalized completeness of IL can be proved (see also section 3).

1.5. Method

I have explained that many aspects of IL can be understood by considering IL as the amalgamation of type logic and modal tense logic. Nevertheless, the formal introduction of IL will not proceed along this line. I will first introduce some other language: Ty2, the language of two sorted type theory. On the basis of Ty2 I will define IL: the algebraic grammar of IL is an algebra derived from the algebraic grammar for Ty2. The reasons for preferring this approach are the following:

1. Modal theoretic
In Ty2 the indices are treated as elements of a certain type just like all elements. This is not the case for IL. In the interpretation of IL indices occur only as domains of certain functions, but not as range. Therefore the models for IL become structures in which carriers of certain types are deleted, whereas, from the viewpoint of an elegant construction, these carriers should be there. In the models for Ty2 they are there. Remarkable properties of IL can be explained from the fact that these carriers are not incorporated in its models. It appears to be better for understanding, and technically more convenient, to describe first the full model, and to remove next certain sets, instead of to start immediately with the remarkable model.

2. Homomorphisms
From the viewpoint of our framework, it is essential to demonstrate that the interpretation of IL is a homomorphism. It seems, however, rather difficult to show that the interpretation homomorphism for type logic and that for modal tense logic can be combined to a single homomorphism for IL. Furthermore, we should, in such an approach, consider first the interpretations of these two languages separately. It is easier to consider only Ty2.
3. Laws

Many of the proof rules for Ty2 are easy to formulate and to understand. This is not the case with IL. It is for instance much easier to prove lambda conversion first for Ty2, and derive from this the rule for IL, than to prove the IL rule directly.

4. Speculation

We will use IL for expressing meanings of natural language expressions because it is a suitable language for that purpose. No explicit reference to indices is possible in IL, and there is no need to do so for the fragment we will consider. But one may expect that for larger fragments it is unavoidable to have in the logic explicit reference to indices. NEEDHAM (1975) has given philosophical arguments for this opinion, Van BENTHEM (1977) has given technical arguments, and GROENENDIJK & STOKHOF (1981) treat a fragment of natural language where the use of Ty2 turned out to be required. Furthermore we will consider in chapter 4 a kind of semantics for programming languages which requires that states can be mentioned explicitly in the logical language, and we will use Ty2 for that purpose.

2. TWO-SORTED TYPE THEORY

In this section the language Ty2 will be defined: the language of two sorted type theory. The name (due to GALLIN 1975) reflects that the language has two basic types (besides the type of truth values). It is a generalization of one sorted type theory which has (besides the type of truth values) one basic type. The language is defined here by means of an algebraic grammar.

Since in logic it is customary to speak of types, rather than of sorts, I will use this terminology, even in an algebraic context. The collection of types of Ty2 is the smallest set Ty such that

1. \{e,s,t\} \subset Ty (e='entity', s='sense', t='truth value').
2. If \(\sigma \in Ty\) and \(\tau \in Ty\) then \(\langle \sigma, \tau \rangle \in Ty\).

This is the standard notation for types which is used in Montague grammar. It is, however, not the standard notation in type theory. Following CHURCH (1940), the standard notation is \((\sigma)\tau\) instead of \(\langle \sigma, \tau \rangle\). I agree with LINK & VARGA (1975) that if we would adopt that notation and some standard conventions from type theory, this would give rise to a simpler notation than
the one defined above. But I prefer not to confuse readers familiar with Montague grammar, and therefore I will use his notation.

For each type $\tau \in Ty$ we have two denumerable sets of symbols:

\[ \text{CON}_\tau = \{ c_{1,\tau}, c_{2,\tau}, \ldots \} \]  
the constants of type $\tau$

and

\[ \text{VAR}_\tau = \{ \nu_{1,\tau}, \nu_{2,\tau}, \ldots \} \]  
the variables of type $\tau$.

So the constants and variables are indexed by a natural number and a type symbol. The elements of $\text{VAR}_\tau$ are called variables since they will be used in $Ty_2$ as variables in the logical sense (they should not be confused with variables in the algebraic sense which occur in polynomials over $Ty_2$).

The generators of type $\tau$ are the variables and constants of type $\tau$. The carrier of type $\tau$ is denoted as $\text{ME}_\tau$ (meaningful expression of type $\tau$). An element of the algebra is called a (meaningful) expression. The standard convention is to call the meaningful expressions of type $\tau$ 'formulas', but I will call all meaningful expressions 'formulas'. (This gives the possibility to distinguish them easily from expressions in other languages).

There are denumerable many operators in the algebra of $Ty_2$, because the operators defined below are rather schemes of operators, in which the types involved occur as parameters. For instance the operator for equalities (i.e. $R_=$) corresponds with a whole class of operators: for each type $\tau \in Ty$ there is an operator $R_{=,\tau}$. These operators $R_{=,\tau}$ all have the same effect: $R_{=,\tau}(\alpha, \beta)$ is defined as being the expression $[\alpha = \beta]$. Therefore we can define a whole class with a single scheme. The scheme for $R_{=}$ should contain $\tau$ as a parameter, and other operations should contain two types as parameter. These types are not explicitly mentioned as parameter, since they can easily be derived from the context. The proliferation of operators just sketched is a consequence of the algebraic approach and caused by the fact that, if two expressions belong to different sorts for one operation (say for function application), they belong to different sorts for all operations (so for equality).

The operators of $Ty_2$ are defined as follows

1. **Equality**

   \[ R_\tau: \text{ME}_\tau \times \text{ME}_\tau \to \text{ME}_\tau \quad \text{where} \quad R_\tau(\alpha, \beta) = [\alpha = \beta]. \]

2. **Function Application**

   \[ R(\_): \text{ME}_{\sigma,\tau} \times \text{ME}_\sigma \to \text{ME}_\tau \quad \text{where} \quad R(\_)(\alpha, \beta) = [\alpha(\beta)]. \]
3. **Quantification**

\[ R_{\exists \nu} : ME_t \rightarrow ME_t \quad \text{where } \nu \in VAR_t \text{ and } R_{\exists \nu}(\phi) = \exists \nu[\phi]. \]

For universal quantification \( R_{\forall \nu} \) analogously. Recall that in chapter 1 arguments were given for not analyzing \( \exists \nu \) any further.

4. **Abstraction**

\[ R_{\lambda \nu} : ME_t \rightarrow ME_{<\sigma,t>} \quad \text{where } \nu \in VAR_\sigma \text{ and } R_{\lambda \nu}(\alpha) = \lambda \nu[\alpha]. \]

5. **Connectives**

\[ R_\wedge : ME_t \times ME_t \rightarrow ME_t \quad \text{where } R_\wedge(\alpha, \beta) = [\alpha \wedge \beta]. \]

Analogously for \( R_\vee, R_\rightarrow, \) and \( R_{\leftrightarrow}. \)

\[ R_{\neg} : ME_t \rightarrow ME_t \quad \text{where } R_{\neg}(\phi) = \neg[\phi]. \]

The (syncategorematic) symbols \([\quad]\) and \([\quad]\) are used to guarantee unique readability of the formulas. They will be omitted when no confusion is likely. The syncategorematic symbols \( \exists \nu \) (existential quantifier), \( \forall \nu \) (universal quantifier) and \( \lambda \nu \) (the lambda-abstraction) are called binders. A variable is called free when it does not occur within the scope of \( \exists \nu, \forall \nu, \) or \( \lambda \nu. \) The notions 'scope' and 'free' can be defined rigorously in the usual way.

This completes the definition of the operators of \( Ty2. \) For the semantics of English, two more operators are needed. They introduce ordering symbols between expressions of type \( s \) (i.e. between index expressions).

6. **Ordering**

\[ R_\lt : ME_s \times ME_s \rightarrow ME_t \quad \text{where } R_\lt(\alpha, \beta) = [\alpha \lt \beta]. \]

\[ R_\gt : ME_s \times ME_s \rightarrow ME_t \quad \text{where } R_\gt(\alpha, \beta) = [\alpha \gt \beta]. \]

After having described the sets of sorts, generators and operators of \( Ty2, \) I will present the algebraic grammar for \( Ty2. \) Let \( R \) be the collection of operators introduced in clauses 1–5. Then an algebraic grammar for \( Ty2 \) is

\[ \langle \langle (CON \cup VAR)_{t \in Ty}, R_{\lt, \gt} \rangle. \]

If \( R \) is replaced by \( R \cup \{ R_\lt, R_\gt \} \); then an algebraic grammar for \( Ty2< \) is obtained.
3. THE INTERPRETATION OF Ty2

The semantic domain in which Ty2 will be interpreted, consists of a large collection of sets, which are built from a few basic ones. These basic sets are the set A of entities, the set I of indices, and the set \(\{0,1\}\) of truth values. The sets \(D_\tau\) of possible denotations of type \(\tau\) are defined by:

1. \(D_0 = \{0,1\}\), \(D_e = A\), \(D_s = I\)
2. \(D_{\langle a, \tau \rangle} = D_a^{D_0}\).

In order to deal with logical variables, we need functions which assign semantical objects to them. The collection \(\text{AS}\) of variable assignments (based on \(A\) and \(I\)) is defined by

\[
\text{AS} = \prod_{\tau \in \text{Ty}} \{D_\tau\}.
\]

Let \(a, a' \in \text{AS}\). We say that \(a \sim_\nu a'\) (as \(a\) is a \(\nu\)-variant of \(a'\)) if for all \(w \in \text{VAR}\) such that \(w \not\in \nu\) holds that \(as(w) = as'(w)\). If \(a \sim_\nu a'\) and \(a'(\nu) = d\) then we write \([\nu, d]a\) as for \(a'\).

Now the necessary preparations are made to say what the elements of the semantic domains are. Let \(A\) and \(I\) be non-empty sets, and let \(D_\tau\) be defined as above. The sets \(M_\tau\) of meanings of types \(\tau\) (based upon \(A\) and \(I\)) are:

\[
M_\tau = D_\tau^{\text{AS}}.
\]

By the semantic domain based upon \(A\) and \(I\), we understand the collection \(\langle M_\tau \rangle_{\tau \in \text{Ty}}\). In such domains we will interpret Ty2. For Ty2< additional structure is required. The set \(I\) has to be the cartesian product of two sets \(W\) and \(T\), where \(T\) is linearly ordered by a relation \(<\). Here \(W\) is called the collection of possible worlds, and \(T\) the collection of time points. An element \(i \in W \times T\) is called a reference point or index.

As is suggested by the definition of semantic domain, the interpretation homomorphism of Ty2 will assign to an expression of type \(\tau\) some element of \(M_\tau\), i.e. the meaning of \(\phi \in M_\tau\) is some function \(f: \text{AS} \to D_\tau\). In chapter one we have formalized the meaning of an expression of predicate logic as a function \(f: \text{AS} \to D_\tau\), and here this approach is generalized to other types. In the case of predicate logic a geometrical interpretation of this process was possible, and this led us towards the cylindric algebras. For the case of Ty2 it is not that easy to find a geometrical
interpretation. In any case, I will not try to give one. But I consider
the interpretation of Ty2 given here as a generalization of the interpreta-
tion of predicate logic with cylindric algebras. Therefore I will call the
interpretation of quantifiers of Ty2 'cylindrifications'.

Analogous to the interpretation of variables, there are functions for
the interpretation of constants. The collection $F$ (based upon $A$ and $I$) of
functions interpreting constants is defined by:

$$F = \prod_{\tau \in Ty} D_{\tau}^{\tau}.$$  

By a model for Ty2 we understand a pair $\langle M, F \rangle$ where
1. $M$ is a semantic domain (based on $A$ and $I$).
2. $F \in F$, hence $F$ is a function for the interpretation of constants (based
   on the same sets $A$ and $I$).

In order to define a homomorphism from Ty2 to some model, the models
should obtain the structure of an algebra similar to the syntactic algebra
of Ty2. That means that I have to say what the carriers, the generators,
and the operators of the models are. The generators and operators will be
defined below, along with the definition of the interpretation homomorphism
$V$ (V = 'valuation'). The carrier of type $\tau$ has already been defined; viz. $M_{\tau}$.
So the value of an element of $M_{\tau}$ under this interpretation $V$ is a function
from $AS$ to $D_{\tau}$. This function will be defined by saying what its value is
for an arbitrary assignment as $\in AS$. I will write $V_{as}(a)$ instead of
$V(a)(as)$ because the former notation is the standard one.

The generators of the semantic algebra are the images of the generators
of the syntactic algebra. These are defined by
a) $V_{as}(v_{\tau,n}) = as(v_{\tau,n})$
b) $V_{as}(c_{\tau,n}) = F(c_{\tau,n})$.

As for the last clause, one should remember that we are defining the inter-
pretation with respect to some model, and that models are defined as con-
sisting of a large collection of sets and a function $F$ which interprets
the constants.

The interpretation of compound expressions of Ty2 will be defined next.
Let $R$ be some operator of the syntactic algebra of Ty2. Then the value
$V_{as}(R(a))$ will be defined in terms of $V_{as}(a)$. In this way it is determined
how $V(R(a))$ is obtained from $V(a)$. Then it is also determined how the
operator $T$, which produces $V(R(a))$ out of $V(a)$ is defined. For each clause
in the definition of $V_{as}(\alpha)$, I will informally describe which semantic operator $T$ is introduced.

1. Equality

$$V_{as}(\alpha = \beta) = \begin{cases} 1 & \text{if } V_{as}(\alpha) = V_{as}(\beta) \\ 0 & \text{otherwise.} \end{cases}$$

So $V(\alpha = \beta)$ is a function from assignments to truth values yielding 1 if $V(\alpha)$ and $V(\beta)$ get the same interpretation for that assignment. Consequently $T_{=}$ is the assignment-wise evaluated equality. To be completely correct, I have to say that there is a class of semantic operators described here and not a single one: for each type $\tau$ there is an equality operator $T_{=,\tau}$.

2. Function application

$$V_{as}(\alpha(\beta)) = V_{as}(\alpha)(V_{as}(\beta)).$$

So if $V(\alpha) \in M^{<\alpha,\tau>\sigma}$, $V(\beta) \in M^{\sigma}$, then $V(\alpha(\beta)) \in M^{\tau}$. And $T_{(\ )} : M^{<\alpha,\tau>\sigma} \times M^{\sigma} \rightarrow M^{\tau}$, where $T_{(\ )}$ is assignment-wise function application of the assignment-wise determined function to the assignment-wise determined argument.

3. Quantification

$$V_{as}(\exists v \phi) = \begin{cases} 1 & \text{if there is an } as^{\tau} \text{ such that } V_{as},(\phi) = 1. \\ 0 & \text{otherwise.} \end{cases}$$

The element $V(\exists v \phi) \in M^{\tau}$ is obtained from $V(\phi) \in M^{\tau}$ by application of $T_{\exists v}$. This operation $T_{\exists v}$ is a cylindrification operation like the ones introduced in chapter 1. $V_{as}(\forall v \phi)$ is defined analogously.

4. Abstraction

Let $v \in VAR_{\tau}$ and $\phi \in M_{\tau}$. Then $V_{as}(\lambda v \phi)$ is that function $f$ with domain $D_{\sigma}$ such that whenever $d$ is in that domain, then $f(d)$ is $V_{as}(\phi)$, where $as^{\prime} = [v \mapsto d] as$. In the sequel I will symbolize this rather long phrase as

$$V_{as}(\lambda v \phi) = \lambda d V_{as}([v \mapsto d](\phi)).$$

Here $\lambda$ might be considered as an abstraction in the meta language, but its role is nothing more than abbreviating the phrase mentioned above: 'that
function which ...'. The semantic operator $T_{\lambda y}$ corresponding to $R_{\lambda y}$ is a function from $M_T$ to $M_{<S,T>}$ where $T_{\lambda y}$ associates with an element $e \in M_T$ some function $f \in M_{<S,T>}$. This function $f$ assigns to an $as \in AS$ the function that for argument $d$ has the value $e(as')$, where $as' = [v^d]as$.

5. Connectives

\[
V_{as}(\phi \land \psi) = \begin{cases} 
1 & \text{if } V_{as}(\phi) = V_{as}(\psi) = 1 \\
0 & \text{otherwise.}
\end{cases}
\]

So $T$ is the assignment-wise evaluated conjunction.

The corresponding operators $T_{\land}, T_{\lor}, T_{\rightarrow}$ and $T_{\leftrightarrow}$ are defined analogously to $T_{\land}$: assignment-wise evaluated connectives.

This completes the interpretation of $Ty2$. For the interpretation of $Ty2<$ an additional clause is required.

6. Ordering

\[
V_{as}(\alpha < \beta) = \begin{cases} 
1 & \text{if the world component of } V_{as}(\alpha) \text{ equals the world component of } V_{as}(\beta), \text{ and the time component of } V_{as}(\alpha) \text{ is before the time component of } V_{as}(\beta) \text{ in the linear ordering of } T. \\
0 & \text{otherwise.}
\end{cases}
\]

Analogously for $V_{as}(\alpha > \beta)$.

This definition means that in case the world components of $\alpha$ and $\beta$ are different, then $V_{as}(\alpha < \beta) = 0$. The relation-symbol $<$ does not correspond with a total ordering, and consequently $\tau(\alpha < \beta)$ is not equivalent with $[\alpha > \beta \lor \alpha = \beta]$.

4. PROPERTIES OF $Ty2$

In the definition of the language $Ty2$, we introduced a lot of operators (corresponding with connectives and quantifiers). Abstraction was just one among them. In a certain sense, however, it is the most important and powerful operator. The other operators are unnecessary since they can be defined in terms of $\lambda$-operators (and $\tau$). Also expressions denoting truth values can be defined in this way. I will present the definitions (originating
from HENKIN 1963), without further explications because I will not use
their details in the sequel. They are presented here for illustrating the
central role of \( \lambda \)-abstraction in this system.

4.1. Example definitions based on \( \lambda \)-operators.

Let \( x, y \in \text{VAR}_t \) and \( f \in \text{VAR}_{<t, t>} \). Then

\[
\begin{align*}
T & = [\lambda x[x] = \lambda x[x]] \\
F & = [\lambda x[x] = \lambda x[T]] \\
f & = [\lambda x[f = x]] \\
\lambda & = [\lambda x y [\lambda f[f(x) = y] = \lambda f[f(T)])] \\
\rightarrow & = [\lambda x y [\lambda y = x]] \\
v & = [\lambda x y [\lambda y = y]]
\end{align*}
\]

Let \( \tau \in Ty \) and \( z \in \text{VAR}_t \); then:

\( \forall z A = [\lambda z A = \lambda z T] \).

4.1. END

In certain circumstances, we may simplify formulas of the form \( \lambda v[\phi](\alpha) \)
by substituting the argument \( \alpha \) for the free occurrences of \( v \) in \( \phi \). This
kind of simplification is called \( \lambda \)-reduction or \( \lambda \)-conversion. In the theory
of \( \lambda \)-calculi this reduction is known under the name \( \beta \)-reduction (\( \alpha \)-reduc-
tion is change of the bound variable \( v \)). I described above the central po-
sition of \( \lambda \)-operators. This implies that the prooftheory of \( Ty_2 \) is essen-
tially the prooftheory of \( \lambda \)-calculus. Therefore it is of theoretical im-
portance to know under what circumstances \( \lambda \)-conversion is allowed. But there
is also an important practical motivation. In the next chapters we will en-
counter frequently formulas with many \( \lambda \)-operators. Then, by \( \lambda \)-conversion,
these formulas can be reduced to a manageable size. This practical aspect
of reducing formulas is the main motivation for considering \( \lambda \)-conversion
here in detail. I start with recalling a theorem which says which kinds of
reductions are allowed in all contexts. Thereafter some theorems will be
given concerning the reduction of formulas containing \( \lambda \)-operators.

4.2. Theorem. Let \( \alpha, \alpha' \in \text{ME}_\omega \) and \( \beta, \beta' \in \text{ME}_t \) such that
a) \( \beta \) is part of \( \alpha \)
b) \( \beta' \) is part of \( \alpha' \)
c) \( \alpha' \) is obtained from \( \alpha \) by substitution of \( \beta' \) for \( \beta \).
Suppose that for all $a$ as $\alpha$ holds $\forall_{as}(\beta) = \forall_{as}(\beta')$.
Then for all $a$ as $\alpha$ holds $\forall_{as}(a) = \forall_{as}(a')$.

**Proof.** Recall that $\beta$ is a part of $\alpha$ if in the production process of $\alpha$ some rule is used which has $\beta$ as one of its arguments. Hence $\beta$ is used in a construction step $R(..., \beta, ...)$, where $R$ is an operator from the algebraic grammar for $Ty_2$. That $\forall_{as}(\beta) = \forall_{as}(\beta')$ for all as $\alpha$ as, means that $\beta'$ has the same meaning as $\beta$. This means that the present theorem is a reformulation of Theorem 6.4 in Chapter Two (here adapted for the present algebra and the present notion of meaning). Hence the same proof applies.

4.2. END

As a consequence of this theorem, two expressions with the same meaning may be replaced by each other 'salva veritate'. This is a generalization of Leibniz' principle (just as the corresponding theorem in Chapter 2) since it applies to formulas of any type to be replaced within formulas of any (other) type. The theorem provides a foundation for all reductions (simplifications) of IL-formulas we will meet in the sequel. A (sub-)formula may be replaced by a formula with the same meaning. This may even be done in case it is not yet known in which larger expression they will occur as subformula. The theorem holds due to the fact that we have an algebraic interpretation of $Ty_2$

4.3. Definition. Let $\phi \in ME_\tau$, $\alpha \in ME_\tau$ and $\nu \in VAR_\tau$. Then $[a/\nu] \phi$ denotes the formula obtained from $\phi$ by substitution of $a$ for all free occurrences of $\nu$ in $\phi$. This substitution is defined recursively as follows

\[
[a/\nu] \phi = \begin{cases}
\phi & \text{if } \nu \notin \phi, \\
[a/\nu] \eta & \text{if } \nu \notin \eta, \\
[a/\nu] \phi(\eta) & \text{if } \nu \notin \phi(\eta), \\
[a/\nu] \phi(\eta) & \text{if } \nu \notin \phi(\eta), \\
[a/\nu] \exists \phi & \exists \phi \\
[a/\nu] \forall \phi & \forall \phi
\end{cases}
\]

analogously for $\forall \phi$ and for $\lambda \phi$

\[
[a/\nu] [\phi \psi] = [a/\nu] [\phi] \land [a/\nu] [\psi]
\]

analogously for the other connectives.

4.3. END
4.4. THEOREM. Suppose no free variable in \( \phi \) becomes bound by substitution of \( \nu \) in \( \phi \). Then \( \lambda \)-conversion is allowed; i.e. for all \( \alpha \in \text{AS} \):

\[
V_{\text{as}}(\lambda \nu[\phi] \alpha) = V_{\text{as}}([\alpha/\nu] \phi).
\]

PROOF. The clause concerning function application in the definition of \( Ty_2 \) says:

\[
V_{\text{as}}(\lambda \nu[\phi] \alpha) = V_{\text{as}}(\lambda \nu[\phi])(V_{\text{as}}(\alpha)).
\]

By definition \( V_{\text{as}}(\lambda \nu[\phi]) \) is that function which for argument \( \phi \) yields value \( V_{[\nu \to \alpha]}(\phi) \). So, writing \( A \) for \( V_{\text{as}}(\alpha) \), we have

\[
V_{\text{as}}(\lambda \nu[\phi])(V_{\text{as}}(\alpha)) = V_{\text{as}}(\lambda \nu[\phi])(A) = V_{[\nu \to A]}(\phi).
\]

We first will prove, that for all \( \alpha \in \text{AS} \)

\[
V_{[\nu \to A]}(\phi) = V_{\text{as}}([\alpha/\nu] \phi).
\]

From this equality the proof of theorem easily follows. The proof of the equality proceeds with induction to the construction of \( \phi \).

1. \( \phi \equiv \sigma \), where \( \sigma \in \text{CON}_\tau \)

\[
V_{[\nu \to A]}(\sigma) = F(\sigma) = V_{\text{as}}(\sigma) = V_{\text{as}}([\lambda \nu] \sigma).
\]

2. \( \phi \equiv \omega \), where \( \omega \in \text{VAR}_\tau \).

2.1. \( \omega \not\equiv \nu \)

\[
V_{[\nu \to A]}(\omega) = V_{\text{as}}(\omega) = V_{\text{as}}([\alpha/\nu] \omega).
\]

2.2. \( \omega \equiv \nu \)

\[
V_{[\nu \to A]}(\nu) = A = V_{\text{as}}(\alpha) = V_{\text{as}}([\alpha/\nu] \nu).
\]

3. \( \phi \equiv [\psi = \eta] \)

\[
V_{[\nu \to A]}([\phi = \eta]) = 1 \text{ iff } V_{[\nu \to A]}(\phi) = V_{[\nu \to A]}(\eta).
\]

By induction hypothesis, this is true iff

\[
V_{\text{as}}([\alpha/\nu] \phi) = V_{\text{as}}([\alpha/\nu] \eta),
\]

hence iff \( V_{\text{as}}([\alpha/\nu][\phi = \eta]) = 1 \).
4. \( \phi \equiv \lambda \psi \)

4.1. \( w \equiv \nu \)

\[
V_{[\nu \rightarrow A] as}(\lambda \psi) = V_{as}(\lambda \nu \psi) = V_{as}[\alpha/\nu][\lambda \nu \psi].
\]

4.2. \( w \neq \nu \).

The conditions of the theorem guarantee that \( w \) does not occur in \( a \). This fact is used in equality I below. Equality II holds since we may apply the induction hypothesis for assignment \([w \rightarrow d] as\), and equality III follows from the definition of substitution.

\[
V_{[\nu \rightarrow A] as}(\lambda \psi) = V_{[\nu \rightarrow V_{as}(\alpha)] as}(\lambda \psi) =
\]

\[
= \lambda d V_{[w \rightarrow d]}(\nu \rightarrow V_{as}(\alpha)] as)(\psi) =
\]

\[
= \lambda d V_{[w \rightarrow d] as}(\alpha)] as)(\psi) =
\]

\[
= V_{as}[\alpha/\nu] \lambda \psi.
\]

The proof for \( \forall \psi \) and \( \exists \psi \) proceeds analogously.

5. \( \phi \equiv \neg \psi \)

\[
V_{[\nu \rightarrow A] as}(\neg \psi) = 1 \text{ iff } V_{[\nu \rightarrow A] as}(\psi) = 0
\]

by induction hypothesis we have

\[
V_{[\nu \rightarrow A] as}(\psi) = V_{as}[\alpha/\nu] \psi.
\]

So \( V_{[\nu \rightarrow A] as}(\neg \psi) = 1 \) iff \( V_{as}[\alpha/\nu] \neg \psi = 1 \).

Analogously for the other connectives.
From theorem 4.2 it follows that in case \( \lambda \)-conversion is allowed on a certain formula, it is allowed in whatever context the formula occurs. So, given a compound formula with several \( \lambda \)-operators, one may first reduce the operators with the smallest scope and so further, but one may reduce also first the operator with the widest scope, or one may proceed in any other sequence. Does this have consequences for the final result? In other words, is there a unique \( \lambda \)-reduced form ('a \( \lambda \)-normal form')? The answer is affirmative. The only reason which prevents a correct application of the \( \lambda \)-conversion is the syntactic constraint that a free variable in \( \alpha \) should not become bound by substitution in \( \phi \). Using \( \alpha \)-conversion (renaming of bound variables), this obstruction can be eliminated. It can then be shown that each formula in \( \mathbb{Ty}_2 \) can be reduced by use of \( \alpha \)- and \( \lambda \)-conversion to a \( \lambda \)-reduced form which is unique, up to the naming of bound variables (see the proof for typed \( \lambda \)-calculus in ANDREWS 1971 of PIETRZYSKOWSKI 1973, which proof can be applied to \( \mathbb{Ty}_2 \) as well). This property of reduction system is known under the name 'Church-Rosser property'.

The theorem we proved concerning \( \lambda \)-conversion gives a syntactic description of situations in which \( \lambda \)-conversion is allowed. It is, however, possible that the condition mentioned in the theorem is not satisfied, but that nevertheless \( \lambda \)-conversion leads to an equivalent formula. A semantic description of situations in which \( \lambda \)-conversion is allowed, is given in the theorem below. This semantic description is not useful for simplifying \( \mathbb{Ty}_2 \) formulas, since there are no syntactic properties which correspond with the semantic description in the theorem. In applications for the semantics of natural languages or programming languages we will have additional information (for instance from meaning postulates) which makes it possible to apply this theorem on the basis of syntactic criteria.

4.5. **Theorem (Janssen 1980).** Let \( \lambda v[\phi](\alpha) \in \mathbb{N} \), and suppose that for all \( as, as' \in As \): \( V_{as}(\alpha) = V_{as'}(\alpha) \).

Then for all \( as \in As \): \( V_{as}(\lambda v[\phi](\alpha)) = V_{as}[(\alpha/v)\phi] \).

**Proof.** Consider the proof of theorem 4.4. The only case where is made use of the fact that no variable in \( \alpha \) becomes bound, is in the equality 1 in case 4. Since the condition for the present theorem requires that the interpretation of \( \alpha \) does not depend on the choice of as, we have

\[
V_{as}(\alpha) = V_{[\alpha \rightarrow d]as}(\alpha).
\]
Consequently the proof of 4.4 applies, using this justification for equality I.

4.5. END

Ty2 contains typed \( \lambda \)-calculus as a sub-theory. Since typed \( \lambda \)-calculus
is not axiomatizable, Ty2 is not axiomatizable either. That typed \( \lambda \)-calculus
is not axiomatizable becomes evident if one realizes that its models are
very rich: they contain models for the natural numbers. The formal proof of
the non-axiomatizability is based upon standard techniques and seems well
known. GALLIN (1975) does not give a reference when remarking that typed
\( \lambda \)-calculus is not axiomatizable, and HENKIN (1950) only gives some hints
concerning a proof. A sketch of a possible proof is as follows. An effective
translation of Peano arithmetic into typed \( \lambda \)-calculus is defined (see below
for an example). Then it is proven that every formula \( \phi \) from Peano arith-
metic is true in the standard model of natural numbers iff the translation
of \( \phi \) is true in all standard models for Ty2. Since arithmetic truth is not
axiomatizable, Ty2 cannot be axiomatizable either.

An example of an effective translation of Peano arithmetic into typed
\( \lambda \)-calculus is given in CHURCH (1940). For curiosity I mention the transla-
tions of some numbers and of the successor operator \( S \). Also the Peano-axioms
can be formulated in typed \( \lambda \)-calculus. The formulas translating 0,1,2 and \( S \),
contain the variables \( x \in \text{VAR}_e, f \in \text{VAR}_{<e,e>,e}, \) and \( v \in \text{VAR}_{<e,e>,e,e,e} \).
One easily checks that \( S(0) = 1 \) and \( S(1) = 2 \).

<table>
<thead>
<tr>
<th>arithmetics</th>
<th>translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \lambda f x[f(x)] )</td>
</tr>
<tr>
<td>1</td>
<td>( \lambda f x[f(x)] )</td>
</tr>
<tr>
<td>2</td>
<td>( \lambda f x[f(f(x))] )</td>
</tr>
<tr>
<td>S</td>
<td>( \lambda v f x[f(v(f(x))) )</td>
</tr>
</tbody>
</table>

As I already said in section 1, Ty2 is generalized complete: i.e. the
class of formulas valid in all generalized models is axiomatizable. The
proof is a simple generalization of the proof for one-sorted type theory
(HENKIN 1950). I will define below the generalized models and present the
axioms without further proof.

The generalized domains of type \( \tau \in \text{Ty} \), denoted \( \text{GD}_\tau \), are defined by
1. \( G_D = A, \ G_D = I, \ G_D = \{0,1\} \)
2. \( G_D^{<\sigma, \tau>} \subseteq G_D^{\tau} \) (\( \sigma, \tau \in Ty \)).

The generalized meanings of type \( \tau \) denoted \( GM_\tau \), are defined by

\[ GM_\tau = G_D^{AS} \quad \text{where} \quad AS \quad \text{is the set of variable assignments.} \]

A generalized model is a pair \(<GM,F>\), where

1. \( GM = U_\tau G_D \)
2. \( F \in F \) where \( F \) is the collection of interpretations of constants
3. the pair \(<GM,F>\) is such that there exists a function \( V \) which assigns to each formula a meaning, and which satisfies the clauses \( a, b, l, \ldots, 6 \) from the definition of \( V \) for \( Ty2 \) in section 3.

Without the last requirement concerning generalized models, there might arise difficulties to define \( V \) for some model: the interpretation of \( \lambda \) might fail because the required function may fail to belong to the model. The addition of requirement 3 makes that such a situation cannot arise. On the other hand, the third condition makes that it is not evident that generalized models exist since the given definition is not an inductive definition. It can be shown, however, that out of any consistent set of formulas such a model can be built.

The axioms for \( Ty2 \) are as follows (GALLIN 1975, p.60):

A1) \( g(T) \land g(F) = \forall x[g(x)] \quad x \in \text{VAR}_\tau, \ g \in \text{VAR}_{<\tau, \tau>} \)

A2) \( x = y \rightarrow f(x) = f(y) \quad x \in \text{VAR}_\tau, \ f \in \text{VAR}_{<\tau, \tau>} \)

A3) \( \forall x[f(x) = g(x)] = [f = g] \quad x \in \text{VAR}_\sigma, \ f, g \in \text{VAR}_{<\sigma, \tau>} \)

A3') \( \lambda x[A(x)](B) = [B/x](A(x)) \quad \text{where the condition of th.4.4 is satisfied.} \)

Furthermore, there is the following rule of inference:

From \( A = A' \) and the formula \( B \) one may infer formula \( B' \), where \( B' \) comes from \( B \) by replacing one occurrence of \( A \) which is part of \( B \), by the formula \( A' \) (cf. theorem 4.2).

5. INTENSIONAL LOGIC

The language of intensional logic is for the most part the same as the language \( Ty2 \). The collection of types of \( IL \) is a subset of the collection types of \( Ty2 \). The set \( T \) of types of \( IL \) (sorts of \( IL \)) is defined as the
smallest set such that
1. \( e \in T \) and \( t \in T \)
2. if \( \sigma, \tau \in T \) then \(<\sigma, \tau> \in T \)
3. if \( \tau \in T \) then \(<s, \tau> \in T \).

The language IL is defined by the following algebra.

\[
\text{IL} = \langle \text{CON}_T \cup \text{VAR}_T \rangle_{T \in T}, R \cup \{ \text{R}_v, \text{R}_^\wedge, \text{R}_\square, \text{R}_W, \text{R}_H \}, \text{t} >
\]

where

a. \( \text{CON}_T \) and \( \text{VAR}_T \) are the same as for Ty2 (as far as the types involved belong to T)

b. \( R \) consists of all the operations of Ty2 (as far as the types involved belong to T).

The new operators are as follows

1. \( \text{R}_v: \text{ME}_{<s, \tau>} \rightarrow \text{ME}_t \) defined by \( \text{E}_v(a) = [v^a] \)
2. \( \text{R}_^\wedge: \text{ME}_t \rightarrow \text{ME}_{<s, \tau>} \) defined by \( \text{E}_^\wedge(a) = [^\wedge a] \)
3. \( \text{R}_\square: \text{ME}_t \rightarrow \text{ME}_t \) defined by \( \text{E}_\square(\phi) = [\square \phi] \)
4. \( \text{R}_W: \text{ME}_t \rightarrow \text{ME}_t \) defined by \( \text{E}_W(\phi) = [\forall \phi] \)
5. \( \text{R}_H: \text{ME}_t \rightarrow \text{ME}_t \) defined by \( \text{E}_H(\phi) = [H \phi] \).

The symbol 'v' is pronounced as 'extension' or 'down', '\(^\wedge\)' as 'intension' or 'up', \( \square \) as 'necessarily', and \( W \) and \( H \) are the future tense operator (\( W \sim \text{'will'} \)), and the past tense operator respectively (\( H \sim \text{'has'} \)). This use of the symbols \( H \) and \( W \) follows Montague's PTQ. It should be observed that his notation conflicts with the tradition in tense logic, where \( P(\text{'past'}) \) and \( F(\text{'future'}) \) are used for this purpose. The operators \( W \) and \( H \) are used in tense logic for respectively 'it always will be the case' and 'it has always be the case'.

The semantics of IL will be defined indirectly: by means of a translation of IL into Ty2<. I will employ the techniques developed in chapter 2 and design a Montague grammar for IL. This means that a homomorphism \( \text{Tr} \) will be defined from the term algebra \( T_{\text{IL}} \) associated with IL, to an algebra \( \text{Der}(\text{Ty2}) \) which is derived from Ty2< (see figure 1). The meaning of an IL expression \( \phi \) is then defined as its image under the composition of this translation \( \text{Tr} \) and the interpretation homomorphism \( V \) for Ty2< (where \( V \) is restricted to the derived algebra). This composition \( \text{Tr} \circ V \) will be denoted
V as well, since from the context it will be clear whether the interpretation of an IL expression or of a Ty2 expression is meant.

\[
\begin{array}{c}
\text{Tr} \\
\downarrow \\
\text{Der(Ty2<)} \\
\downarrow \\
\text{V} \\
\downarrow \\
\text{Der(M)}
\end{array}
\]

Figure 1. IL as a Montague grammar

The translation Tr will introduce the variables \( v_{1,s} \) and \( v_{2,s} \), which will play a special role in the interpretation of IL. Following GROENENDIJK & STOKHOF 1981, these variables will be written \( a \) and \( a' \) respectively. The translation Tr introduces only variable \( a \) as a free variable of type \( s \). Since the expressions of IL contain no variables of type \( s \), this means that the interpretation of an IL-expression is determined by the interpretation of the IL-variables and the interpretation of \( a \). Hence the meaning of an IL-expression can be considered as a function with domain the assignments to pairs consisting of the value of \( a \), and an assignment to IL-variables.

Let the collection \( G \) of assignments to IL variables be defined by

\[
\text{VAR} \\
G = \prod_{i \in I} D^g_t.
\]

The meaning of an IL-formula \( \phi \in ME_t \) is then an element of \( D^{I \times G} \), where the component \( i \) determines the value assigned to \( a \). The interpretation of \( a \) with respect to \( i \in I \) and \( g \in G \) is denoted by \( v_{i,g}^t(a) \).

From chapter 2, I recall a special method for the description of derived algebras. Let the original algebra be \( A = \langle (A_s)_{s \in S}, (F^P_v)_{v \in I} \rangle \). Then a derived algebra is uniquely determined by the following information (see chapter 2, theorem 8.13).

1) A collection \( S' \) of sorts of the derived algebra and a mapping \( \tau : S' \rightarrow S \) which associates new sorts with old sorts.

2) A collection \( (H_{s'})_{s' \in S'} \) of generators for the new algebra, such that \( H_{s'} \subset A_{\tau(s')} \).

3) A collection \( P \subset \text{POL}^A \) of operators of the new algebra, and a typegiving
function for these operators which has to satisfy certain requirements.

The interpretation of IL will be defined by means of an algebra which is derived from Ty2 in the just mentioned way. The specification of the three components is as follows.
1) The set $T$ of types of IL is a subset of set of types of $Ty2^\prec$. Hence the set of types of the derived algebra is $T$. For the mapping $\tau$ we take the identity mapping.
2) There are two kinds of generators in the derived algebra: those which correspond with variables of IL, and those which correspond with the constants of IL. As generators corresponding with the IL-variables, the same $Ty2$-variables are taken. For the constants we do not proceed in this way. Would we have done so, then a constant of IL would always been associated with one and the same semantical object, because the $Ty2$-constants have this property. For applications this is not desirable. For instance, the constant walk will be interpreted (at a given index) as a function from entities to truth values, thus determining the set of entities walking on that index. We desire, however, that for another index this set may be different. Therefore it is not attractive to take constants of $Ty2^\prec$ as generators of the derived algebra. We will use the variable $a \in \text{VAR}_S$ for indicating the current index. The generator corresponding to the IL-constant $c^a_{n, \sigma}$ is the compound formula $c^a_{n, \sigma} \circ (a)$. Note that this formula contains the constant with the same index, but of one intension level higher. These considerations explain the following definition

$$H_{\tau} = \text{VAR}_{\tau} \cup \{c^a_{n, \sigma} \circ (a) \mid n \in \mathbb{N}\}.$$ 

3) Note first that the type giving function for the polynomials does not need to be specified because all types of IL are types of $Ty2$. There are two kinds of operators. Some operators of IL are also operators of $Ty2^\prec$ as well. The polynomial symbols for these operators (see chapter 2, remark after theorem 4.6) are incorporated in $P$. The polynomial symbols corresponding with the other operators of IL are as follows

$$R_{V, \tau} : X_{1, \tau}(a)$$

$$R_{\lambda, \tau} : \lambda a [X_{1, \tau}]$$

$$R_{0} : \text{Va}[X_{1, \tau}]$$
In the polynomials for \( R_\delta \) and \( R_\lambda \), a binder for \( \varepsilon \) is introduced. In most cases this does not give rise to vacuous quantification or abstraction since the variable \( X \) will often be replaced by an expression containing a free variable \( a \) introduced by the translation of some constant. The polynomial for \( R_\delta \) might be read as \( \exists a' \exists [a'[a/X_1,t]](a') \) and for \( R_\lambda \) analogously (but these expressions are not polynomial symbols).

The information given above completely determines a unique derived algebra. Theorems of chapter 2 guarantee that in this indirect way the interpretation of IL is defined as the composition of the translation of IL into \( \text{Der}(T\!y_2<) \) with the interpretation of this derived algebra.

6. PROPERTIES OF IL

Below, and in the next section, some theorems concerning IL will be presented. The proofs will rely on the way in which the meaning of IL is defined: as the composition of the translation homomorphism \( \text{Tr} \) and the meaning homomorphism \( V \) for \( T\!y_2< \). Hence the interpretation \( V_{\delta,\hat{g}}(\phi) \) of an IL-expression \( \phi \) equals the interpretation \( V_{\delta,\hat{g}}(\text{Tr}(\phi)) \) of its translation in \( T\!y_2< \), where \( \delta \) is an assignment to \( T\!y_2< \)-variables such that \( \delta(a) = i \) and \( \delta(v) = g(v) \) for all IL-variables \( v \). Hence we may prove \( V_{\delta,\hat{g}}(\phi) = V_{\delta,\hat{g}}(\psi) \) by proving \( V_{\delta,\hat{g}}(\text{Tr}(\psi)) = V_{\delta,\hat{g}}(\text{Tr}(\phi)) \) for such a \( \delta \)-assignment \( \delta \). If \( \eta \) is an expression of \( T\!y_2 \), then the notation \( V_{\delta,\hat{g}}(\eta) \) will be used for \( V_{\delta,\hat{g}}(\eta) \), where \( \delta \) is an arbitrary assignment to \( T\!y_2 \) variables such that \( \delta(a) = i \) and \( \delta(v) = g(v) \) for all IL-variables \( v \). If \( \delta \) is of type \( t \), we will often write \( [i,g] \models \phi \) instead of \( V_{\delta,\hat{g}}(\phi) = 1 \). When \( i \) or \( g \) are arbitrary, they will be omitted. Hence \( \models \phi \) means that for all \( i \) and \( g \) it is the case that \([i,g] \models \phi \).

In section 4 we notified the theoretical importance of \( \lambda \)-conversion for \( T\!y_2 \): all quantifiers and connectives can be defined by means of lambda operators. For \( T\!y_2 \) the same holds, so it is of theoretical importance to know under which circumstances \( \lambda \)-conversion is allowed. But there also is an important practical motivation. We will frequently use \( \lambda \)-conversion for simplifying formulas. For these reasons I will consider the IL-variants of the theorems concerning the substitution of equivalents and concerning \( \lambda \)-conversion.
6.1. **THEOREM.** Let \( \alpha, \alpha' \in \mathfrak{M}_g \) and \( \beta, \beta' \in \mathfrak{M}_t \) such that

a) \( \beta \) is part of \( \alpha \)

b) \( \beta' \) is part of \( \alpha' \)

c) \( \alpha' \) is obtained from \( \alpha \) by substitution of \( \beta \) for \( \beta' \). Suppose that for all \( i \in I \) and \( g \in G \)

\[
V_{i,g}(\beta) = V_{i,g}(\beta').
\]

Then for all \( i \in I \) and \( g \in G \)

\[
V_{i,g}(\alpha) = V_{i,g}(\alpha').
\]

**PROOF.** This theorem could be proven in the same way as the corresponding theorem for \( Ty_2 \) by reference to theorem 6.4 from chapter 2. I prefer, however, to prove the theorem by means of translation into \( Ty_2 \) because this shows some arguments which will be used (implicitly or explicitly) in the other proofs.

From (1) we may conclude that (2) holds, from which (3) immediately follows:

1. For all \( i \in I, g \in G \): \( V_{i,g}(\beta) = V_{i,g}(\beta') \)
2. For all \( i \in I, g \in G \): \( V_{i,g}(\text{Tr}(\beta)) = V_{i,g}(\text{Tr}(\beta')) \)
3. For all \( \alpha \in \mathfrak{A}_s \): \( V_{\alpha}(\text{Tr}(\beta)) = V_{\alpha}(\text{Tr}(\beta')) \).

Recall that \( \beta \) is a part of \( \alpha \) if there is in the production of \( \alpha \) an application \( R(\ldots, \beta, \ldots) \) of an operator \( R \) with \( \beta \) as one of its arguments. Since \( \text{Tr} \) is a homomorphism defined on such production processes, it follows that

\[
\text{Tr}(\alpha) = \text{Tr}(\ldots, R(\ldots, \beta, \ldots), \ldots) = \ldots R'(\ldots, \text{Tr}(\beta), \ldots) \ldots
\]

(here is \( R' \) the polynomial operator over \( Ty_2 \) which corresponds with the IL-operator \( R \)). This says that the translation of a part of \( \alpha \) is a part of the translation of \( \alpha \). Consequently we may apply to (3) theorem 4.2 and conclude that (4) holds.

4. For all \( \alpha \in \mathfrak{A}_s \): \( V_{\alpha}(\text{Tr}(\alpha)) = V_{\alpha}(\text{Tr}(\alpha')) \).

From this follows

5. For all \( i \in I, g \in G \): \( V_{i,g}(\alpha) = V_{i,g}(\alpha') \).

6.1. END

An important class of expressions are the expressions which contain neither constants, nor the operators \( \land, \lor \) or \( \forall \). Such expressions are called modally closed; a formal definition is as follows.
6.2. DEFINITION. An expression of IL is called **modally closed** if it is an element of the subalgebra

\[
\langle \{ \text{VAR} \}_{\tau \in Ty}, R \cup \{ R^\wedge, R^\vee \} \rangle
\]

where \( R \) consists of the operators of Ty2.

6.2. END

The theorem for \( \lambda \)-conversion which corresponds with theorem 4.4 reads as follows

6.3. **THEOREM.** Let \( \lambda v[\phi](a) \in ME_\tau \), and suppose that no free variable in \( a \) becomes bound by substitution of \( a \) for \( v \) in \( \phi \). Suppose that one of the following two conditions holds:

1. no occurrence of \( v \) in \( \phi \) lies within the scope of \( \wedge, \vee, \top, \) or \( \Box \)
2. \( a \) is modally closed.

Then for all \( i \in I \) and \( g \in G \)

\[ i, g \vdash \lambda v[\phi](a) = [a/v]\phi. \]

**PROOF.** Part 1.

Suppose condition 1 is satisfied.

The translation \( Tr(a) \) of \( a \) contains the same variables as \( a \), except for the possible introduction of variables of type \( s \). The translation \( Tr(\phi) \) of \( \phi \) contains the same binders as \( \phi \) since only \( \wedge, \vee, \top, \) and \( \Box \) introduce new binders (see the definition of \( Tr \)). Since \( \phi \) itself does not contain binders for variables of type \( s \), we conclude that:

No free variable in \( Tr(\phi) \) becomes bound by substitution of \( Tr(a) \) for \( v \) in \( Tr(\phi) \).

Theorem 6.1 allows us to conclude from this that for all \( \in AS \),

\[ V_\alpha (\lambda v[Tr(\phi)](Tr(a))) = V_\alpha ([Tr(a)/v]Tr(\phi)). \]

Note that \( Tr(\phi) \) has the same occurrences of \( v \) as \( \phi \), hence one easily proves with induction that

\[ [Tr(a)/v]Tr(\phi) = Tr([a/v]\phi). \]

Consequently \( V_\alpha (Tr(\lambda v[\phi](a))) = V_\alpha (Tr([a/v]\phi)). \)
So \( \text{Tr} \circ V(\lambda v[\phi](a)) = \text{Tr} \circ V([a/v]\phi) \).

From this it follows that for all \( g \in G \) and \( i \in I \)

\[ g, i \vdash \lambda v[\phi](a) = [a/v]\phi. \]

Part 2.

Suppose that condition 2 is satisfied.

The translation of \( \phi \) may introduce binders for variables of type \( s \), but it does not introduce binders for variables of other types (see the definition of \( \text{Tr} \)). The expression \( a \) does not contain free variables of type \( s \), and the translation in this case does not introduce such variables since the only kind of expressions which give rise to new free variables are constants, and the operators \( \vee, \wedge, \) and \( \oplus \). So we may conclude that:

No free variable in \( \text{Tr}(a) \) becomes bound by substitution of \( \text{Tr}(v) \) for \( v \) in \( \text{Tr}(\phi) \).

From this we can prove the theorem in the same way as we did for the first condition.

6.3. END

In theorem 4.5 a semantic description was given of situations in which \( \lambda \)-conversion is allowed. The IL variant of this theorem reads as follows.

6.4. THEOREM (IL). Let \( \lambda v[\phi](a) \in M_2 \) and suppose for all \( i,j \in I \) and \( g,h \in G \):

\[ V_{i,g}(a) = V_{j,h}(a). \]

Then for all \( i \in I \) and \( g \in G \):

\[ V_{i,g}(\lambda v[\phi](a)) = V_{i,g}([a/v]\phi). \]

PROOF. By translation into Ty2 and application of theorem 4.5.

6.4. END

In the light of the role of \( \lambda \)-conversion, it is interesting to know whether \( \lambda \)-conversion in IL has the Church-Rosser property, i.e. whether there is an unique lambda-reduced normal form for IL. In much practical experience with intensional logic I learned that it does not matter in which order a formula containing several \( \lambda \)-operators is simplified: first applying \( \lambda \)-reduction to the most embedded operators, or first the most outside ones, the final result was the same. It was a big surprise that FRIEDMAN & WARREN (1980b) found an IL-expression where different reduction sequences yield different final results. Their example is
\[ \lambda x(\lambda y[x^\beta y = u(x)])(c) \]

where \( x \) and \( y \) are variables of some type \( \tau \), \( c \) a constant of type \( \tau \), and \( u \) a variable of type \( \langle \tau, c, \tau \rangle \). For each of the \( \lambda \) operators the conditions for the theorem are satisfied. Reducing first \( \lambda x \) yields

\[ \lambda y[x^\beta y = u(c)](c) \]

which cannot be reduced further since the conditions for \( \lambda \)-conversion are not satisfied. Reducing first \( \lambda y \) yields

\[ \lambda x[x^\beta x = u(x)](c) \]

which cannot be reduced either. We end up with two different, although logically equivalent, formulas; i.e. there is no \( \lambda \)-reduced normal form for \( \Pi \).

The example depends on the particular form for \( \lambda \)-contraction: for all occurrences of the variable the substitution takes place in one and the same step. FRIEDMAN & WARREN (1980) is equivalent to

\[ [\lambda x[x^\beta x]](c) = u(c). \]

This formula is in some sense further reduced. They conjecture that for a certain reformulation of \( \lambda \)-conversion the Church-Rosser property could be provable.

It is interesting to compare the above discussion with the situation in \( Ty_2 \), where there is a unique \( \lambda \)-reduced form. The \( Ty_2 \)-translation of the Friedman-Warren formula is \( (c' \in CON_{\langle s, \tau \rangle} !) \)

\[ \lambda x[\lambda y[\lambda a[y] = u(x)]](x) ]c'(a). \]

This reduces to

\[ \lambda y[\lambda a[y] = u(c'(a)) ]c'(a). \]

After renaming the bound variable \( a \) to \( i \), this reduces further to
\( \lambda y[\lambda z[c'(a)] = u(c'(a))]. \)

Note that this last reduction is possible here (and not in IL) because of the explicit abstraction \( \lambda i \), instead of the implicit abstraction in \( \land y \).

A lot of laws of IL are variants of well-known laws for predicate logic and type logic. An exception to this description is formed by alws involving constants. The constants of IL are interpreted in a remarkable way: their interpretation is state dependent. Invalid is, for instance, the existential generalization \( \Box \forall \alpha(a) \rightarrow \exists \Box \forall \alpha(x) \), whereas \( \forall y[\Box \forall \alpha(y) \rightarrow \exists \Box \forall \alpha(x)] \) is valid. Invalid is \( \forall x \in c \rightarrow \Box [x = c] \), whereas \( \forall x \forall y[x = y \rightarrow \Box [x = y]] \) is valid. Other examples of invalid formulas are \( \exists y \Box [y = c] \) and \( \forall x[\Box A(x) \rightarrow \Box \alpha(a)] \), where \( \Box \) abbreviates \( \land \Box \land \).

Since IL contains type theory as a sublanguage, there is no axiomatization of IL (see also section 4). But IL is generalized complete as is proved by GALLIN (1975). The proof is obtained by combining the proof of generalized completeness of type theory (HENKIN 1950), and the completeness proof for modal logic (see HUGHES & CRESSWELL 1968). The following axioms for IL are due to GALLIN (1975); the formulation is adapted

\[
\begin{align*}
A1 & \quad [g(y) \land g(x)] = \forall x [g(x)] & x \in \text{VAR}_\xi, g \in \text{VAR}_{\xi,t} \\
A2 & \quad x = y \rightarrow f(x) = f(y) & x \in \text{VAR}_\sigma, f \in \text{VAR}_{\sigma,t} \\
A3 & \quad \forall x[f(x) = g(x)] = [f = g] & x \in \text{VAR}_\sigma, f,g \in \text{VAR}_{\sigma,t} \\
A4 & \quad \lambda x[\alpha](\beta) = [\beta/x] \alpha & \text{if the conditions of theorem 5.2. are satisfied} \\
A5 & \quad \Box[\forall \gamma \neq \gamma] = [f = g] & f,g \in \text{VAR}_{\xi,\tau} \\
A6 & \quad \forall \alpha \rightarrow \alpha = \alpha & \alpha \in \text{ME}_\xi .
\end{align*}
\]

The rule of inference is:

From \( A = A' \) and the formula \( B \) one may infer to formula \( B' \), where \( B' \) comes from \( B \) by replacing one occurrence of \( A \), that is part of \( B \), by \( A' \).

Notice that the translation of axiom \( A5 \) into \( \text{Ty}_2 \), would lead to a formula of the form of \( A3 \), and that the translation of \( A6 \) into \( \text{Ty}_2 \) would be of the form of \( A4 \). Axiom \( A6 \) will be considered in detail in the next section.

I will not consider details of this axiomatization for the following three reasons.
1) This axiomatization was designed for constituting a basis for the completeness proof, and not for proving theorems in practice. To prove the most simple theorems on the basis of the above axioms would be rather difficult. All proofs that will be given in the sequel are semantic proofs and not syntactic proofs: i.e. the proofs will be based upon the interpretation of formulas and not on axioms.

2) We will work with models which obey certain postulates. These postulates express many important semantic details, and most of the proofs we are interested in, are based upon these special properties and not on the general properties described by the axioms.

3) We do not work with generalized models, but with standard models. So the axiomatization is not complete in this respect.

7. EXTENSION AND INTENSION

In this section special attention is paid to the interaction of the extension operator and the intension operator. In this way some insight is obtained in these operators and their sometimes remarkable properties. The 'Bigboss' example, which will be given below, is important since the Bigboss will figure as (counter)example on several occasions.

7.1. THEOREM. For all i, g: \( V_{i,g}^{\land \beta} = V_{i,g}^{\beta} \).

PROOF. \( \text{Tr}^{\land \beta}(\alpha) = \text{Tr}^{\beta}(\alpha) = \lambda a[\text{Tr}(\alpha)](a) = [\alpha/a]\text{Tr}(a) = \text{Tr}(\alpha) \).

Note that \( \lambda \)-conversion is allowed because the condition of theorem 4.5 is satisfied.

7.1. END

It was widely believed that the extension operator should be the right inverse of the intension operator as well. This belief is expressed in PARTEE (1975, p.250) and in GEBAUER (1978, p.47). It is true, however, only in certain cases. In order to clarify the situation, consider the following description of the effect of \( \land \beta \). Let I and D be denumerable, so 
\( D_\beta = \{d_1, d_2, \ldots\} \) and \( I = \{i_1, i_2, \ldots\} \). Let \( \alpha \in \mathcal{M}_\beta \), hence the meaning of \( \alpha \) is a function with domain I and range \( D_\beta \). We may represent \( \alpha \) as a denumerable sequence of elements from \( D_\beta \). An example is given in figure 2.
The interpretation for index i of $\Diamond \alpha$ is some function with domain I and range $D_\alpha$. Which function it is does not depend on the choice of $i$, because $\text{Tr}(\Diamond \alpha)$ which equals $\lambda a[\text{Tr}(\alpha)(a)]$, contains no free variables of type $\tau$.

The function $V_{i_1, g}(\Diamond \alpha)$ yields for argument $i_n$ as value the value of $V_{i_n, g}(\alpha)$ for argument $i_n$. So in the above example for argument $i_1$ it yields as value $d_1$, for $i_2$ it yields $d_2$, and for $i_3$ it yields $d_3$ (the underlined elements). One observes that $\Diamond \alpha$ is the diagonalization of $\alpha$. So $\Diamond \alpha = \alpha$ will hold for all $i, g$ if for all $i, j \in I$: $V_{i, g}(\alpha) = V_{j, g}(\alpha)$. A syntactic description of a class of formulas for which the equality holds, is given in the following theorem.

7.2. THEOREM. Suppose $\alpha$ is modally closed. Then for all $i, g$: $V_{i, g}(\Diamond \alpha) = V_{i, g}(\alpha)$.

Proof. The functions $\text{Tr}(\Diamond \alpha)$ and $\text{Tr}(\alpha)$ denote functions with domain I, and their values for an arbitrary argument $i$ are the same:

$$\text{Tr}(\Diamond \alpha)(i) = \lambda a[\text{Tr}(\alpha)(a)](i) = [i/a]\text{Tr}(\alpha)([i/a]a) = \text{Tr}(\alpha)(i).$$

Notice that $[i/a]\text{Tr}(\alpha) = \text{Tr}(\alpha)$ since $\alpha$ is modally closed. So for all $\alpha$ as $\Diamond \alpha$, $\text{Tr}(\Diamond \alpha) = \text{Tr}(\alpha)$, which proves the theorem.

7.2. END

The insights obtained from the 'diagonalization' point of view, can be used to obtain a counterexample for the case that $\alpha$ is not modally closed. It suffices to find an expression $\alpha$ of type $\tau$ which has at index $i_1$ as its denotation a constant function from I to $D_\alpha$, say with constantly value $d_1$, and at index $i_2$ as its denotation a constant function yielding some other value, say $d_2$. This situation is represented in figure 3.
Now $V_{1,8}^{(\land \forall)}(\alpha)(i_1) = d_1 \neq d_2 = V_{2,8}^{(\land \forall)}(\alpha)(i_1)$.

One way to obtain this effect is by means of a constant. I give an example of a somewhat artificial nature (due to JANSSEN 1980). Let the valuation of the constant Bigboss ∈ CON_{<s,e>} for index $i$ be the function constantly yielding the object $d \in D_e$ to which the predicate 'is the most powerful man on earth' applies on that index. A possible variant of this example would be the constant Miss-world ∈ CON_{<s,e>}, to which the predicate applies 'is elected as most beautiful woman in the world'.

Assume that for the constant Bigboss holds that for all $j \in I$ both

$$V_{1,8}^{[\text{Bigboss}]}(j) = V_{1,8}^{[\text{Reagan}]}$$

and

$$V_{2,8}^{[\text{Bigboss}]}(j) = V_{2,8}^{[\text{Bresnjev}]}.$$

Then

$$V_{1,8}^{(\land \forall \text{ Bigboss})}(i_2) = \lambda i[V_{1,8}^{[\text{Bigboss}]}(i)](i_2) =$$

$$= V_{2,8}^{[\text{Bigboss}]}(i_2) = V_{2,8}^{[\text{Bresnjev}]}.$$

So

$$V_{1,8}^{(\land \forall \text{ Bigboss})}(i_2) \neq V_{1,8}^{[\text{Bigboss}]}(i_2).$$

This effect does not depend on the special interpretations for constants.

Another way to obtain the desired effect is by taking for $a$ the expression $x$ where $x$ is a variable of type $<s,<s,e>$. Let $g(x)$ be defined such that for all $j \in I$: $g(x)(j) = V_{1,8}^{[\text{Bigboss}]}$. Then $\forall x \neq [x]$: because $V_{1,8}^{(\forall x)}(i_2) = \lambda i[g(x)(i)](i_2) = V_{1,8}^{[\text{Bigboss}]}(i_2) =$

$$= V_{2,8}^{[\text{Bresnjev}]}$$

whereas $V_{1,8}^{[x]}(i_2) = g(x)(i_2) = V_{1,8}^{[\text{Bigboss}]}(i_2) = V_{1,8}^{[\text{Reagan}]}$. 

\begin{tabular}{|c|c|c|c|c|}
\hline
arguments & $i_1$ & $i_2$ & $i_3$ & $i_4$ \\
\hline
$V_{1,8}^{(\land \forall)}(\alpha)$: values for the respective arguments & $d_1$ & $d_1$ & $d_1$ & $d_1$ \\
$V_{2,8}^{(\land \forall)}(\alpha)$: values for the respective arguments & $d_2$ & $d_2$ & $d_2$ & $d_2$ \\
\hline
\end{tabular}

Figure 3. A counterexample for $\land \forall \alpha$. 


The next example concerns the situation that IL is extended with the if-then-else construct. Let $\beta$ be of type $\tau$ and $\phi$ and $\psi$ of type $\tau$, and define

$$V_{i, g}[if \beta \ then \ \phi \ else \ \psi] = \begin{cases} V_{i, g}(\phi) & \text{if } V_{i, g}(\beta) = 1 \\ V_{i, g}(\psi) & \text{otherwise.} \end{cases}$$

Let $x$ and $y$ be variables of type $<s, e>$ and assume that $g(x) \neq g(y)$, but that for some $i$ holds that $g(x)(i) = g(y)(i)$. Then it is not true that for all $i, g$

$$i, g \models \forall [if \ x = y \ then \ x \ else \ y] = [if \ x = y \ then \ x \ else \ y].$$

This kind of expression is rather likely to occur in the description of semantics of programming languages.

The last example is due to GROENENDIJK & STOKHOFF (1981). They consider the semantics of whether-complements. An example is

"John knows whether Mary walks."

The verb know is analysed as a relation between an individual and a proposition. Which proposition is John asserted to know? If it is the case that Mary walks, then John is asserted to know that Mary walks. And if Mary does not walk, then he is asserted to know that Mary does not walk. So the proposition John knows appears to be

$$if \ walk(mary) \ then \ walk(mary) \ else \ \neg walk(mary).$$

This example provides for a rather natural example of a formula $\phi$ for which $\forall \phi$ does not reduce.