position clitics that is at the heart of Penn's analysis is a famous instance of the same theoretical problem. Bracketing paradoxes have been an unsolved puzzle for many linguistic theories, and it is not obvious how the traditional linearization theories of HPSG could be extended naturally in order to give a satisfactory account of the empirical data. The expressive power and the flexible grammar architecture of RSRL enable Penn to provide a genuinely new solution for a long-standing, theoretical problem. Penn's work is thus an excellent example of how a mathematical formalism can support precise and innovative answers to open questions of linguistic theory.

5.4 Model-theoretic Semantics[19]

The traditional semantic structures of HPSG grammars were built on ideas that came from situation semantics (Barwise and Perry, 1983). That means that in Pollard and Sag 1994, the configurations under *content* entities that are defined by the signature and restricted by the SEMANTICS PRINCIPLE, the QUANTIFIER BINDING CONDITION and the CONTROL THEORY are inspired by situation semantics, and it is not straightforward to compare them with other, model-theoretic approaches to natural language. In an attempt to bridge the gap between HPSG and much of the current linguistic literature of model-theoretic semantics, Richter and Sailer 1999a,b introduce a completely different signature for *content* entities together with a new set of principles that restrict the admissible configurations under *content* entities. The idea is to furnish signs with CONTENT values that can be regarded as representations of a symbolic language with a model-theoretic interpretation. Richter and Sailer choose the language Ty2 of two-sorted type theory (Gallin, 1975) as the semantic object language under CONTENT of their grammars of French and Polish negative polarity items, because Ty2 is one of the standard representation languages of natural language semantics.[20]

In this section, I will sketch how a signature and a theory of Ty2 can be defined in RSRL. For that purpose, I will specify an RSRL grammar whose intended exhaustive model is the language of Ty2. In fact, a proof by

[19] The text of this section is partly based on joint work with Manfred Sailer published in the appendix to Richter and Sailer 1999a.

[20] In particular, the fragment of Ty2 that is actually used in linguistic analyses is equivalent to Richard Montague's Intensional Logic, but it has a number of technical advantages. See Zimmermann 1989 for discussion.

Sailer (2000) shows that the models of that grammar can be thought of as containing expressions of Ty2. In contrast to the theories of Sections 5.2 and 5.3, the formalization of Ty2 in RSRL does not in and by itself constitute a new analysis of some empirical linguistic phenomenon. It is a technical way of combining two formal systems that already existed independently before, a formalism for HPSG and two-sorted type theory. The integration of Ty2 with HPSG offers the opportunity to study the interaction between the techniques of model-theoretic semantics and of constraint-based grammar in rigorously formalized HPSG grammars. The existence of a mathematical formalism for HPSG grammars is a necessary prerequisite for making Ty2 available to HPSG linguists.

Before I go into the details of the specification of Ty2 in RSRL, it might be useful to illustrate the objective with a small example. The goal is to specify a signature and the principles of a grammar that defines a standard semantic representation language. This grammar can then be added to a grammar that is lacking semantic representations. With additional principles that constrain the combination of semantic and syntactic (and, potentially, other) structures, we obtain an analysis of signs whose model-theoretic meaning is determined by the semantic representations that are their CONTENT value. With the PRINCIPLE OF FULL INTERPRETATION (PFI), we will later look at an example of an interface principle between syntax and semantics. The PFI constrains the admissible semantic representations, which are often called the *logical form*, of utterances.[21]

We start with lexical specifications. What we want to be able to do in the end is to specify the semantics of the universal quantifier *every* in its lexical entry as illustrated in (137). The variable w is a world index.

(137) *The relevant part of the lexical entry of* every

$$\begin{bmatrix} \textit{word} \\ \text{PHON} \ \langle \textit{every} \rangle \\ \text{SS LOC CONT LF} \ \ \lambda P \lambda Q. \forall x[P(w)(x) \rightarrow Q(w)(x)] \end{bmatrix}$$

The value of the attribute LOGICAL-FORM (LF), which is appropriate to *content*, is the expression $\lambda P \lambda Q. \forall x[P(w)(x) \rightarrow Q(w)(x)]$. Of course, what we really have to do is to *describe* that expression, but once we will have

[21] More interface principles within the present architecture of logical forms can be found in Richter and Sailer 1999a,b, and in Sailer 2000.

explained how we can do that, we may as well write down the expression directly, which is much more readable than an AVM formula describing it. The standard interpretation of the given Ty2 expression determines the meaning of the word *every*.

Once we have lexical specifications of logical forms of the kind exemplified in (137), we can introduce a new SEMANTICS PRINCIPLE. In the new SEMANTICS PRINCIPLE, we will define functional application as the mode of combining the semantic contributions of two signs when they form a phrase. (138) illustrates the effect with the nominal phrase *every unicorn*. The semantic contributions of the quantifier *every* and of the noun *unicorn* are combined by functional application of the former to the latter. Note that, in addition, the resulting expression is fully β-reduced. To obtain that result, we will also have to define a mechanism for β-reduction in RSRL.[22]

(138) *Functional application in HPSG*

NP

$$\begin{bmatrix} \text{PHON} & \langle \textit{every, unicorn} \rangle \\ \text{SS LOC CONT LF} & \lambda Q.\forall x[\mathsf{unicorn}'(w)(x) \rightarrow Q(w)(x)] \end{bmatrix}$$

C — Det

$$\begin{bmatrix} \text{PHON} & \langle \textit{every} \rangle \\ \text{SS LOC CONT LF} & \lambda P \lambda Q.\forall x[P(w)(x) \rightarrow Q(w)(x)] \end{bmatrix}$$

H — N

$$\begin{bmatrix} \text{PHON} & \langle \textit{unicorn} \rangle \\ \text{SS LOC CONT LF} & \mathsf{unicorn}'(w) \end{bmatrix}$$

5.4.1 A Grammar of the Syntax of Ty2

To be explicit about what the RSRL grammar of Ty2 is supposed to specify, it is necessary to recall the definition of the syntax of Ty2. Assume that e, t, and s are three distinct objects, none of which is an ordered pair. Let $Types$ be the the smallest set that contains e, t, and s, and is closed under pairs. $Types$ is the set of *two-sorted types* (hence *types*). If $\tau \in Types$, then Var_τ is a denumerably infinite set of variables of type τ. I write $v_{n,\tau}$ for the $(1+n)$th variable of type τ. Similarly, if $\tau \in Types$, then $Const_\tau$ is a denumerably infinite set of constants of type τ, and I write $c_{n,\tau}$ for the $(1+n)$th constant of type τ.

[22] β-reduction is sometimes also called lambda conversion in the literature. My terminology follows the introductory textbook by Hindley and Seldin (1986).

Definition 98 *The* ***meaningful expressions of Ty2*** *are the smallest family* $(\mathrm{Ty2}_\tau)_{\tau \in Types}$ *such that*

for each $\tau \in Types$, $Var_\tau \cup Const_\tau \subset \mathrm{Ty2}_\tau$,

for each $\tau \in Types$, *for each* $\tau' \in Types$,

if $\alpha \in \mathrm{Ty2}_{\langle \tau', \tau \rangle}$ *and* $\beta \in \mathrm{Ty2}_{\tau'}$, *then* $\alpha(\beta) \in \mathrm{Ty2}_\tau$,

for each $\tau \in Types$, *for each* $\tau' \in Types$, *for each* $n \in \mathbb{N}$, *for each* $v_{n,\tau'} \in Var_{\tau'}$, *for each* $\alpha \in \mathrm{Ty2}_\tau$,

$(\lambda x.\alpha) \in \mathrm{Ty2}_{\langle \tau', \tau \rangle}$,

for each $\tau \in Types$, *for each* $\alpha \in \mathrm{Ty2}_\tau$, *for each* $\beta \in \mathrm{Ty2}_\tau$,

$(\alpha = \beta) \in \mathrm{Ty2}_\tau$,

for each $\alpha \in \mathrm{Ty2}_t$,

$\neg\alpha \in \mathrm{Ty2}_t$,

for each $\alpha \in \mathrm{Ty2}_t$, *for each* $\beta \in \mathrm{Ty2}_t$,

$(\alpha \wedge \beta) \in \mathrm{Ty2}_t$, *and* *(analogously for* $\vee, \rightarrow$, *and* $\leftrightarrow$*)*

for each $\tau \in Types$, *for each* $n \in \mathbb{N}$, *for each* $v_{n,\tau} \in Var_\tau$, *for each* $\alpha \in \mathrm{Ty2}_t$,

$\exists v_{n,\tau}\alpha \in \mathrm{Ty2}_t$. *(and analogously for* $\forall$*)*

For simplicity, the RSRL grammar of the meaningful expressions of Ty2 will describe a slight notational variant of the expressions of Definition 98. In the grammar, I give every meaningful expression a subscript that indicates its type instead of only giving type subscripts to variables and constants. For example, if α is a variable of type $\langle e, t\rangle$ and $c_{n,e}$ is a constant, the RSRL grammar will specify structures of the form $\alpha(c_{n,e})_t$ as the functional application of α to $c_{n,e}$ instead of $\alpha(c_{n,e})$, as it is done in Definition 98.

In a complete definition of Ty2, it would now be necessary to say what a Ty2 model is and how the meaningful expressions of Ty2 are interpreted in models. These definitions, however, are not needed here, because we only want to specify the syntax of Ty2 with RSRL. The semantics of the expressions will only become interesting in linguistic applications when we ask ourselves what signs mean. Since I will not discuss linguistic applications, I omit the definition of Ty2 models and the semantics of Ty2 expressions.[23]

We are now ready to define the RSRL grammar of the meaningful expressions of Ty2. My specification of the grammar of Ty2 closely follows the appendix to Richter and Sailer 1999a with the modifications proposed in Sailer 2000. The ontology of Ty2 expressions is determined by a hierarchy of sorts under the supersort *meaningful-expressions* (*me*). The sort *me* will be defined as appropriate value for the pair ⟨*content*, LOGICAL-FORM⟩, and thus be anchored in the feature geometry of signs.

(139) The sort hierarchy of the sort *meaningful-expression* (*me*):

```
me   TYPE   type
    var(iable)   N-NUM(BER)   n-num(ber)
    const(ant)   N-NUM(BER)   n-num(ber)
    appl(ication)   FUNC(TOR)     me
                    ARG(UMENT)    me
    abstr(action)   VAR   variable
                    ARG   me
    equ(ation)   ARG1   me
                 ARG2   me
    neg(ation)   ARG   me
    l(ogical)-const(ant)   AGR1   me
                           ARG2   me
        dis(junction)
        con(junction)
        imp(lication)
        bi-imp(lication)
    quant(ifiers)   VAR     variable
                    SCOPE   me
        uni(versal)
        exi(stential)
```

[23]For models and a semantics of Ty2 expressions that are directly compatible with my DEFINITION 98, see Zimmermann 1989, p. 66.

The entities in the denotation of the new species are the variables, non-logical constants, and logical operators of the expressions of Ty2. The attributes whose interpretation is defined on them bear intuitively clear names. A functional application consists of a FUNCTOR and of an ARGUMENT. Since the functor and the argument in an application are meaningful expressions, the values of both attributes are again meaningful expressions. The same holds for the two arguments, ARG1 and ARG2, of equations, and for the two arguments of the binary logical operators (*l-const*). Quantifiers and (λ-) abstraction are different. Since quantification is quantification over variables and abstraction is abstraction over variables, the entities that model them bear an attribute, VAR, whose value is a variable, i.e., an entity of sort *var*. The second attribute that is defined on quantifiers and on abstractions is the attribute ARG or SCOPE, respectively. Both attributes have meaningful expressions as values. Finally, entities of sort *negation* have one argument, which is a meaningful expression.

The attribute TYPE is defined on every entity of sort *me*. Its value encodes the type of the expression and can be read as a subscript of the expression. Entities of sort *var* and entities of sort *const* have one more subscript. The second subscript is the natural number, n, that is used to number the variables (and constants) of each type. To number the variables and constants of each type, a representation of natural numbers is needed. With existential component quantification, a theory of natural numbers can easily be defined on the basis of a simple signature:

(140) The natural numbers:

n-number

  zero

  non-zero   N-NUMBER   *n-number*

The theory of the natural numbers:

$n\text{-}number \rightarrow \; \exists x \; {}^{x}[zero]$

A natural number (*n-number*) is either zero (*zero*) or non-zero (*non-zero*). If a natural number is not zero, then it has a predecessor, which we find by interpreting the attribute N-NUMBER on it. The theory of entities of sort *n-number* says that after finitely many iterations of applying the function N-NUMBER to predecessors of a natural number, we will finally find the last predecessor, *zero*, of each natural number. 0 does not have a predecessor.

For 1, we find the last predecessor immediately. For 2, we find it after the second application of N-NUMBER, and so on.

Every meaningful expression has a subscript that encodes its type. A type is either atomic or a pair. There is a species for each atomic type, *entity* for the type e, *truth* for the type t, and *w-index* for the type s. Complex types are modeled by entities of sort *complex-type*. They have two attributes, IN and OUT, which are again *type*-valued. The entity under IN models the first type in the pair that constitutes a complex type, and the entity under OUT models the second type in the pair.

(141) The sort hierarchy of *type*:

```
type
    atomic-type
        entity
        truth
        w-index
    complex-type  IN   type
                  OUT  type
```

The ontology of meaningful expressions that is given with an interpretation of the signature of (139), (140), and (141) may contain configurations of entities that obviously do not model expressions of Ty2. For example, the first and second argument of logical constants do not need to be of type t, contrary to DEFINITION 98. A grammar of Ty2 expressions must guarantee the proper types of the Ty2 expressions in the models of our grammar of the meaningful expressions of Ty2. In (142), I state the necessary type restrictions for meaningful expressions. The order in which they are given mirrors the order of the corresponding clauses of DEFINITION 98.

(142) Type restrictions on the logical operators:

$$appl \rightarrow \begin{bmatrix} \text{TYPE} & \boxed{2} \\ \text{FUNC TYPE} & \begin{bmatrix} \text{IN} & \boxed{1} \\ \text{OUT} & \boxed{2} \end{bmatrix} \\ \text{ARG TYPE} & \boxed{1} \end{bmatrix},$$

$$abstr \rightarrow \begin{bmatrix} \text{TYPE} & \begin{bmatrix} \text{IN} & \boxed{1} \\ \text{OUT} & \boxed{2} \end{bmatrix} \\ \text{VAR TYPE} & \boxed{1} \\ \text{ARG TYPE} & \boxed{2} \end{bmatrix},$$

$$equ \rightarrow \begin{bmatrix} \text{TYPE} & truth \\ \text{ARG1 TYPE} & \boxed{1} \\ \text{ARG2 TYPE} & \boxed{1} \end{bmatrix},$$

$$neg \rightarrow \begin{bmatrix} \text{TYPE} & truth \\ \text{ARG TYPE} & truth \end{bmatrix},$$

$$l\text{-}const \rightarrow \begin{bmatrix} \text{TYPE} & truth \\ \text{ARG1 TYPE} & truth \\ \text{ARG2 TYPE} & truth \end{bmatrix},$$

$$quant \rightarrow \begin{bmatrix} \text{TYPE} & truth \\ \text{SCOPE TYPE} & truth \end{bmatrix}$$

In an entity of sort *application* in a model of the grammar, the first element in the pair that is the complex type of the functor (its IN value) must be identical to the type of the argument. And the type of the *application* entity itself is identical to the OUT value of the type of the functor. Analogously, the other descriptions in (142) restrict the types of abstraction entities, equations, and so forth.

With the signature for meaningful expressions, types, and natural numbers, and the theory of natural numbers and the type restrictions on the logical operators, the RSRL grammar of Ty2 expressions is almost complete. We only need three more restrictions. Firstly, cyclic structures must be excluded, because cyclic expressions or cyclic types are not permitted in the syntax of Ty2. If they were not explicitly excluded, our grammar of Ty2 would license them. Secondly, all expressions must be finite: Each meaningful expression entity has a finite number of components. Again, the grammar would license infinite expressions if we did not explicitly exclude them. The third condition, adopted from Sailer 2000, is not strictly necessary, but it makes the correspondence between the representations of Ty2 expressions in models of our grammar and the regular notation of Ty2 expressions more transparent. The condition requires that in each meaningful expression, isomorphic configurations of entities be identical. For example, that means that there can only be one entity of sort *truth* or of the sort *w-index* in each expression; and each occurrence of the variable $v_{n,\langle e,t\rangle}$ in an expression will be modeled by the same configuration of entities. Loosely speaking, the principle that enforces the identity of isomorphic configurations of entities prevents spurious ambiguities in the representations of Ty2 expressions, because it entails that all the configurations of entities that model a certain expression form an indiscernibility class.

In the following paragraphs, I discuss these three additional restrictions in turn. For the principles, I presuppose that the sort hierarchy of the signature of the RSRL grammar of Ty2 contains a sort, *ty2*, that is an immediate supersort of the sorts *me*, *type*, and *n-number*.

Sailer 2000 shows how to formulate a GENERAL NON-CYCLICITY PRINCIPLE (GNP) over arbitrary signatures with a finite set of attributes. The GNP can be used to exclude cyclic structures in components of arbitrary entities in interpretations of a given signature. Informally, it says that for each component of each entity, the component is not the entity itself. By adding an appropriate antecedent, the GNP can be turned into a principle that excludes cyclic structures in entities in the denotation of some sort(s), as determined by the antecedent. Below I will formulate a principle that restricts the effect of the GNP to entities of sort *ty2*.

Sailer's GENERAL NON-CYCLICITY PRINCIPLE uses a binary relation, `component`. The appropriate meaning of `component` can be defined in grammars with signatures whose set of attributes is finite and whose set of relation symbols contains a symbol `component` that is assigned arity two by the arity function. Assume that $\Sigma = \langle \mathcal{G}, \sqsubseteq, \mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{R}, \mathcal{AR} \rangle$ is such a signature. The set of clauses that define `component` is the smallest set C such that

$$\texttt{component}(x, y) \overset{\forall}{\Longleftarrow} x = y \;\in C, \text{ and}$$

for each $\alpha \in \mathcal{A}$,

$$\texttt{component}(x, y) \overset{\forall}{\Longleftarrow} {}^{y}[\alpha\ \boxed{1}] \wedge \texttt{component}(x, \boxed{1}) \;\in C.$$

In the definition of the meaning of `component`, x is a component of y. The TY2 NON-CYCLICITY PRINCIPLE, (143), uses that relation for requiring that no complex meaningful expression occur in any proper subexpression of itself. The principle is formulated with respect to a signature, $\Sigma = \langle \mathcal{G}, \sqsubseteq, \mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{R}, \mathcal{AR} \rangle$, that includes the signature of Ty2 and has a finite set of attributes.

(143) The TY2 NON-CYCLICITY PRINCIPLE:

$$ty2 \rightarrow$$

$$\forall \boxed{1} \left(\left(\bigvee_{\alpha \in \mathcal{A}} [\alpha\ \boxed{1}] \right) \rightarrow \neg\, \texttt{component}(:, \boxed{1}) \right)$$

The consequent of the principle in (143) is Sailer's GNP. The NON-CYCLICITY PRINCIPLE can be paraphrased as saying that for each entity, o, in the grammar of Ty2, if an attribute α is defined on o and $\boxed{1}$ is its value, then o is not a component of $\boxed{1}$. Informally, o is not cyclic.

Similarly, Sailer 2000 shows that it is possible to formulate a GENERAL FINITENESS PRINCIPLE (GFP) over signatures with a finite set of attributes. The GFP requires that every entity in every model of the grammar has finitely many components. Again, Sailer's general principle can easily be restricted to some subset of entities in the model of a grammar by turning it into the consequent of a principle with an appropriate antecedent. In our grammar of Ty2, an appropriate antecedent is again *ty2*.

(144) The TY2 FINITENESS PRINCIPLE:

$$\textit{ty2} \rightarrow$$
$$\exists\boxed{1}\forall\boxed{2}\left(\texttt{component}(\boxed{2},:) \rightarrow \texttt{member}\left(\boxed{2},\boxed{1}[\textit{chain}]\right)\right)$$

The crucial point of the TY2 FINITENESS PRINCIPLE is the reference to chains. Since chains are finite by definition, requiring that there is a chain of which every component of an entity is a member amounts to demanding that there only be finitely many components of that entity. Note that it is not necessary to require that every component occur only once on the chain. Repetitions of elements on the chain can be allowed without causing any harm. Note also that for *n-number* entities, (144) does not introduce any new restrictions, because they are already finite due to the theory of natural numbers, given in (140).

The TY2 IDENTITY PRINCIPLE follows the pattern of the preceding two principles. The GENERAL IDENTITY PRINCIPLE (GIP) of Sailer 2000 requires that whenever there is a pair of entities, o_1 and o_2, that have a common ancestor and that are the root of isomorphic configurations of entities, then o_1 and o_2 are identical. In other words, there is the maximal number of possible identities between the components of each entity. It is clear that the GIP is not an appropriate principle for all configurations of entities in the models of linguistic grammars. For example, if it were added to the grammar of Pollard and Sag 1994, it would require that in each sentence, each pair of indices whose NUMBER, GENDER and PERSON values are of the same species be identical. This would rule out all sentences in which the BINDING THEORY demands that two isomorphic indices be distinct. In the grammar of Ty2, however, an IDENTITY PRINCIPLE is useful. A very simple example may illustrate this. Consider the possible representations of the constant $c_{1,\langle e,\langle e,t\rangle\rangle}$ in models of the grammar of Ty2 as we have defined it so far. Without the TY2 IDENTITY PRINCIPLE, models of the grammar may contain two non-isomorphic representations of $c_{1,\langle e,\langle e,t\rangle\rangle}$, one satisfying the description in (145a), and the other one satisfying the description in (145b).

(145) a. $\begin{bmatrix} \textit{const} & \\ \text{N-NUM N-NUM} & \textit{zero} \\ \text{TYPE} & \begin{bmatrix} \text{IN} & \boxed{1}\ \textit{entity} \\ \text{OUT} & \begin{bmatrix} \text{IN} & \boxed{1}\ \textit{entity} \\ \text{OUT} & \textit{truth} \end{bmatrix} \end{bmatrix} \end{bmatrix}$

b. $\begin{bmatrix} \textit{const} & \\ \text{N-NUM N-NUM} & \textit{zero} \\ \text{TYPE} & \begin{bmatrix} \text{IN} & \boxed{1}\ \textit{entity} \\ \text{OUT} & \begin{bmatrix} \text{IN} & \boxed{2}\ \textit{entity} \\ \text{OUT} & \textit{truth} \end{bmatrix} \end{bmatrix} \end{bmatrix}$
$\wedge \neg \boxed{1} = \boxed{2}$

The only difference between the configurations under a *const* entity satisfying (145a) and (145b), respectively, is that those that satisfy (145a) have one *entity* component, and those that satisfy (145b) have two distinct *entity* components. Therefore, they are not isomorphic, although they represent the same non-logical constant. With the TY2 IDENTITY PRINCIPLE, we eliminate the second possibility. It should again be stressed that eliminating the possibility of non-isomorphic representations of the same Ty2 expressions is not a logical necessity, as long as we are able to identify the equivalence classes of entities that represent any given Ty2 expression.[24] However, turning the representations into unique indiscernibility classes is aesthetically preferable and makes the correspondence proof more straightforward.

Sailer's GIP uses a relation that can be defined with respect to any signature with a finite set of attributes. I call that relation copy. Assume that $\langle \mathcal{G}, \sqsubseteq, \mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{R}, \mathcal{AR} \rangle$ is a signature with a finite set of attributes, $\mathcal{A}$, whose set of relation symbols contains the symbol copy, which is assigned arity two by the arity function of the signature. Following Sailer 2000, I define the meaning of copy in one clause:

[24] For example, Richter and Sailer 1999c does not introduce the identity requirement in its grammar of the semantic representation language Ty2U. Ty2U is a variant of Ty2 that allows for underspecified semantic representations.

(146) $\texttt{copy}(x,y) \overset{\forall}{\Longleftarrow}$

$$\bigvee_{\sigma\in\mathcal{S}} \left({}^{x}[\sigma] \wedge {}^{y}[\sigma]\right) \wedge$$
$$\bigwedge_{\alpha\in\mathcal{A}} \left(\forall\boxed{1}\left({}^{x}[\alpha\ \boxed{1}] \rightarrow \exists\boxed{2}\left({}^{y}[\alpha\ \boxed{2}] \wedge \texttt{copy}(\boxed{1},\boxed{2})\right)\right)\right)$$

y is a copy of x if x and y have the same species (second line), and if an attribute α with value $\boxed{1}$ is defined on x, then α is also defined on y, and $\boxed{1}$ and the value of α on y, $\boxed{2}$, are in the copy relation (third line).

Two entities that stand in the copy relation are of the same species, and they have the same attributes defined on them; and corresponding attribute values stand again in the copy relation. In the TY2 IDENTITY PRINCIPLE, I require that all the corresponding entities in the copy relation of representations of Ty2 expressions be identical:

(147) The TY2 IDENTITY PRINCIPLE:

$$\textit{ty2} \rightarrow$$
$$\forall\boxed{1}\forall\boxed{2}\left(\texttt{copy}(\boxed{1},\boxed{2}) \rightarrow \boxed{1} = \boxed{2}\right)$$

The consequent of my principle is Sailer's GIP.

The TY2 IDENTITY PRINCIPLE concludes the specification of the RSRL grammar of Ty2 expressions. Sailer 2000 proves the crucial properties of the RSRL grammar of Ty2. Firstly, there is an exhaustive model of the grammar whose universe contains exactly the natural numbers, the set of types, *Types*, and the meaningful expressions of Ty2. This result shows that the grammar indeed specifies what we want to specify with it. Secondly, for each indiscernibility class of entities of sort *me* in an exhaustive model of the grammar, there exists a corresponding Ty2 expression that has the same meaning. To prove this result, it is of course necessary to define the notion of *correspondence* between indiscernibility classes of *me* entities and Ty2 expressions, and to define a semantics for indiscernibility classes of *me* entities. In fact, both definitions are straightforward: Equivalence classes of indiscernible *me* entities receive the standard Ty2 semantics, and the correspondence is defined in the way that is intuitively expected. For example, an entity in an exhaustive model of the Ty2 grammar that is described by (145a) corresponds to the constant $c_{1,\langle e,\langle e,t\rangle\rangle}$. Thirdly, for each Ty2 expression there exists an AVM description that picks out exactly the indiscernibility class of entities in models of the RSRL grammar of Ty2 whose *me* entities correspond to the Ty2 expression. It is this result that allows us to use Ty2 expressions in AVM descriptions in places where the AVM syntax would require a description of the Ty2 expression. Since those descriptions are very cumbersome to write and to read, the abbreviatory convention of notating the described expression instead is very convenient.

With the Ty2 grammar in place, I can formulate a first approximation to a SEMANTICS PRINCIPLE (SP). For expository reasons, I assume a binary branching syntactic structure, where every phrase has a head daughter and exactly one nonhead daughter. It is not hard to generalize this specialized SP to different assumptions about the syntactic phrase structure. The SP should license semantic composition by functional application. This leaves two possibilities: Either the head daughter is the functor, and the nonhead daughter is the argument, or *vice versa*. The type of the semantic representations of the daughters determines which one of the two possibilities holds. If neither one of the two cases applies, the SP in (148) excludes the phrase from models of the grammar.

(148) SEMANTICS PRINCIPLE (first approximation)

$$
\textit{phrase} \rightarrow
\left(\begin{bmatrix} \text{SYNSEM} & \begin{bmatrix}\text{LOC CONTENT LF} & \begin{bmatrix} \textit{application} & \\ \text{FUNC} & \boxed{3} \\ \text{ARG} & \begin{bmatrix}\textit{abstraction} & \\ \text{VAR} & \begin{bmatrix}\text{TYPE} & \textit{w-index}\\ \text{N-NUM} & \textit{zero}\end{bmatrix} \\ \text{ARG} & \boxed{4}\end{bmatrix}\end{bmatrix}\end{bmatrix} \\ \text{DTRS} & \begin{bmatrix}\text{HEAD-DTR SYNSEM LOC CONTENT LF} & \boxed{1}\\ \text{NONHEAD-DTR SYNSEM LOC CONTENT LF} & \boxed{2}\end{bmatrix}\end{bmatrix} \wedge \left(\begin{array}{c} (\boxed{1}=\boxed{3} \;\wedge\; \boxed{2}=\boxed{4}) \\ \vee \\ (\boxed{1}=\boxed{4} \;\wedge\; \boxed{2}=\boxed{3})\end{array}\right)\right)
$$

If $\boxed{1} = \boxed{3}$ and $\boxed{2} = \boxed{4}$, then the head daughter is the functor, and the nonhead daughter is the argument. If $\boxed{1} = \boxed{4}$ and $\boxed{2} = \boxed{3}$, then the nonhead daughter is the functor, and the head daughter is the argument. (148) also shows that the standard analysis of functional application in natural language semantics in Ty2 involves lambda abstraction over the world index of the argument. For that reason, the ARG value of the *application* is an *abstraction*. To ensure that the SP abstracts over the right world index, I assume with Groenendijk and Stokhof 1982, p. 187, that all our basic expressions, i.e., the expressions in the lexical entries of the linguistic grammar, contain the same free index variable. The SP abstracts over the world index variable that is the world index variable of the lexical entries. Like Groenendijk and Stokhof, I reserve the first variable of type s, $v_{0,s}$ for that purpose. In the Ty2 notation below, I write w for that variable.

The example in (149) illustrates the effect of the SP with the nominal phrase *every unicorn*. The type of the semantic expressions determines that the syntactic nonhead, *every*, must be the functor, and *unicorn* is the argument. In addition, the SP introduces the lambda abstraction over the world index, w, of *unicorn*.

(149) *An illustration of the* SEMANTICS PRINCIPLE

NP

$$\begin{bmatrix} \textsc{phon} & \langle \textit{every, unicorn} \rangle \\ \textsc{ss loc cont lf} & \lambda P \lambda Q.\forall x[P(w)(x) \rightarrow Q(w)(x)](\lambda w.\mathsf{unicorn}'(w)) \end{bmatrix}$$

C H

Det N

$$\begin{bmatrix} \textsc{phon} & \langle \textit{every} \rangle \\ \textsc{ss loc cont lf} & \lambda P \lambda Q.\forall x[P(w)(x) \rightarrow Q(w)(x)] \end{bmatrix} \qquad \begin{bmatrix} \textsc{phon} & \langle \textit{unicorn} \rangle \\ \textsc{ss loc cont lf} & \mathsf{unicorn}'(w) \end{bmatrix}$$

In the introductory example, (138), I have indicated that my goal is to specify the principles of semantics in a way that leads to a fully β-reduced expression when the logical forms of two phrases are combined by functional application. As we can see at the top node in (149), this is not the case yet with the SP in (148). If we want the expressions that represent the meaning of linguistic signs to be redex free, we need to add a mechanism that expresses β-reduction.

5.4.2 An Operational Extension

In this section, I sketch an extension of the grammar of Ty2 that furnishes our logical forms with β-reduction. This operational extension of Ty2 is another example of the descriptive power of RSRL. It shows how a technically desirable mechanism can be made part of a declarative grammar. I proceed in three steps: First, I anchor the sorts and attributes of the signature of the meaningful expressions of Ty2 and of the theory of β-reduction in the standard signature of HPSG grammars by declaring the appropriate attributes and attribute values for *content*, and I specify the theory of β-reduction as a principle in the theory of the grammar. Second, I introduce a relation that allows me to require that a meaningful expression be redex free. The new relation leads directly to the β-REDUCTION PRINCIPLE, which relies on the theory of β-reduction specified before. The β-REDUCTION PRINCIPLE requires that all LOGICAL-FORM values of *content* entities be redex free. Third, the SEMANTICS PRINCIPLE is modified to respect the addition of β-reduction to the grammar.

My specification of β-reduction essentially follows the proposal by Richter and Sailer (1999a), who choose to implement β-reduction as a recursive sort within CONTENT values. To provide the "work space" for β-reduction, two attributes are appropriate to *content*: The attribute LOGICAL-FORM, which we have already seen above, and the attribute β-REDUCTION. The meaningful expressions of Ty2 that determine the model-theoretic meaning of a linguistic sign are the value of the attribute LOGICAL-FORM. A new sort, β*-reduction*, is appropriate for the attribute β-REDUCTION. With the theory of the sort β*-reduction*, to which I will turn presently, the β-REDUCTION values in models of the grammar are recursively structured configurations of β*-reduction* entities that specify the β-reduction steps of complex expressions.[25]

(150) The sort hierarchies of *content* and of β*-reduction*:

content LOGICAL-FORM *me*
β-REDUCTION β*-reduction*

β*-reduction* TERM *me*
AUX *me*

no-reduction

one-reduction REC β*-reduction*

[25] The present encoding of β*-reduction* as recursively structured configurations of entities bears a noticeable similarity to the junk slot method of encoding relations that is discussed in Section 2.2.3. As mentioned there, a reification of relations as configurations of entities is regarded a serious conceptual problem that keeps linguists from using junk slots, and the same argument could be leveled against the related encoding of β*-reduction*. Fortunately, Sailer 2000 shows how in RSRL β*-reduction* can alternatively be encoded as chains in the arguments of relations, i.e., Sailer's encoding avoids additional attributes and entities of new species in signs, and the reduction steps are no longer present in the components of signs. The reason I do not follow Sailer's construction here is that the present approach is technically much simpler. My choice should not be taken as an indication of preference.

According to the principle formulated in (151) below, the value of the attribute TERM is an expression that is either identical to the expression under AUX, or it is derived from the AUX value by β-reduction. The idea of the recursive structure of *β-reduction* is that if no reduction step is performed, the AUX value equals the TERM value. If there is a reduction, the REC AUX value is deduced from the AUX value by one reduction step, and the TERM value is passed up from the REC TERM value, i.e., ultimately it is passed up from the bottom of the reduction chain deeper down in the recursive embedding of *β-reduction* entities.

(151) The theory of *β-reduction*:

$$\textit{no-reduction} \rightarrow \begin{bmatrix} \text{TERM} & \boxed{1} \\ \text{AUX} & \boxed{1} \end{bmatrix}$$

$$\begin{array}{l} \textit{one-reduction} \rightarrow \begin{bmatrix} \text{TERM} & \boxed{1} \\ \text{AUX} & \boxed{2} \\ \text{REC} & \begin{bmatrix} \text{TERM} & \boxed{1} \\ \text{AUX} & \boxed{3} \end{bmatrix} \end{bmatrix} \\ \quad \wedge\ \texttt{component}\left({}^{y}\begin{bmatrix} \textit{appl} & \\ \text{FUNC} & \begin{bmatrix} \textit{abstr} & \\ \text{VAR} & \boxed{5} \\ \text{ARG} & \boxed{4} \end{bmatrix} \\ \text{ARG} & \boxed{6} \end{bmatrix}, \boxed{2} \right) \\ \quad \wedge\ \texttt{replace}(\boxed{4}, \boxed{5}, \boxed{7}, \boxed{6}) \\ \quad \wedge\ \texttt{replace}(\boxed{2}, y, \boxed{3}, \boxed{7}) \end{array}$$

In the non-recursive case, the TERM value and the AUX value are simply identical. In the recursive case, the β-reduced TERM value is passed up from REC TERM, and the reduction step is taken care of by the relations `component` and `replace`. The relation `component` is used to identify a redex sub-expression, y, of $\boxed{2}$. Informally, y is a redex in the input formula, $\boxed{2}$, which is the AUX value of the *one-reduction* entity. $\boxed{7}$ is a sub-expression of the REC AUX value, $\boxed{3}$, such that $\boxed{7}$ is obtained from y by β-reduction. In the general case, the replacement of the variable, $\boxed{5}$, by $\boxed{6}$ in the argument of the abstraction, $\boxed{4}$, is not the only replacement. The second occurrence of the `replace` relation ensures that we get the overall β-reduced expression, $\boxed{3}$, by replacing the former abstraction, y, in $\boxed{2}$, by the expression, $\boxed{7}$, that we obtain from contracting y.

The real work is done by the relation `replace`, whose definition is sketched in (152).

(152) The relation `replace`:

The base cases:

$$\texttt{replace}(x, y, v, w) \stackrel{\forall}{\Longleftarrow} x = y \ \land \ v = w$$

$$\texttt{replace}(x, y, v, w) \stackrel{\forall}{\Longleftarrow} \neg x = y \land {}^{x}[\mathit{var} \lor \mathit{const}] \land x = v$$

An example of a recursive case:

$$\texttt{replace}(x, y, v, w) \stackrel{\forall}{\Longleftarrow} \neg x = y \land {}^{x}\begin{bmatrix} \mathit{abstr} & \\ \text{TYPE} & \boxed{1} \\ \text{VAR} & \boxed{2} \\ \text{ARG} & \boxed{3} \end{bmatrix} \land {}^{v}\begin{bmatrix} \mathit{abstr} & \\ \text{TYPE} & \boxed{1} \\ \text{VAR} & \boxed{2} \\ \text{ARG} & \boxed{4} \end{bmatrix} \land \texttt{replace}(\boxed{3}, y, \boxed{4}, w)$$

v is obtained by replacing y by w in x. The recursion goes over the structure of the expression x. There are two base cases: If x is identical to y, then the reduced version of y, namely w, is identical to v. If x is not identical to y, but an atomic expression, i.e., either a variable or a constant, then x is identical to v. In this case, we were checking a sub-expression of x that did not contain y. In the recursive case, the structure of x is copied to v, and the relation `replace` is applied to all recursive sub-expressions of x.

Note that as it stands, our β-reduction differs from the standard definition in textbooks:[26] We never rename bound variables or, in other words, α-conversion is not provided by the above specification of the theory of β-reduction. Except for the variable whose occurrences are replaced in the reduction, the variables in the redex and in the contracted expression are identical. Variables that are free in the argument of a β-redex may, thus, become bound in the contracted expression. It follows that our β-reduction does not necessarily preserve the meaning of an expression. In (153), I illustrate the (potential) problem. The symbol '$\triangleright_{1\beta}$' indicates that one β-reduction step is performed to get from the expression to its left to the expression to its right. The symbol '$\not\triangleright_{1\beta}$ signals non-standard β-reduction as permitted by (151). The symbol '$\equiv_\alpha$' signals α-conversion, sometimes also called a change of a bound variable.

(153) a. $(\lambda v_{1,e}.\exists v_{2,e}\, \mathsf{pred}(v_{2,e}, v_{1,e}))\,(v_{2,e}) \ \ \not\triangleright_{1\beta}\ \exists v_{2,e}\, \mathsf{pred}(v_{2,e}, v_{2,e})$

[26] For example, see Hindley and Seldin 1986, p. 11.

b. $(\lambda v_{1,e}.\exists v_{2,e}\ \mathsf{pred}(v_{2,e}, v_{1,e}))\ (v_{2,e})$
$\equiv_\alpha\ (\lambda v_{1,e}.\exists v_{3,e}\ \mathsf{pred}(v_{3,e}, v_{1,e}))\ (v_{2,e})$
$\rhd_{1\beta}\ \exists v_{3,e}\ \mathsf{pred}(v_{3,e}, v_{2,e})$

The first example, (153a), is an instance of the pseudo-β-reduction licensed by our theory (151) but not licensed by standard β-reduction, as indicated by '$\not\rhd_{1\beta}$.' This pseudo-reduction step does not preserve the meaning of the expression, because the variable $v_{2,e}$ that is free in the argument of the redex in the first expression is bound by the existential quantification in the expression to the right of '$\not\rhd_{1\beta}$.' The second example, (153b), is an instance of standard β-reduction, which involves α-conversion, i.e., a change of the variable bound by the existential quantifier. Procedurally speaking, we first replace the bound occurrences of the variable, $v_{2,e}$, by $v_{3,e}$, thus obtaining the expression $(\lambda v_{1,e}.\exists v_{3,e}\ \mathsf{pred}(v_{3,e}, v_{1,e}))\ (v_{2,e})$, and then we perform the β-reduction.

Sailer 2000 shows how (151) can be modified to forbid the non-standard β-reduction indicated in (153a). For that purpose, Sailer enriches the sort hierarchy under *one-reduction* with two additional subsorts of *one-reduction*, *β1-reduction* and *α-conversion*. The former theory of *one-reduction* becomes the theory of *β1-reduction* by adding a conjunct to the consequent of the principle that blocks contraction if an unbound variable would get into the scope of some logical operator, and enforces α-conversion instead. *α-conversion* is of course governed by an appropriate principle that restricts the admissible configurations under *α-conversion* entities. I omit Sailer's extension here, since it does not add any new insights into the descriptive flexibility of RSRL to what we have already seen.[27]

Given the theory of the sort *β-reduction*, we only need to define one more auxiliary relation, `redex-free`, before we can state a principle that demands that the LOGICAL-FORM value of each *content* entity be redex free. The definition of `redex-free` in (154) uses the relation `component`, defined on page 358.

[27] It should be mentioned that the linguistic theories in Richter and Sailer 1999a,b deliberately exploit the non-standard definition of β-reduction given in (151). Some analyses of indefinites in natural languages postulate that their logical forms contain free variables that must be bound by a mechanism of existential closure within certain syntactic domains. Richter and Sailer show that the non-standard β-reduction offers the possibility of introducing the necessary existential closure by appropriate lexical elements, such as Polish finite verbs with the negation prefix *nie* (Richter and Sailer, 1999b, pp. 272–274).

(154) The relation `redex-free`:

$$\texttt{redex-free}(x) \stackrel{\forall}{\Longleftarrow} \neg\, \exists y \left({}^{y}\begin{bmatrix} \textit{appl} \\ \text{FUNC} \;\; \textit{abstr} \end{bmatrix} \wedge \texttt{component}(y, {}^{x}[\textit{me}]) \right)$$

According to (154), a meaningful expression x is redex free exactly if it does not contain an application whose functor is an abstraction.

The β-REDUCTION PRINCIPLE, (155), requires that the LF value of a *content* entity be always fully β-reduced. It equals the "output" of our mechanism for β-reduction, the β-REDUCTION TERM value of the *content* entity:

(155) The β-REDUCTION PRINCIPLE:

$$\textit{content} \rightarrow \left(\begin{bmatrix} \text{LOGICAL-FORM} & \boxed{1} \\ \beta\text{-REDUCTION TERM} & \boxed{1} \end{bmatrix} \wedge \texttt{redex-free}(\boxed{1}) \right)$$

We are now ready to reformulate the SEMANTICS PRINCIPLE. Recall that, for ease of exposition, we assume an overall binary branching syntactic structure when we state the SP. For each phrase, the SP provides the possibility of functional application of one syntactic daughter to the other. In conjunction with the theory of *β-reduction*, (151), and the β-REDUCTION PRINCIPLE, (155), the new SP says that the LOGICAL-FORM value of a phrase is the redex free expression that is derived by β-reduction from the expression obtained by functional application of the LOGICAL-FORM value of one syntactic daughter to the LOGICAL-FORM value of the other syntactic daughter of the phrase. As already explained in connection with the first formulation of the SP, we always abstract over the world index, $v_{0,s}$, of the argument of the functional application.

(156) *The* SEMANTICS PRINCIPLE *(final version)*

$$\textit{phrase} \rightarrow \begin{bmatrix} \text{SS} & \begin{bmatrix} \text{LOC CONT } \beta\text{-REDUCT} & \begin{bmatrix} \textit{\beta-reduction} \\ \text{AUX} & \begin{bmatrix} \textit{application} \\ \text{FUNC} & \boxed{3} \\ \text{ARG} & \begin{bmatrix} \textit{abstraction} \\ \text{VAR} & \begin{bmatrix} \text{TYPE} & \textit{w-index} \\ \text{N-NUM} & \textit{zero} \end{bmatrix} \\ \text{ARG} & \boxed{4} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{DTRS} & \begin{bmatrix} \text{HEAD-DTR SS LOC CONT LF} & \boxed{1} \\ \text{NONHEAD-DTR SS LOC CONT LF} & \boxed{2} \end{bmatrix} \end{bmatrix}$$
$$\wedge \; ((\boxed{1} = \boxed{3} \wedge \boxed{2} = \boxed{4}) \vee (\boxed{1} = \boxed{4} \wedge \boxed{2} = \boxed{3}))$$

The formulation of the SP in (156) leads to the desired result, as illustrated in (157):

(157) *An illustration of the final* SEMANTICS PRINCIPLE

$$\text{NP}$$

$$\begin{bmatrix} \text{PHON} & \langle \textit{every, unicorn} \rangle \\ \text{SS LOC CONT} & \begin{bmatrix} \textit{content} & \\ \text{LF} & \boxed{1}\ \lambda Q.\forall x[\mathsf{unicorn}'(w)(x) \rightarrow Q(w)(x)] \\ \beta\text{-REDUCTION} & \begin{bmatrix} \text{TERM} & \boxed{1} \\ \text{AUX} & (\boxed{2})(\lambda w\ \boxed{3}) \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

C — H

$$\text{Det}$$

$$\begin{bmatrix} \text{PHON} & \langle \textit{every} \rangle \\ \text{SS LOC CONT LF} & \boxed{2}\ \lambda P \lambda Q.\forall x[P(w)(x) \rightarrow Q(w)(x)] \end{bmatrix}$$

$$\text{N}$$

$$\begin{bmatrix} \text{PHON} & \langle \textit{unicorn} \rangle \\ \text{SS LOC CONT LF} & \boxed{3}\ \mathsf{unicorn}'(w) \end{bmatrix}$$

The semantic types of the quantifier and of the common noun determine that the logical form of the former is applied to the logical form of the latter. This application can be observed in the β-REDUCTION AUX value of the *content* of the mother node. The β-REDUCTION PRINCIPLE in combination with the theory of β-reduction leads to the fully contracted expression in the β-REDUCTION TERM value, which is in turn identified with the LOGICAL-FORM value of the mother node (due to the β-REDUCTION PRINCIPLE).

5.4.3 A Principle at the Syntax-Semantics Interface

I conclude the discussion of the integration of two-sorted type theory into HPSG grammars with an example of a principle that illustrates how the interaction of syntax and semantics can be accounted for in the present architecture. It demonstrates how to express restrictions on logical form straightforwardly as restrictions on expressions of Ty2.

It is well-known that every arbitrary logical expression may not be the logical form of a matrix sentence. Richter and Sailer 1999a,b subsume restrictions on the possible logical form of matrix sentences under their PRINCIPLE OF FULL INTERPRETATION (PFI). The PFI imposes well-formedness conditions on the possible logical forms of certain distinguished expressions of a natural language. Richter and Sailer's PFI restricts the logical form of unembedded signs, i.e., expressions that carry illocutionary force.[28] Matrix sentences are a prototypical instance of unembedded signs.

(158) *The* PRINCIPLE OF FULL INTERPRETATION *(PFI) (after Richter and Sailer 1999a,b)*

In the logical form of an unembedded sign,

1. every quantifier binds an occurrence of a variable, and
2. except for the world index variable, $v_{0,s}$, there are no free variables.

The PFI has a number of non-trivial implications for unembedded sentences. For example, those theories in which indefinites are analyzed as introducing free variables must provide a mechanism for binding off these variables in unembedded sentences. This prompts the question where the logical operators that do that come from, where they can enter into the logical form of a sentence, and which principles control their occurrence. Here, I will not pursue questions of the empirical and technical adequacy of the PFI. In the present discussion, the PFI is only meant as an illustration of how principles at the interface between the logical form of signs and some module of the grammar outside of semantics are formalized when the logical form of a sign is an expression of some logical language.

What is formally interesting about the PFI is that for some syntactic environments (unembedded signs), it imposes complex restrictions on the shape of the expressions that may occur as their logical form. Since these expressions determine the meaning of a sign, the PFI also restricts the possible meanings of unembedded signs. Just like the principles of Pollard and Sag 1994 restrict admissible configurations of syntactic or situation-semantic entities, the PFI looks at logical forms as configurations of entities on which it imposes well-formedness conditions.

The kind of principle that we see in (158) is reminiscent of the QUANTIFIER BINDING CONDITION and of the BINDING THEORY. Where the QUANTIFIER BINDING CONDITION declares that every occurrence of a certain index must be captured by a certain quantifier, the first clause of the PFI

[28] The PFI in (158) differs slightly from the version given in Richter and Sailer 1999a,b. In particular, I omit their third clause that is motivated by specific technical assumptions of their grammars of French and Polish.

says that every quantifier binds at least one variable; and where Principle A of the BINDING THEORY declares that certain anaphors must be locally o-bound, the second clause of the PFI says that with one exception, all variables in the logical form of unembedded signs must be bound by a quantifier. Given the conceptual similarities between the principles, and since it is not hard to express the QUANTIFIER BINDING CONDITION and Principle A of the BINDING THEORY in RSRL, a formalization of the PFI does not pose any problems either.

For the formalization of (158), we need one new binary relation, `free`, which relates variables to the complex expressions in which they occur free. In (159), x occurs free in y:

(159) The relation `free`:

$$\texttt{free}(x,y) \overset{\forall}{\Longleftarrow} x = y \ \wedge\ {}^{y}[\textit{var}]$$

$$\texttt{free}(x,y) \overset{\forall}{\Longleftarrow} {}^{y}\begin{bmatrix} \textit{appl} & \\ \text{FUNC} & \boxed{1} \\ \text{ARG} & \boxed{2} \end{bmatrix} \wedge \left(\texttt{free}(x, \boxed{1}) \vee \texttt{free}(x, \boxed{2})\right)$$

$$\texttt{free}(x,y) \overset{\forall}{\Longleftarrow} {}^{y}\begin{bmatrix} \text{ARG1} & \boxed{1} \\ \text{ARG2} & \boxed{2} \end{bmatrix} \wedge \left(\texttt{free}(x, \boxed{1}) \vee \texttt{free}(x, \boxed{2})\right)$$

$$\texttt{free}(x,y) \overset{\forall}{\Longleftarrow} {}^{y}\begin{bmatrix} \textit{neg} & \\ \text{ARG} & \boxed{1} \end{bmatrix} \wedge \texttt{free}(x, \boxed{1})$$

$$\texttt{free}(x,y) \overset{\forall}{\Longleftarrow} {}^{y}\begin{bmatrix} \textit{quant} & \\ \text{VAR} & \boxed{1} \\ \text{SCOPE} & \boxed{2} \end{bmatrix} \wedge \neg x = \boxed{1} \wedge \texttt{free}(x, \boxed{2})$$

$$\texttt{free}(x,y) \overset{\forall}{\Longleftarrow} {}^{y}\begin{bmatrix} \textit{abstr} & \\ \text{VAR} & \boxed{1} \\ \text{ARG} & \boxed{2} \end{bmatrix} \wedge \neg x = \boxed{1} \wedge \texttt{free}(x, \boxed{2})$$

With the relation `free` in hand, I can express the PFI without any further definitions. The only other relation that is needed is the relation `component`, which was defined on page 358.

(160) *The* PRINCIPLE OF FULL INTERPRETATION *formalized (after Richter and Sailer 1999a, p. 289)*

$$
\begin{array}{l}
\textit{unembedded-sign} \quad \rightarrow \\
\left(\begin{array}{l}
\left[\text{SS LOC CONT LF } \boxed{1}\right] \\
\wedge\, \forall x \left(\begin{array}{l}
\texttt{component}\left({}^{x}[\textit{quant}], \boxed{1}\right) \\
\rightarrow \exists \boxed{3} \exists \boxed{2} \left({}^{x}\begin{bmatrix}\text{VAR} & \boxed{3} \\ \text{SCOPE} & \boxed{2}\end{bmatrix} \wedge \texttt{free}(\boxed{3}, \boxed{2})\right)
\end{array}\right) \\
\wedge\, \forall x \left(\texttt{component}\left({}^{x}[\textit{var}], \boxed{1}\right) \rightarrow \left({}^{x}\begin{bmatrix}\textit{var} & \\ \text{N-NUMBER} & \textit{zero} \\ \text{TYPE} & \textit{w-index}\end{bmatrix} \vee \neg\, \texttt{free}(x, \boxed{1})\right)\right)
\end{array}\right)
\end{array}
$$

The second conjunct in the consequent says that if a *quantifier* entity, x, in the logical form, $\boxed{1}$, of an unembedded sign, quantifies over the variable $\boxed{3}$, then $\boxed{3}$ occurs free in the expression, $\boxed{2}$, that is the SCOPE value of the quantifier. In other words, each quantifier binds a variable. The third, and last, conjunct says that for each *variable* entity in the logical form, $\boxed{1}$, of an unembedded sign, the variable does not occur free in $\boxed{1}$, with the possible exception of the world index variable, $v_{0,s}$. In other words, no variable except for $v_{0,s}$ may occur unbound in the logical form of an unembedded sign.

5.4.4 Summary

In this section, we saw a number of technical principles that tell us more about the expressive power of RSRL. Inasmuch as RSRL is a formal explication of the formalism that is implicit in Pollard and Sag 1994, they also tell us more about possible HPSG 94 grammars of natural languages.

The main result of this section is that HPSG 94 is sufficiently expressive to allow for the specification of symbolic languages like Ty2, including an operational extension for β-reduction and α-conversion as they are usually defined in the λ-calculus. The linguistic significance of this result is that without an extension of the formalism that is implicit in HPSG 94 from the very beginning, HPSG grammars may employ expressions of standard logical languages, just as it is usually done elsewhere in the literature on the formal semantics of natural languages. It is therefore possible to write rigorously formalized HPSG grammars of natural languages that comprise sophisticated syntactic structures as well as expressions of standard logical languages with well-studied model-theoretic interpretations. The integration of Ty2 with the semantically uninterpreted syntactic structures of HPSG grammars offers new perspectives to grammar writers, allows for a more direct comparison of the semantic structures of HPSG grammars with the semantic structures of other linguistic theories that employ logical forms, and makes the empirical and analytical insights of the pertinent literature more easily applicable to the design of HPSG grammars. With Richter and Sailer 1999a and Richter and Sailer 1999b, there already exist two studies that illustrate this new line of research.

The specification of a grammar of Ty2 with an operational extension crucially requires the use of all extensions that RSRL adds to SRL. Existential and universal component quantification, relations in the scope of negation, and chains are indispensable for the formalization of the semantic representations that I discussed above. Two technical principles that are needed in the specification of Ty2 deserve to be mentioned in this context, since they are potentially relevant for other grammatical specifications as well. The TY2 NON-CYCLICITY PRINCIPLE, (143), showed that, as long as the set of attributes of the signature of a grammar is finite, cyclic structures can be forbidden either throughout all models of the grammar or in designated parts of the models. For example, cyclic structures might be allowed everywhere except for the expressions of Ty2. The TY2 FINITENESS PRINCIPLE, (144), showed that, under the same restriction on the signatures of a grammar, entities with infinitely many components can be excluded from all models of the grammar. Apart from being an important formal result about the expressive power of a formalism of HPSG 94, a general finiteness principle might, at least to some linguists, be a theoretically welcome constraint on models of natural languages.