Subtyping can have a Simple Semantics[*]

Herman Balsters , Maarten M. Fokkinga[†]
University of Twente, Department of Computer Science
PO Box 217, NL 7500 AE Enschede
The Netherlands

Abstract

Consider a first order typed language, with semantics $[\![\ ]\!]$ for expressions and types. Adding subtyping means that a partial order $\leq$ on types is defined and that the typing rules are extended to the effect that expression e has type τ whenever e has type σ and $\sigma \leq \tau$. We show how to adapt the semantics $[\![\ ]\!]$ in a *simple set-theoretic* way, obtaining a semantics $[\{\ \}]$ that satisfies, in addition to some obvious requirements, also the property: $[\{\sigma\}] \subseteq [\{\tau\}]$, whenever $\sigma \leq \tau$.

1 Introduction and results

The usefulness of a typing discipline in programming is widely known and recognized: compile-time type checking may detect errors before they lead to calamitous results, it may facilitate efficiency improvements (such as the omission of run-time domain checks), and it may guarantee nice semantic properties (such as termination, or the existence of simple set-theoretic semantics). A typing discipline means that in a program each constituent part is assigned—in some way or another—an attribute (called *type*) and that certain relationships are required to hold between the types assigned, if the program is to be considered well-formed and acceptable for evaluation. Typing disciplines have been extensively studied; see e.g. [Gries 1978; introduction to Part IV], [Fokkinga 1981, 1987], [Cardelli, Wegner 1985], [Hindley, Seldin 1986] and many others.

Subtyping is a feature of a typing discipline that may control the automatic insertion of implicit operations; it may also be used to model the inheritance relation in object-oriented languages, [Cardelli 1984]. Roughly said, we speak of subtyping when

- a partial order exists on types, and for types σ, τ with $\sigma \leq \tau$ there exists a ("conversion") operation $cv_{\sigma\leq\tau}$ that behaves like a function mapping arguments of type σ into results of type τ
- an expression e of type σ is allowed to occur at a position where something of type τ is required, provided that $\sigma \leq \tau$ and that the operation $cv_{\sigma\leq\tau}$ is applied (implicitly) to the value of e

Reynolds [Reynolds 1985] gives an excellent overview of various possibilities of typing and subtyping.

Our description of typing and subtyping, mentioned above, is of a syntactical nature. It goes without saying that the question arises quickly whether types themselves have a meaning, i.e.

[*]This work was partly conducted within the PRISMA project, a joint effort with Philips Research Eindhoven, partially supported by the Dutch "Stimulerings Projectteam Informaticaonderzoek Nederland (SPIN)".
[†]Currently at Centre for Mathematics and Computer Science, Amsterdam.

whether there exists a (mathematical) semantics for types (and subtyping). Let us denote the semantics of closed expressions e and types τ by $[\![e]\!]$ and $[\![\tau]\!]$, respectively. It would be nice if the semantics $[\![\tau]\!]$ of type τ is merely a set such that $[\![e]\!] \in [\![\tau]\!]$ whenever e has type τ. However, only for simple (so-called first order, non-recursive) types such a simple set-theoretic semantics seems possible. Most often one finds types interpreted as "domains" (continuous lattices or the like) and sometimes a set-theoretic interpretation is proved to be impossible [Reynolds 1984]. The semantics of *sub*typing is our prime concern in this paper.

We set out to construct, by *simple set-theoretic* means, a semantics for types –in the presence of subtyping– such that

$$[\![\sigma]\!] \subseteq [\![\tau]\!] \quad \text{whenever} \quad \sigma \leq \tau$$

This poses serious semantical problems. Consider for example the following situation:

- Assume that $[\![(\sigma \to \tau)]\!]$ = some non-empty set of functions that have domain $[\![\sigma]\!]$ and co-domain $[\![\tau]\!]$
- Assume that $[\![\mathit{int}]\!] \subset [\![\mathit{real}]\!]$
- Assume that $\mathit{int} \leq \mathit{real}$, so that, as motivated in Section 3, $(\mathit{real} \to \tau) \leq (\mathit{int} \to \tau)$

We then find that the desire $[\![(\mathit{real} \to \tau)]\!] \subseteq [\![(\mathit{int} \to \tau)]\!]$ contradicts the following two observations:

- Functions $f \in [\![(\mathit{real} \to \tau)]\!]$ cannot belong to $[\![(\mathit{int} \to \tau)]\!]$ because the domain of f differs from $[\![\mathit{int}]\!]$
- The cardinality of $[\![(\mathit{real} \to \tau)]\!]$ is strictly larger than that of $[\![(\mathit{int} \to \tau)]\!]$

Several authors have attacked this problem, and have solved it by non-simple (Scottery, categorical) domain constructions for $[\![\tau]\!]$; [MacQueen et al 1984], [Cardelli 1984], [Bruce, Wegner 1987].

Our solution, on the contrary, is as simple as effective, and can be stated in a single line. Given a semantics $[\![\ ]\!]$ for the language without subtyping, we form a new semantics $[\{\ \}]$ when subtyping is added, by defining

$$[\{\tau\}] = \bigcup_{\sigma \leq \tau} [\![\sigma]\!]$$

For now we have, when $\sigma \leq \tau$, that

$$\begin{array}{rcll} [\{\sigma\}] & = & \bigcup_{\rho \leq \sigma} [\![\rho]\!] & \\ & \subseteq & \bigcup_{\rho \leq \tau} [\![\rho]\!] & (\textit{transitivity of } \leq, \sigma \leq \tau) \\ & = & [\{\tau\}] & \end{array}$$

Note that we have used only elementary, primary school set-theoretic constructions in the definition of $[\{\ \}]$ for types. However, this still leaves us with the problem of defining $[\{\ \}]$ for expressions in such a way that

- $[\{e\}] \in [\{\tau\}]$ whenever expression e has type τ, and
- $[\{\ \}]$ is in a natural way related to $[\![\ ]\!]$

The first part is not hard to achieve. The second part poses some technical problems: we would like to "define" $[\{\ \}]$ by certain equations – these equations, however, turn out to be ambiguous. We can only succeed in showing that the ambiguity is not harmful by defining a *minimal typing* (that is sound and complete with respect to the given typing), defining a semantics based on this minimal typing, and then proving that the desired equations do hold for those semantics.

The method described above, viz. constructing the required semantics $[\{\ \}]$ from a given semantics $[\![\ ]\!]$, is demonstrated by means of a simple language containing representatives for quite a number of practical programming language constructs. One concept that we do not take into account is (general) recursion; as a consequence the given semantics $[\![\ ]\!]$ can be kept quite simple. (If one adds recursion, the semantics $[\![\tau]\!]$, for types τ, should be a complete partial order (c.p.o.) or an even more complex structure. But even then our technique of defining $[\{\tau\}] = \bigcup_{\sigma\leq\tau}[\![\sigma]\!]$ works, even though the resulting $[\{\tau\}]$ is not a c.p.o. - and note that there is also no need for it to be a c.p.o.). There are also various aspects of polymorphism, apart from subtyping, that we do not take into account in this paper. One particular aspect of polymorphism is type inference, according to which expressions can have many types, and that type instances of these expressions belong to certain type schemes (e.g. a function like $(\lambda x.\ x)$ has many types, all being instances of the type scheme $\alpha \rightarrow \alpha$). Only recently there have appeared several studies of combining subtyping with type inference (cf. [Wand 1987], [Fuh, Mishra 1988], and [Stansifer 1988]), but these studies all address only the syntactic aspects. Further investigation is required to determine whether our technique for a semantics of subtyping also applies in this case.

The remainder of this paper is organized as follows. In the next section we motivate and formally treat a language without subtyping. Then, in Section 3, we introduce subtyping and give the semantics $[\{\ \}]$ for types and express our intentions for the semantics $[\{\ \}]$ for expressions, (Definition 3.13). In Section 4 we define minimal typing, and define $[\{\ \}]$ for expressions and show that it indeed satisfies our intentions.

2 A language without subtyping

Our method of adapting a semantics $[\![\ ]\!]$ for a base language without subtyping to a semantics $[\{\ \}]$ for subtyping, seems to be largely independent of the particular choice of the base language. It would be nice if we could abstract away completely from the base language. However, in order to provide formal proofs, we have to make some choice or another.

In the choice of the base language we have been lead by the overview of Reynolds [Reynolds 1985]. He discusses typing in general and does so by considering a language that has

- unrestricted abstraction (i.e. functions)
- records (or tuples, both with named and unnamed components)
- discriminated unions (or variants)
- lists (homogeneous, possibly infinite)
- some basic data types, like integral and real numbers, truth values and so on
- the conditional (*if then else*) construct

- recursion

It turns out that not only functions give rise to semantical problems when subtyping is added (as we have shown in the introduction), but also records (as we will point out in Remark 3.8). Lists pose no semantical problems, and neither do variants and the conditional construct. However, these constructs do give rise to the notions of least upper bound ($\sqcup$) in the definition of minimal typing. So, in order to offer a sufficiently general treatment, we should take at least one of these constructs into consideration. We choose to leave out lists, the definitions for lists being the most straightforward. In order to save some space we only consider records with named components; records with named components are more interesting in the presence of subtyping than those with unnamed components, since - as Reynolds has pointed out - records with named components are better fit for allowing field-forgetting conversions of records and record types. We do not treat full, unrestricted recursion; it would complicate the semantics of the base language considerably, so that the gain of a simple, set-theoretic adaptation to subtyping is of lesser importance in this case.

In the remainder of this section we offer a formal treatment of the syntax and semantics of the base language.

2.1 Postulation Let B be a set (of *basic types*). Let $bool \in B$. As further examples one might think of basic types int and $real$. We let β vary over B.

2.2 Postulation Let L be a totally ordered set (of *labels*). We let a vary over L.

2.3 Remark We shall require, below, that $a < a'$ in a record type $\langle a : \sigma, a' : \tau \rangle$, thus enforcing a canonical form. In a concrete program representation, $\langle a : \sigma, a' : \tau \rangle$ might also be written as $\langle a' : \tau, a : \sigma \rangle$. Similarly for variant types.

2.4 Notational convention We abbreviate "$\langle a_1 : \tau_1, \ldots, a_m : \tau_m \rangle$" to "$\langle a_i : \tau_i \ (i \in m) \rangle$". That is to say, "$(i \in m)$" is a postfix qualification, meaning "for all i from 1 to m". The predicate "i is some value between, and including, 1 and m" is not abbreviated to $(i \in m)$ but to $1 \leq i \leq m$. The abbreviation is also used in other contexts.

2.5 Definition The set T (of *types*) is inductively defined as follows

1. $\beta \in T$, whenever $\beta \in B$
2. $(\sigma \to \tau) \in T$, whenever $\sigma, \tau \in T$
3. $\langle a_i : \tau_i \ (i \in m) \rangle \in T$, whenever $a_i \in L,\ \tau_i \in T \ (i \in m)$ and $a_1 < a_2 < \ldots < a_m$ and $m \geq 0$
4. $[a_i : \tau_i \ (i \in m)] \in T$, whenever $a_i \in L,\ \tau_i \in T \ (i \in m)$ and $a_1 < a_2 < \ldots < a_m$ and $m \geq 1$

We let ρ, σ, τ vary over T.

(Clause 2 defines function types, σ being the parameter type and τ being the result type. Clause 3 defines record types, the fields being labelled by $a_1, \ldots, a_m$. Clause 4 defines disjoint unions or variant types, the summands being tagged with labels $a_1, \ldots, a_m$. Even though allowing $m = 0$ in clause 4 would not give problems in Definitions 2.10 and 2.15, it would make Definition 2.19 problematic and Theorem 2.20 as well. But allowing for $m = 0$, however, *would* invalidate Theorem 4.7.2, and Theorem 4.8 can not even be formulated anymore (because the σ mentioned in this theorem need not exist).)

2.6 Postulation For each $\tau \in T$ let C_τ be a (possibly empty) set (of *constants*), mutually disjoint. We let c vary over C_τ. $C_{bool} = \{true, false\}$.

As further examples of constants one might think of $zero \in C_{int}$, $succ \in C_{int \to int}$, $null \in C_{real}$, $add1 \in C_{real \to real}$. To get "interesting" programs, there should be a primitive recursion construct $primrec \in C_{int \to (int \to int) \to int}$. All these constants get their meaning assigned in Postulation 2.17. Notice, by the way, that disjointness, here, means that there is no overloading (one symbol having several types, and therefore several meanings). One should not confuse disjointness of C_σ and C_τ with disjointness of $[\![\sigma]\!]$ and $[\![\tau]\!]$ (cf. Postulation 2.14).

2.7 Postulation For each $\tau \in T$ let X_τ be a set (of *variables*), mutually disjoint, countably infinite and disjoint from the sets C_σ $(\sigma \in T)$. We let x vary over X_τ.

2.8 Remark The postulation that *variables are typed* eliminates the need for introducing a type assignment (that assigns a type to variables), and therefore simplifies the presentation slightly.

2.9 Definition The set E (of *expressions*) is defined inductively as follows

1. $c \in E$, whenever $\tau \in T,\ c \in C_\tau$
2. $x \in E$, whenever $\tau \in T,\ x \in X_\tau$
3. $(\lambda x.e) \in E$, whenever $\sigma \in T,\ x \in X_\sigma,\ e \in E$
4. $e(e') \in E$, whenever $e, e' \in E$
5. $(\mathit{if}\ e\ \mathit{then}\ e'\ \mathit{else}\ e'') \in E$, whenever $e, e', e'' \in E$
6. $\langle a_i = e_i\ (i \in m)\rangle \in E$, whenever $a_i \in L,\ e_i \in E\ (i \in m)$ and $a_1 < a_2 < \ldots < a_m$ and $m \geq 0$
7. $e.a \in E$, whenever $e \in E,\ a \in L$
8. $[a = e] \in E$, whenever $e \in E,\ a \in L$
9. $(\mathit{case}\ e\ \mathit{of}\ a_1 : e_1, \ldots, a_m : e_m) \in E$, whenever $e, e_i \in E,\ a_i \in L\ (i \in m)$ and $a_1 < a_2 < \ldots < a_m$ and $m \geq 1$

We let e vary over E.

(Clause 3 defines function expressions, with parameter x and body e. Clause 4 defines function application, e being the function and e' the argument expression. Clause 6 defines a record expression, clause 7 a record selection. Clause 8 defines the expression for *injection* into some variant: e tagged with a as a member of some disjoint union. Clause 9 defines a case-selection: variant value e is untagged and then subject to function e_i if its tag was a_i. All the above-mentioned intended meanings of expressions are formalized in the semantics below, in 2.19.)

2.10 Definition The relation : on $E \times T$ ($e : \tau$ is pronounced as: *e is well-typed and has type τ*) is defined inductively as follows

1. $c : \tau$, whenever $c \in C_\tau$
2. $x : \tau$, whenever $x \in X_\tau$
3. $(\lambda x.e) : (\sigma \to \tau)$, whenever $x \in X_\sigma,\ e : \tau$
4. $e(e') : \tau$, whenever $e : (\sigma \to \tau),\ e' : \sigma$
5. $(\mathit{if}\ e\ \mathit{then}\ e'\ \mathit{else}\ e'') : \tau$, whenever $e : \mathit{bool},\ e' : \tau,\ e'' : \tau$

6. $\langle a_i = e_i \ (i \in m)\rangle : \langle a_i : \tau_i \ (i \in m)\rangle$, whenever $e_i : \tau_i \ \ (i \in m)$

7. $e.a : \tau$, whenever $e : \langle a_i : \tau_i \ (i \in m)\rangle, \ a = a_j, \ \tau = \tau_j$ for some $j \ \ (1 \le j \le m)$

8. $[a = e] : [a_i : \tau_i \ (i \in m)]$, whenever $a = a_j, \ e : \tau_j$ for some $j \ \ (1 \le j \le m)$

9. $(\mathit{case}\ e\ \mathit{of}\ a_1 : e_1, \ldots, a_m : e_m) : \tau$, whenever $e : [a_i : \tau_i \ (i \in m)], \ e_i : (\tau_i \to \tau) \ \ (i \in m)$

As an example, it is easy to verify that, for $x \in X_{int}$, $(\lambda x.\ succ(succ(x)))\ :\ (int \to int)$, and, for $x \in X_{real}$, $(\lambda x.\ add1(add1(x)))\ :\ (real \to real)$.

2.11 Lemma For any $e \in E, \ \tau \in T$, there is at most one way to derive $e : \tau$.
Proof Easy induction on the structure of e.

2.12 Remark It is not true that for any $e \in E$ there is at most one $\tau \in T$ for which $e : \tau$. The ambiguity in the type of e is entirely due to clause 8 of 2.10. Lemma 2.11 shows that a type *derivation* is not ambiguous. So we may formulate definitions by induction on the derivation of a typing $e : \tau$ as in 2.19 below.

2.13 Definition For $\tau \in T$ we define $E_\tau = \{e \in E \mid e : \tau\}$.

* * *

Now we turn to the semantics of types and expressions.

2.14 Postulation For $\beta \in B$ let $[\![\beta]\!]$ be a non-empty set. Let $[\![bool]\!] = \{tt, ff\}$ with $tt \neq ff$. (We do not require disjointness of the $[\![\beta]\!]$. For example, one could postulate $[\![int]\!] = \mathbf{Z}$, $[\![real]\!] = \mathbf{R}$, with, as usual, $\mathbf{Z} \subset \mathbf{R}$. However, it is also possible to postulate $[\![int]\!] = \langle + | - \rangle\langle\text{digit}\rangle^*$ and $[\![real]\!] = \langle + | - \rangle\langle\text{digit}\rangle^*\langle . \rangle\langle\text{digit}\rangle^*$, so that $[\![int]\!]$ and $[\![real]\!]$ are disjoint.)

2.15 Definition For each $\tau \in T$ a set $[\![\tau]\!]$ is defined by induction on the structure of τ as follows

1. $[\![\beta]\!]$ has been postulated in 2.14

2. $[\![(\sigma \to \tau)]\!] \ = \ [\![\sigma]\!] \to [\![\tau]\!] \ =$ the set of all total functions from $[\![\sigma]\!]$ to $[\![\tau]\!]$

3. $[\![\langle a_i : \tau_i \ (i \in m)\rangle]\!] \ =$ the set of total functions with domain $\{a_1, \ldots, a_m\}$ that map a_i into $[\![\tau_i]\!]$ for all $i \in m$. We shall denote such a function f by its "graph" $\{(a_1, d_1), \ldots, (a_m, d_m)\}$, or $\{(a_i, d_i) \ \mid \ i \in m\}$, meaning that $f(a_i) = d_i \ (i \in m)$

4. $[\![[a_i : \tau_i \ (i \in m)]]\!] \ = \ \{(a_i, d_i) \mid 1 \le i \le m \wedge d_i \in [\![\tau_i]\!]\}$

We let d vary over any $[\![\tau]\!]$.

2.16 Definition $U = \bigcup_{\tau \in T}[\![\tau]\!]$, the universe in which the semantics of both types and expressions shall find their place, both with and without subtyping.

2.17 Postulation For each $\tau \in T, \ c \in C_\tau$ let $[\![c]\!]$ be some member of $[\![\tau]\!]$.
Let $[\![true]\!] = tt, \ [\![false]\!] = ff$.
(Here the semantics $[\![zero]\!]$, $[\![succ]\!]$, $[\![null]\!]$, $[\![add1]\!]$, $[\![primrec]\!]$ have to be chosen in such a way that we get the intended respective meanings of these constants.)

2.18 Definition An *assignment* A is a family of functions $A_\tau \in X_\tau \to [\![\tau]\!], \ (\tau \in T)$. For assignment $A, \ \tau \in T, \ x \in X_\tau, \ d \in [\![\tau]\!]$ we define the assignment $A[x \mapsto d]$ for all $\sigma \in T, \ y \in X_\sigma$ by

$$\begin{array}{rclcl} (A[x \mapsto d])_\sigma(y) & = & A_\sigma(y) & \text{, if } \sigma \neq \tau \quad \text{or} \quad y \neq x \\ & = & d & \text{, if } \sigma = \tau \quad \text{and} \quad y = x \end{array}$$

2.19 Definition Let A be an assignment. Functions $[\![\ ]\!]^\tau_A \in E_\tau \to U$ are defined by induction on the derivation of their typing as follows

1. $[\![c]\!]^\tau_A \ = \ [\![c]\!]$ as postulated in 2.17, whenever $c \in C_\tau$
2. $[\![x]\!]^\tau_A \ = \ A_\tau(x)$, whenever $x \in X_\tau$
3. $[\![(\lambda x.e)]\!]^{\sigma \to \tau}_A \ = \ \lambda d \in [\![\sigma]\!].\ [\![e]\!]^\tau_{A[x \mapsto d]}$, whenever $x \in X_\sigma,\ e : \tau$.
   (On the right hand side we have used λ as a notation on the meta-level for functions.)
4. $[\![e(e')]\!]^\tau_A \ = \ f(d)$, where $f = [\![e]\!]^{\sigma \to \tau}_A,\ d = [\![e']\!]^\sigma_A$, whenever $e : (\sigma \to \tau),\ e' : \sigma$
5. $$\begin{array}{rcll} [\![\mathit{if}\ e\ \mathit{then}\ e'\ \mathit{else}\ e'']\!] & = & [\![e']\!]^\tau_A & \text{, if } [\![e]\!]^{\mathit{bool}}_A = \mathit{tt} \\ & = & [\![e'']\!]^\tau_A & \text{, if } [\![e]\!]^{\mathit{bool}}_A = \mathit{ff} \end{array}$$
   whenever $e : \mathit{bool},\ e' : \tau,\ e'' : \tau$
6. $[\![\langle a_i = e_i\ (i \in m)\rangle]\!]^{\langle a_i : \tau_i\ (i \in m)\rangle}_A \ = \ \{(a_i, [\![e_i]\!]^{\tau_i}_A) \ \mid\ i \in m\}$, whenever $e_i : \tau_i \ \ (i \in m)$
7. $[\![e.a]\!]^\tau_A \ = \ f(a)$, where $f = [\![e]\!]^{\langle a_i : \tau_i\ (i \in m)\rangle}_A$, whenever $e : \langle a_i : \tau_i\ (i \in m)\rangle,\ a = a_j,\ \tau = \tau_j$ for some $j,\ 1 \leq j \leq m$
8. $[\![[a = e]]\!]^{[a_i : \tau_i\ (i \in m)]}_A \ = \ (a, [\![e]\!]^{\tau_j}_A)$, whenever $a = a_j,\ e : \tau_j$ for some $j,\ 1 \leq j \leq m$
9. $$\begin{array}{rcll} [\![\mathit{case}\ e\ \mathit{of}\ a_1 : e_1, \ldots, a_m : e_m]\!]^\tau_A & = & [\![e_1]\!]^{\tau_1}_A(d) & \text{, if } a = a_1 \\ & \cdot & & \\ & \cdot & & \\ & = & [\![e_m]\!]^{\tau_m}_A(d) & \text{, if } a = a_m \end{array}$$
   (where $(a, d) = [\![e]\!]^{[a_i : \tau_i\ (i \in m)]}_A$), whenever $e : [a_i : \tau_i\ (i \in m)],\ e_i : (\tau_i \to \tau) \ \ (i \in m)$.

2.20 Theorem For each $\tau \in T,\ e \in E_\tau :\ \ [\![e]\!]^\tau_A \in [\![\tau]\!]$.
Proof Easy induction on the derivation of $e : \tau$.

2.21 Remark It is now standard practice to show that $[\![e]\!]^\tau_A = [\![e]\!]^\tau_{A'}$ if A and A' coincide on the free variables of e. Therefore, for closed e one may set $[\![e]\!]^\tau = [\![e]\!]^\tau_A$ for any A.

3 Adding subtyping

We speak of subtyping when there exists a partial order on types, and the typing rules are extended to the effect that

an expression e that has type σ may occur at a position where a supertype τ of σ is required,

or in other words

$$e : \tau \textit{ whenever } e : \sigma \textit{ and } \sigma \leq \tau.$$

This, however, is only a syntactic consequence of subtyping. Semantically the discipline of subtyping may be used

- to control the automatic insertion of fixed "conversion" functions $cv_{\sigma \leq \tau}$ at appropriate places
- to model the inheritance relation in (abstract) object-oriented languages (cf. [Cardelli 1984])
- to reflect syntactically (axiomatically) some semantic facts like $[\![int]\!] \subseteq [\![real]\!]$.

It happens that the first of these uses also covers the second and the third, by simply choosing some conversion functions to be the identity function. Reynolds [Reynolds 1985] gives a thorough syntactic treatment of subtyping with special attention to the first use above (but discusses the semantics only informally), and we shall follow him closely. We urge the reader to consult [Reynolds 1985] for more information.

3.1 Postulation Let $\leq_B$ be a relation on $B \times B$ and let, for each $\beta, \beta' \in B$ with $\beta \leq_B \beta'$, $cv_{\beta \leq \beta'}$ be a function in $[\![\beta]\!] \to [\![\beta']\!]$, such that the following properties hold true

$(\leq_B)$ $\leq_B$ is a partial order

(LUB_B) if two basic types have a common $\leq_B$- upper bound, then they have a $\leq_B$- least upper bound

(GLB_B) if two basic types have a common $\leq_B$- lower bound, then they have a $\leq_B$- greatest lower bound

(ID_B) $cv_{\beta \leq \beta} = identity_\beta \in [\![\beta]\!] \to [\![\beta]\!]$

(TR_B) $cv_{\beta' \leq \beta''} \circ cv_{\beta \leq \beta'} = cv_{\beta \leq \beta''}$, for $\beta \leq \beta' \leq \beta''$, where the operation $\circ$ denotes function composition: $(f \circ g)(x) = f(g(x))$.

As an example, whether $[\![int]\!] \subseteq [\![real]\!]$ actually holds or not, one may choose $int \leq real$, provided that $cv_{int \leq real}$ is defined as some function from $[\![int]\!]$ to $[\![real]\!]$ satisfying the requirements listed above.

3.2 Remark Cardelli [Cardelli 1984] models the inheritance relationship in object-oriented languages by means of subtyping, and then chooses $\leq_B$ to be the identity on basic types. This simplification does not simplify the theorems or proofs in an essential way.

3.3 Remark Another special case of the postulation above is the requirement that for $\beta \leq_B \beta'$ it holds that $[\![\beta]\!] \subseteq [\![\beta']\!]$. In this case we can *define* the conversion functions $cv_{\beta \leq \beta'}$ as identities, and we can *prove* (ID_B) and (TR_B). Again we have chosen the more general case above, because it does not complicate the forthcoming definitions and proofs.

3.4 Definition We define a relation $\leq$ on $T \times T$ and, simultaneously, for each pair $\sigma, \tau \in T$ with $\sigma \leq \tau$, a function $cv_{\sigma \leq \tau} \in [\![\sigma]\!] \to [\![\tau]\!]$, by induction as follows

1. if $\beta \leq_B \beta'$ then:
   - $\beta \leq \beta'$
   - $cv_{\beta \leq \beta'}$ is postulated in 3.1

2. let $\sigma = (\sigma_1 \rightarrow \sigma_2)$ and $\tau = (\tau_1 \rightarrow \tau_2)$; if $\tau_1 \leq \sigma_1$ and $\sigma_2 \leq \tau_2$ then:

   - $\sigma \leq \tau$
   - $cv_{\sigma \leq \tau}(f) \;=\; cv_{\sigma_2 \leq \tau_2} \circ f \circ cv_{\tau_1 \leq \sigma_1}$ for $f \in [\![\sigma_1 \rightarrow \sigma_2]\!]$

   (Note the monotonicity of $\leq$ in the result part and the anti-monotonicity in the parameter part.)

3. let $\sigma = \langle a_i : \sigma_i \ (i \in m)\rangle$ and $\tau = \langle a_{j_i} : \tau_{j_i} \ (i \in n)\rangle$; if $j_1, \ldots, j_n$ is a (not necessarily contiguous) sub-sequence of $1, \ldots, m$ and $\sigma_{j_i} \leq \tau_{j_i} \ \ (i \in n)$ then:

   - $\sigma \leq \tau$
   - $cv_{\sigma \leq \tau}(\{(a_i, d_i) \mid \ i \in m \ \}) = \{(a_{j_i}, cv_{\sigma_{j_i} \leq \tau_{j_i}}(d_{j_i})) \mid \ i \in n \ \}$

4. let $\sigma = [a_{j_i} : \sigma_{j_i} \ (i \in n)]$ and $\tau = [a_i : \tau_i \ (i \in m)]$; if $j_1, \ldots, j_n$ is a (not necessarily contiguous) sub-sequence of $1, \ldots, m$ and $\sigma_{j_i} \leq \tau_{j_i} \ \ (i \in n)$ then:

   - $\sigma \leq \tau$
   - $cv_{\sigma \leq \tau}((a_{j_i}, d)) = (a_{j_i}, cv_{\sigma_{j_i} \leq \tau_{j_i}}(d))$.

3.5 Remark

1. In [Reynolds 1985] a different definition is given of subtyping for record and variant types. This is done by splitting clauses 3 and 4 in Definition 3.4 in both cases into two separate sub-clauses. For example, in the case of record types, clause 3 is replaced by

   3a. $\sigma = \langle a_i : \sigma_i \ (i \in m)\rangle,\ \tau = \langle a_i : \tau_i \ (i \in m)\rangle,\ \sigma_i \leq \tau_i \ (i \in m) \ \Rightarrow \ \sigma \leq \tau$

   3b. $\sigma = \langle a_i : \sigma_i \ (i \in m)\rangle,\ \tau = \langle a_{j_i} : \sigma_{j_i} \ (i \in n)\rangle \ \Rightarrow \ \sigma \leq \tau$

   This alternative, as such, however, leads to the invalidness of the desired conclusion that $\leq$ constitutes a partial order (cf. Lemma 3.6), because transitivity of $\leq$ can not be proved anymore, *without* explicitly adding an extra clause to such a definition that any *combination* of the clauses mentioned also generates a pair of types belonging to the subtype relation (but this is just what the property of transitivity amounts to)! In such an alternative definition the steps are just too small to imply transitivity. (For example, $\langle a : \mathit{int},\ b : \mathit{bool}\rangle \leq \langle a : \mathit{real}\rangle$ can not be proved by either appealing to clause 3a or appealing to clause 3b, given that $\mathit{int} \leq \mathit{real}$, but it can be proved by appealing to our clause 3 in Definition 3.4.)

2. Any of the clauses -except for the first- in the preceding definition may be omitted without invalidating the lemmas and theorems to come. Actually, it is the very existence of a "natural" conversion function $cv \in [\![\sigma]\!] \rightarrow [\![\tau]\!]$ that allows, but does not force, to add the definitions $\sigma \leq \tau$ and $cv_{\sigma \leq \tau} = cv$. Here "natural" can be made precise: the addition of the clauses $\sigma \leq \tau$ and $cv_{\sigma \leq \tau} = cv$ should not invalidate the next lemma.

3.6 Lemma (cf. [Reynolds 1985]) The relation $\leq$ and functions $cv_{\sigma \leq \tau}$ (for $\sigma \leq \tau$) satisfy the following properties

$(\leq)$ $\quad$ $\leq$ is a partial order on $T \times T$

(LUB) if two types have a common $\leq$-upper bound, then they have a $\leq$-least upper bound

(GLB) if two types have a common $\leq$-lower bound, then they have a $\leq$-greatest lower bound

(ID) $cv_{\tau\leq\tau} \; = \; identity_\tau \; \in \; [\![\tau]\!] \to [\![\tau]\!]$

(TR) $cv_{\sigma\leq\tau} \circ cv_{\rho\leq\sigma} = cv_{\rho\leq\tau}$, for $\rho \leq \sigma \leq \tau$.

Proof

Case **(≤)** It is easily verified that any of the defining clauses for $\leq$ preserves the reflexivity, anti symmetry, and transitivity (of the initial partial order $\leq_B$)

Case **(LUB, GLB)** First we constructively define partial operations $\sqcup, \sqcap \in T \times T \hookrightarrow T$ that will yield the required least and greatest bounds:

- For $\beta, \beta' \in B$ that have a $\leq_B$-upper bound, we define $\beta \sqcup \beta'$ to be the $\leq_B$-lub that exists on account of postulation 3.1; analogously, for $\beta, \beta' \in B$ that have a $\leq_B$-lower bound we define $\beta \sqcap \beta'$ to be the $\leq_B$-glb that exists on account of 3.1
- For $\sigma = (\sigma_1 \to \sigma_2)$, $\tau = (\tau_1 \to \tau_2)$ for which $\sigma_1 \sqcup \tau_1$, $\sigma_1 \sqcap \tau_1$, $\sigma_2 \sqcup \tau_2$, $\sigma_2 \sqcap \tau_2$ exist, we define
$$\begin{aligned} \sigma \sqcup \tau &= (\sigma_1 \sqcap \tau_1) \to (\sigma_2 \sqcup \tau_2) \\ \sigma \sqcap \tau &= (\sigma_1 \sqcup \tau_1) \to (\sigma_2 \sqcap \tau_2) \end{aligned}$$
- For $\sigma = \langle a_i : \sigma_i \; (i \in m)\rangle$, $\tau = \langle b_j : \tau_j \; (j \in n)\rangle$ we define $\sigma \sqcup \tau$, $\sigma \sqcap \tau$ as follows. Let $c_1, \ldots, c_p$ be the ordered sequence of labels (*of minimal length*) containing exactly all a_i $(i \in m)$ and b_j $(j \in n)$. Furthermore let $d_1, \ldots, d_q$ be the (not necessarily contiguous) sub-sequence (*of maximal length*) of $c_1, \ldots, c_p$ that is a sub-sequence of both $a_1, \ldots, a_m$ and $b_1, \ldots, b_n$. Then
$\sigma \sqcup \tau = \langle d_l : \rho_l \; (l \in q)\rangle$, where $\rho_l = \sigma_i \sqcup \tau_j$, with i, j such that $a_i = d_l = b_j$
$\sigma \sqcap \tau = \langle c_k : \rho_k \; (k \in p)\rangle$, where
$$\rho_k = \begin{cases} \sigma_i & \text{, if } c_k = a_i \notin \{b_1, \ldots, b_n\} \\ \sigma_i \sqcap \tau_j & \text{, if } a_i = c_k = b_j \\ \tau_j & \text{, if } c_k = b_j \notin \{a_1, \ldots, a_m\} \end{cases}$$
where it is assumed that all the $\sigma_i \sqcup \tau_j$ and $\sigma_i \sqcap \tau_j$ occurring in the formulas above exist.
- For $\sigma = [a_i : \sigma_i \; (i \in m)]$, $\tau = [b_j : \tau_j \; (j \in n)]$ we define $\sigma \sqcup \tau$, $\sigma \sqcap \tau$ as follows. Let $c_1, \ldots, c_p$ and $d_1, \ldots, d_q$ be ordered sequences of labels as constructed above, then
$\sigma \sqcup \tau = [c_k : \rho_k \; (k \in p)]$, where
$$\rho_k = \begin{cases} \sigma_i & \text{, if } c_k = a_i \notin \{b_1, \ldots, b_n\} \\ \sigma_i \sqcup \tau_j & \text{, if } a_i = c_k = b_j \\ \tau_j & \text{, if } c_k = b_j \notin \{a_1, \ldots, a_m\} \end{cases}$$
$\sigma \sqcap \tau = [d_l : \rho_l \; (l \in q)]$, where $\rho_l = \sigma_i \sqcap \tau_j$, with i, j such that $a_i = d_l = b_j$.
It is furthermore assumed that all the $\sigma_i \sqcup \tau_j$ and $\sigma_i \sqcap \tau_j$ occurring in the formulas above exist.

Now it is easy to prove, for arbitrary ρ, σ, τ :

$$\begin{aligned} \rho \leq \tau \;\; \wedge \;\; \sigma \leq \tau \;\; &\Longrightarrow \;\; \rho, \sigma \leq \rho \sqcup \sigma \text{ (exists!) } \leq \tau \\ \rho \leq \sigma \;\; \wedge \;\; \rho \leq \tau \;\; &\Longrightarrow \;\; \rho \leq \sigma \sqcap \tau \text{ (exists!) } \leq \sigma, \tau \end{aligned}$$

by induction on the derivation of $\rho \leq \tau$ and using the following fact:

for arbitrary σ and τ,

$\sigma \leq \tau$ implies

either: σ and τ are both basic types and $\sigma \leq_B \tau$

or: $\sigma = (\sigma_1 \rightarrow \sigma_2)$, $\tau = (\tau_1 \rightarrow \tau_2)$ and $\tau_1 \leq \sigma_1$ and $\sigma_2 \leq \tau_2$

or: $\sigma = \langle a_i : \sigma_i \ (i \in m)\rangle$, $\tau = \langle a_{j_i} : \tau_{j_i} \ (i \in n)\rangle$ and $j_1, \ldots, j_n$ is a sub-sequence of $1, \ldots, m$ and $\sigma_{j_i} \leq \tau_{j_i}$ $(i \in n)$

or: $\sigma = [a_{j_i} : \sigma_{j_i} \ (i \in n)]$, $\tau = [a_i : \tau_i \ (i \in m)]$ and $j_1, \ldots, j_n$ is a sub-sequence of $1, \ldots, m$ and $\sigma_{j_i} \leq \tau_{j_i}$ $(i \in n)$

This fact can be proved by induction on the derivation of $\sigma \leq \tau$.

Case **(ID, TR)** These cases are easily proved by induction on the derivation of the subtype relation, in case (TR) using again the above fact.

3.7 Remark None of the properties (LUB), (GLB), (ID) or (TR) is used in the present section: the definition of the syntax and semantics of the language with subtyping. Properties (LUB) and (GLB), as well as the operations $\sqcup$ and $\sqcap$, are needed to define a minimal typing and to prove the soundness and completeness, in 4.2 and 4.7 below. Properties (ID) and (TR) are then used to complete the proof of the well-formedness of the semantics, in 4.8.

3.8 Remark It was the main intention of this paper to show that subtyping should , somehow, imply set inclusion; however, for the semantics for types defined thus far this is not yet the case – i.e., $\sigma \leq \tau$ does not imply $[\![\sigma]\!] \subseteq [\![\tau]\!]$, for arbitrary types σ, τ. For example, take $\sigma = \langle a : int,\ b : real\rangle$ and $\tau = \langle a : int\rangle$ (the reader can easily verify that in this case $\sigma \leq \tau$ holds, but *not* $[\![\sigma]\!] \subseteq [\![\tau]\!]$). Also we can take (cf. Section 1) $\sigma = (real \rightarrow int)$ and $\tau = (int \rightarrow int)$ to yield a contradiction for the statement $\sigma \leq \tau \Rightarrow [\![\sigma]\!] \subseteq [\![\tau]\!]$. This motivates to define a new semantics, written $[\!\{\ \}\!]$.

3.9 Definition For $\tau \in T$ we define

$$[\!\{\tau\}\!] = \bigcup_{\sigma \leq \tau} [\![\sigma]\!]$$

3.10 Theorem For $\sigma, \tau \in T$: $\sigma \leq \tau \Rightarrow [\!\{\sigma\}\!] \subseteq [\!\{\tau\}\!]$.
Proof

$$\begin{array}{rcll} [\!\{\sigma\}\!] & = & \bigcup_{\rho \leq \sigma} [\![\rho]\!] & \\ & \subseteq & \bigcup_{\rho \leq \tau} [\![\rho]\!] & \text{(by transitivity of } \leq \text{ and } \sigma \leq \tau) \\ & = & [\!\{\tau\}\!] \ . & \end{array}$$

3.11 Definition A new relation : on $E \times T$ is defined inductively as follows

1. – **9.** as for the old relation : in Definition 2.10

10. $e : \tau$, whenever $e : \sigma$ and $\sigma \leq \tau$.

3.12 Remark The new relation : is an extension of the old relation. Note that due to clause 10 an expression may have several types ($e : \sigma$ and $e : \tau$ for distinct σ, τ) and that a typing $e : \tau$ may have several derivations.

3.13 Definition Let A be an assignment. Functions $[\!\{\ \}\!]^{\tau}_{A} \in E_\tau \rightarrow U$ are defined by induction on the derivation of the argument's type

1. - 9. as for the functions $[\![\ ]\!]^{\tau}_{A} \in E_{\tau} \to U$ in Definition 2.19 (replacing $[\![\ ]\!]$ by $[\{\ \}]$)

10. $\quad [\{e\}]^{\tau}_{A} \;=\; cv_{\sigma\leq\tau}([\{e\}]^{\sigma}_{A})$ whenever $e : \sigma$ and $\sigma \leq \tau$.

3.14 Remark For given e and τ there may exist several distinct derivations of $e : \tau$ and therefore we have to show that this *syntactic ambiguity does not lead to semantic ambiguity.* In principle, we cannot claim that Definition 3.13 defines functions $[\{\ \}]^{\tau}_{A}$ but only relations "$[\{\ \}]^{\tau}_{A} \;= \ldots$"; we are faced with the problem to prove directly that the relations are functions, i.e.

$$[\{e\}]^{\tau}_{A} = d \;\wedge\; [\{e\}]^{\tau}_{A} = d' \;\Rightarrow\; d = d'$$

Another problem is that functions $[\{\lambda x.e\}]^{\sigma\to\tau}_{A}$ become -in principle- nondeterministic and, compared to Definition 2.19, the structure of the universe changes drastically. We are faced with some serious technical problems here. The next section is devoted to their solution.

4 Minimal typing

In this section we give another system for the language with subtyping, that in view of Theorem 4.7 is called a system with *minimal typing*. Minimal typing turns out to be sound and complete with respect to the typing in Section 3. A minimal type of an expression can be derived in at most one way, ensuring that we can safely base a definition of a semantics $[\![\ ]\!]^{*}_{A}$ on the derivation of an expression's minimal type, like in Definition 2.19. In terms of $[\![\ ]\!]^{*}_{A}$ we can express the unique solution of the equations for $[\{\ \}]^{*}_{A}$ in Definition 3.13.

4.1 Definition A partial operation $\sqcup \in T \times T \hookrightarrow T$ is defined as follows. For $\sigma, \tau \in T$ that have a common $\leq$-upper bound,
$\sigma \sqcup \tau \;=$ the $\leq$-least upper bound (that exists on account of (LUB) in Lemma 3.6).

4.2 Definition The relation $::$ on $E \times T$ ($e :: \tau$ is pronounced as τ *is the minimal type of* e) is defined inductively as follows

1. $c :: \tau$, whenever $c \in C_{\tau}$
2. $x :: \tau$, whenever $x \in X_{\tau}$
3. $(\lambda x.e) :: (\sigma \to \tau)$, whenever $x \in X_{\sigma},\; e :: \tau$
4. $e(e') :: \tau$, whenever $e :: (\sigma \to \tau),\; e' :: \sigma'$ and $\sigma' \leq \sigma$
5. $(\textit{if } e \textit{ then } e' \textit{ else } e'') :: \tau$, whenever $e :: \rho,\; \rho \leq \textit{bool},\; e' :: \sigma',\; e'' :: \sigma''$ and $\tau = \sigma' \sqcup \sigma''$ (and exists)
6. $\langle a_i = e_i\ (i \in m)\rangle :: \langle a_i : \tau_i\ (i \in m)\rangle$, whenever $e_i :: \tau_i\ (i \in m)$
7. $e.a :: \tau$, whenever $e :: \langle a_i : \tau_i\ (i \in m)\rangle,\; a = a_j,\; \tau = \tau_j$ for some j, $1 \leq j \leq m$
8. $[a = e] :: [a : \tau]$, whenever $e :: \tau$
9. $(\textit{case } e \textit{ of } a_1 : e_1, \ldots, a_m : e_m) :: \tau$, whenever $e :: \sigma,\; \sigma \leq [a_i : \sigma_i\ (i \in m)]$,
   $e_i :: (\sigma_i \to \tau_i)\ (i \in m),\; \tau = \tau_1 \sqcup \ldots \sqcup \tau_m$ (and exists)

We say that e is *minimally typable* if $e :: \tau$ for some $\tau \in T$.

(Notice, in advance, that, by Theorem 4.7.2, every typable expression has a minimal type.)

4.3 Lemma For any $e \in E$ there is at most one $\tau \in T$ such that $e :: \tau$ and there is at most one derivation of $e :: \tau$.

Proof Easy by induction on the structure of e.

4.4 Definition Let A be an assignment. A partial function $[\![\ ]\!]^*_A \in E \hookrightarrow U$ is defined, for minimally typable expressions, as follows by induction on the derivation of the minimal type of its argument

1. $[\![c]\!]^*_A = [\![c]\!]$ as postulated in 2.17, whenever $c \in C_\tau$
2. $[\![x]\!]^*_A = A_\tau(x)$, whenever $x \in X_\tau$
3. $[\![\lambda x.e]\!]^*_A = \lambda d \in [\![\sigma]\!].[\![e]\!]^*_{A[x\mapsto d]}$, whenever $x \in X_\sigma,\ e :: \tau$
4. $[\![e(e')]\!]^*_A = f(cv_{\sigma' \leq \sigma}(d))$, where $f = [\![e]\!]^*_A,\ d = [\![e']\!]^*_A$, whenever $e :: (\sigma \to \tau),\ e' :: \sigma'$ and $\sigma' \leq \sigma$
5. $$\begin{array}{lcll} [\![\mathit{if}\ e\ \mathit{then}\ e'\ \mathit{else}\ e'']\!]^*_A & = & cv_{\sigma' \leq \tau}([\![e']\!]^*_A) & \text{, if } d = tt \\ & = & cv_{\sigma'' \leq \tau}([\![e'']\!]^*_A) & \text{, if } d = ff \end{array}$$
   (where $d = cv_{\rho \leq bool}([\![e]\!]^*_A))$, whenever $e :: \rho,\ \rho \leq bool,\ e' :: \sigma',\ e'' :: \sigma'',\ \tau = \sigma' \sqcup \sigma''$ (and exists)
6. $[\![\langle a_i = e_i\ (i \in m)\rangle]\!]^*_A\ =\ \{(a_i, [\![e_i]\!]^*_A) \mid i \in m\}$, whenever $e_i :: \tau_i\ (i \in m)$
7. $[\![e.a]\!]^*_A\ =\ f(a)$, where $f = [\![e]\!]^*_A$, whenever $e :: \langle a_i : \tau_i\ (i \in m)\rangle,\ a = a_j$ for some $j,\ 1 \leq j \leq m$
8. $[\![[a = e]]\!]^*_A\ =\ (a, [\![e]\!]^*_A)$, whenever $e :: \tau$
9. $$\begin{array}{lcll} [\![\mathit{case}\ e\ \mathit{of}\ a_1 : e_1, \ldots, a_m : e_m]\!]^*_A & = & cv_{\tau_1 \leq \tau}([\![e_1]\!]^*_A(d)) & \text{, if } a = a_1 \\ & \cdot & & \\ & \cdot & & \\ & = & cv_{\tau_m \leq \tau}([\![e_m]\!]^*_A(d)) & \text{, if } a = a_m \end{array}$$
   (where $(a, d) = cv_{\sigma \leq [a_i : \sigma_i\ (i \in m)]}([\![e]\!]^*_A))$, whenever $e :: \sigma,\ \sigma \leq [a_i : \sigma_i\ (i \in m)]$, $e_i :: (\sigma_i \to \tau_i)\ (i \in m),\ \tau = \tau_1 \sqcup \ldots \sqcup \tau_m$ (and exists).

4.5 Theorem For $e \in E,\ \tau \in T$:

$$e :: \tau\ \Rightarrow\ [\![e]\!]^*_A \in [\![\tau]\!]$$

Proof Easy induction on the derivation of $e :: \tau$.

4.6 Corollary For $e \in E,\ \tau \in T$:

$$e :: \tau \Rightarrow [\![e]\!]^*_A \in [\{\tau\}].$$

Thus we have succeeded in designing semantics $[\![\ ]\!]^*_A$ and $[\{\ \}]$ such that

- $\sigma \leq \tau \Rightarrow [\{\sigma\}] \subseteq [\{\tau\}]$, and
- $e :: \tau \Rightarrow [\![e]\!]^*_A \in [\{\tau\}]$

However, the well-formedness of Definition 3.13 has yet to be shown.

4.7 Theorem [Reynolds 1985]

1. (soundness) $e :: \sigma \;\Rightarrow\; e : \sigma$
2. (completeness) $e : \tau \;\Rightarrow\; e :: \sigma$, for some $\sigma \in T$
3. (minimality) $e : \tau \wedge e :: \sigma \;\Rightarrow\; \sigma \leq \tau$

Proof

1. Easy induction on the derivation of $e :: \sigma$, using the fact that if $\tau = \rho \sqcup \sigma$ exists, then $\rho \leq \tau$ and $\sigma \leq \tau$ (from (LUB)).

2,3. These are proved simultaneously, i.e.

$$e : \tau \;\Rightarrow\; e :: \sigma \quad , \text{ for some } \sigma \leq \tau$$

by induction on some (any!) derivation of $e : \tau$ using property (LUB) and transitivity of $\leq$.

4.8 Theorem Definition 3.13 defines functions $[\![\{\ \}]\!]_A^\tau \in E_\tau \to [\![\{\tau\}]\!]$ given by

$$[\![\{e\}]\!]_A^\tau \;=\; cv_{\sigma \leq \tau}([\![e]\!]_A^*)$$

where σ is the existent and unique type such that $e :: \sigma$ and $\sigma \leq \tau$ (by Theorem 4.7).

Proof It is rather simple to show by straightforward reasoning (no induction required) that the functions $cv_{\sigma \leq \tau} \circ [\![\ ]\!]_A^*$ (where σ is the minimal type of the argument $\in E_\tau$) satisfy each of the equations in Definition 3.13. Here properties (ID) and (TR) of Lemma 3.6 are used. On the other hand, for any function $[\![\{\ \}]\!]_A^\tau$ satisfying equations 1-10 of 3.13, it easily follows by induction on the structure of e that $[\![\{e\}]\!]_A^\tau = cv_{\sigma \leq \tau}([\![e]\!]_A^*)$, for all $e \in E_\tau$, where σ is the minimal type of e. Here again properties (ID) and (TR) are of importance.

4.9 Remark In retrospect, the results achieved in this paper can be summarized in a nut shell by the following equation

$$[\![\{e\}]\!]_A^\sigma = [\![e]\!]_A^* \in [\![\sigma]\!] \subseteq \bigcup_{\rho \leq \sigma} [\![\rho]\!] = [\![\{\sigma\}]\!] \subseteq [\![\{\tau\}]\!]$$

for $e :: \sigma \leq \tau$.

Acknowledgement We thank Jan Kuper for fruitful discussions.

REFERENCES

[Bruce, Wegner 1987]
Bruce, K. and Wegner, P.: "An Algebraic Model of Subtype and Inheritance"; in: Proceedings of the Workshop on Database Programming Languages; Roscoff, France, 1987, pp.107 -132.

[Cardelli 1984]
Cardelli, L.: "A Semantics of Multiple Inheritance"; in: "Semantics of Data Types" (eds. Kahn, MacQueen and Plotkin), Lecture Notes in Computer Science 173, Springer Verlag 1984, pp. 51-68.

[Cardelli, Wegner 1985]
Cardelli, L. and Wegner, P.: "On understanding types, data abstractions and polymorphism". Comp. Surveys 17, pp. 471-522, 1985.

[Fokkinga 1981]
Fokkinga, M.M.: "On the notion of strong typing"; in: Algorithmic Languages (eds. de Bakker, van Vliet), North Holland, Amsterdam, 1981, pp. 305-320.

[Fokkinga 1987]
Fokkinga, M.M.: "Programming Language Concepts – the Lambda Calculus Approach"; in: Essays on Concepts, Formalisms, and Tools (eds. Asveld, Nijholt), CWI Tract 42, CWI, Amsterdam, 1987, pp. 129-162.

[Fuh, Mishra 1988]
Fuh, Y.C., Mishra, P.: "Type Inference with Subtypes"; in: Proceedings 2^{nd} European Symposium on Programming (ESOP 88) (ed. H. Ganzinger), Lecture Notes in Computer Science 300, 1988, pp.94-114.

[Gries 1978]
Gries, D.: "Programming Methodology- a collection of articles by members if IFIP WG 2.3", Springer Verlag, 1978.

[Hindley, Seldin 1986]
Hindley, J.R, Seldin, J.P.: "Introduction to Combinators and Lambda Calculus", London Mathematical Society Student Texts 1, Cambridge University Press, Cambridge (U.K.), 1986.

[MacQueen et al 1984]
MacQueen, D.B., Seti, R., Plotkin, G.D.: "An Ideal Model for Recursive Polymorphic Types"; in: Conference record of the 11^{th} annual ACM Symposium on Principles of Programming Languages (11^{th} POPL), ACM, 1984, pp. 165-175.

[Reynolds 1984]
Reynolds, J.C.: "Polymorphism is not Set-Theoretic"; in: Semantics of Data Types (eds. Kahn, MacQueen, Plotkin), Lecture Notes in Computer Science 173, Springer Verlag 1984, pp. 145-156.

[Reynolds 1985]
Reynolds, J.C.: "Three Approaches to Type Structure"; in "Mathematical Foundations of Software Development"(eds. Ehrig e.a.), Lecture Notes in Computer Science 185, Springer Verlag 1985, pp. 97-138.

[Stansifer 1988]
Stansifer, R.: "Type Inference with Subtypes"; in: Conference Record of the 15^{th} Annual ACM Symposium on Principles of Programming Languages (POPL 88), 1988, pp.88-97.

[Wand 1987]
Wand, M.: "Complete Type Inference for Simple Objects"; in: Proceedings 2^{nd} Annual Symposium on Logic in Computer Science (LICS 87), 1987, pp. 37-44.