LFCS

Laboratory for Foundations of Computer Science
Department of Computer Science - University of Edinburgh

Co-induction in Relational Semantics

by

Robin Milner and Mads Tofte

Co-induction in Relational Semantics

LFCS Report Series

ECS-LFCS-88-65
(also published as CSR-278-88)

OCTOBER 1988

LFCS
Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ

Co-induction in Relational Semantics

Robin Milner
Mads Tofte
Laboratory for Foundations of Computer Science
Department of Computer Science
University of Edinburgh
Edinburgh EH9 3JZ, U.K.

October 6, 1988

Abstract: An application of the mathematical theory of maximum fixed points of monotonic set operators to relational semantics is presented. It is shown how an important proof method, called *co-induction*, can be used to prove the consistency of the static and the dynamic relational semantics of a small functional programming language with recursive functions.

1 Introduction

The purpose of this paper is to present one instance among several we have encountered where the use of non-well-founded sets, maximum fixed points of monotonic operators and a proof method, which we call *co-induction*, are essential tools in studying the semantics of programming languages.

A set A is *non-well-founded* if there is an infinite sequence $A_1, A_2, \ldots$ such that A_{n+1} is a member of A_n, for all $n \geq 1$. Otherwise it is said to be *well-founded*. Although it is often assumed in set theory that all sets are well-founded, Aczel's anti-foundation axiom [2] leads to an alternative set theory which is very useful in computer science. The significance of maximum fixed points and non-well-founded relations in connection with concurrency has been demonstrated in work by (among others) Park [8, 9] and Milner [7]

Non-well-founded objects occur naturally in programming language semantics. The example we present in this paper is the soundness of a type inference system with respect to the dynamic relational semantics of a little functional programming language. The language is essentially the lambda calculus enriched with an explicit construction for recursive functions. In the dynamic relational semantics

all functions evaluate to *closures* of the form $\langle x, exp, E\rangle$, where x is the formal parameter of the function, *exp* is the body of the function and E is an environment containing bindings for the variables that occur free in *exp*. If the closure is the value of a recursive function, then E should map the name of the function to the entire closure itself. For example, the evaluation of the expression

$$\texttt{fix factorial(n)= .... factorial(pred n)} \tag{1}$$

in the empty environment should yield a closure satisfying

$$cl_{\text{fact}} = \Big\langle \texttt{n}\,,\, \texttt{...factorial(pred n)}\,,\, \{\texttt{factorial} \mapsto cl_{\text{fact}}\} \Big\rangle. \tag{2}$$

By encoding tuples and finite maps as sets, one can view a solution to (2) as a non-well-founded set. Alternatively, one can consider non-well-foundedness of other objects than sets with respect to other relations than membership. In our case, for every n-tuple $\langle x_1, \ldots, x_n\rangle$ let us say that each x_i is a *constituent* of the tuple and for every finite map $\{x_1 \mapsto y_1, \ldots, x_n \mapsto y_n\}$ let us say that each y_i is a *constituent* of the map. Let us write $x \succ y$ to mean that y is a constituent of x. A *constituent sequence* is a finite or denumerable sequence of objects such that if y is the successor of x in the sequence then $x \succ y$. If we broaden the notion of membership to constituentship in this way, then it is quite natural to call an object *non-well-founded* when it occurs in some infinite constituent sequence. Note that any cl_{fact} satisfying (2) is non-well-founded as it occurs in the infinite (periodic) constituent chain

$$cl_{\text{fact}} \succ \{\texttt{factorial} \mapsto cl_{\text{fact}}\} \succ cl_{\text{fact}} \succ \{\texttt{factorial} \mapsto cl_{\text{fact}}\} \succ \ldots$$

Whereas structural induction is a powerful technique for proving properties of well-founded objects, *co-induction* may be used for proving properties of non-well-founded objects. Co-induction is not a new mathematical tool, but it is perhaps not as well-known as it deserves. We hope that the proofs we present in this paper will stimulate the awareness of the underlying theory.

The reader may suggest, at this point, that there is no need to take a closure to be a non-well-founded object, since one can deal instead with perfectly well-founded objects — namely a finite expression, formed by a recursion operator, which represents the infinite unfolding of the closure. This can be done, but it does not remove the need for co-inductive proof; indeed, we have pursued this approach and have found that the proof presented here requires only minor modifications. We prefer to deal with closures as non-well-founded objects because it appears most natural to do so.

Relational semantics borrows the idea of inference rule from formal logic to define the semantics of programming languages. It derives from Plotkin's work on "structural operational semantics"[10]. Kahn and his group use the term "natural semantics" [5] for what we call relational semantics. Whatever the name, it is

of a more syntactical and mechanical nature than denotational semantics, where programs are mapped to objects (so-called denotations) in a mathematical model. Using denotational semantics it has been proved that the type inference system we define below is sound, i.e. that if an expression *exp* has a type τ according to the type inference system and d is the denotation of *exp* then d is a member of the set (actually an ideal) which models the type τ (see e.g.[6]). In this paper we shall prove the corresponding result for relational semantics. This gives us the opportunity to review and apply the principle of co-induction, without which we have not been able to prove the consistency of the type inference system and the dynamic semantics.

The rest of this paper is organised as follows. In Section 2 we define the syntax and the dynamic semantics of the language; in Sections 3 we define the statics semantics of the language; in Section 4 we introduce the idea of maximum fixed points and co-induction and in Section 5 we use it to prove the consistency of the static and the dynamic semantics. In Section 6 we finally discuss alternative notions of what it is for a value to have a type, some of which take one beyond the realm of co-induction, since this concept depends on the monotonicity of functions.

The reader is supposed to know elementary set theory; the basic ideas in relational semantics are simple and will be introduced when they are used. In order to allow the reader to concentrate on the basic proof method, we have chosen to state and prove a relatively elementary theorem.

2 The Language and its Dynamic Semantics

For the definition of the language we assume a set Const of constants, ranged over by c, and a set Var of variables, ranged over by x and f. The language Exp of expressions is

exp	::=	x	Variable
		c	Constant
		fn x => *exp*	Abstraction
		fix $f(x)$ = *exp*	Recursive function
		$exp_1\ exp_2$	Application.

The abstraction **fn** x => *exp* corresponds to lambda abstraction in the lambda calculus. In **fix** $f(x)$ = *exp*, the function f is defined recursively.

In what follows, we use $\uplus$ to mean disjoint union of sets and $A \xrightarrow{\text{fin}} B$ denotes the partial functions from the set A to B that have a finite domain. If $f \in A \xrightarrow{\text{fin}} B$ the domain and range of f are denoted by $\mathrm{Dom}(f)$ and $\mathrm{Rng}(f)$, respectively. Every finite map $f \in A \xrightarrow{\text{fin}} B$ can be written on the form $\{a_1 \mapsto b_1, \ldots, a_n \mapsto b_n\}$; in particular $\{\}$ means the empty map. For every $f, g \in A \xrightarrow{\text{fin}} B$ the map $f + g \in A \xrightarrow{\text{fin}} B$, called f *modified by* g is the finite map with domain $\mathrm{Dom}(f) \cup \mathrm{Dom}(g)$

and values $(f+g)(a) = g(a)$, if $a \in \mathrm{Dom}(g)$, and $(f+g)(a) = f(a)$ otherwise.

$$\frac{}{E \vdash c \longrightarrow c} \tag{3}$$

$$\frac{x \in \mathrm{Dom}\, E}{E \vdash x \longrightarrow E(x)} \tag{4}$$

$$\frac{}{E \vdash \mathbf{fn}\ x\ \texttt{=>}\ exp \longrightarrow \langle x, exp, E\rangle} \tag{5}$$

$$\frac{cl_\infty = \langle x, exp, E + \{f \mapsto cl_\infty\}\rangle}{E \vdash \mathbf{fix}\ f(x)\ \texttt{=}\ exp \longrightarrow cl_\infty} \tag{6}$$

$$\frac{E \vdash exp_1 \longrightarrow c_1 \quad E \vdash exp_2 \longrightarrow c_2 \quad c = \mathrm{APPLY}(c_1, c_2)}{E \vdash exp_1\ exp_2 \longrightarrow c} \tag{7}$$

$$\frac{\begin{array}{rcl} E & \vdash & exp_1 \longrightarrow \langle x', exp', E'\rangle \\ E & \vdash & exp_2 \longrightarrow v_2 \\ E' + \{x' \mapsto v_2\} & \vdash & exp' \longrightarrow v \end{array}}{E \vdash exp_1\ exp_2 \longrightarrow v} \tag{8}$$

Figure 1: Dynamic Semantics

We now give a relational semantics for Exp in the form of a set of inference rules the conclusions of which are of the form $E \vdash exp \longrightarrow v$, read "*exp evaluates to v in E*". To handle recursion we allow our semantic objects to be non-well-founded. More precisely, it is possible to define sets Val, Clos, and Env so that they satisfy the set equations

$$\begin{array}{rcll} v \in \mathrm{Val} & = & \mathrm{Const} \uplus \mathrm{Clos} & \text{Values} \\ E \in \mathrm{Env} & = & \mathrm{Var} \xrightarrow{\text{fin}} \mathrm{Val} & \text{Environments} \\ cl \text{ or } \langle x, exp, E\rangle \in \mathrm{Clos} & = & \mathrm{Var} \times \mathrm{Exp} \times \mathrm{Env} & \text{Closures} \end{array}$$

and so that for all f, x, exp, E there is a unique closure $cl_\infty \in \mathrm{Clos}$ solving the equation

$$cl_\infty = \langle x, exp, E + \{f \mapsto cl_\infty\}\rangle. \tag{9}$$

Mathematical justification that it really is possible to find sets Val, Env and Clos which meet these requirements (in particular the requirement that (9) have one

and only one solution for cl_∞) can be found in Aczel's book [2]. Intuitively, the solution to (9) can be understood as the non-well-founded object which results from repeated application of (9) as a rewriting rule. Note that, because Clos = Var × Exp × Env, every closure cl in Clos is a triple and if $\langle x_1, exp_1, E_1\rangle = \langle x_2, exp_2, E_2\rangle$ then $x_1 = x_2$, $exp_1 = exp_2$ and $E_1 = E_2$.

A closure is the value of an abstraction. As in the literature, a closure takes the form $\langle x, exp, E\rangle$ where x is the formal parameter, exp the body of the function and E an environment which maps each free variable of exp to the value it assumes at the time of the declaration of the function. In general, evaluation of a fix expression yields a non-well-founded closure as illustrated with the `factorial` example in the Introduction.

To handle the application of constants to values we assume a partial function APPLY : Const × Const → Const. With these conventions we define the dynamic semantics of Exp by the inference rules in Figure 1; the rules allow us to infer statements of the form $E \vdash exp \longrightarrow v$, read *exp evaluates to v (in E)*. For instance, rule 7 can be read: if exp_1 evaluates to c_1 and exp_2 evaluates to c_2 and $c = \text{APPLY}(c_1, c_2)$ then $exp_1\ exp_2$ evaluates to c.

3 Static Semantics

The static semantics of Exp is defined by an inference system, more precisely a simple monomorphic type inference system, as follows.

The set Type of *type expressions* (just called types in the following), ranged over by τ, is defined by

$$\tau ::= \pi \mid \tau_1 \to \tau_2,$$

where π ranges over a set of primitive types, e.g., *int* and *bool*. A *type environment* is a finite map from variables to types:

$$TE \in \text{TyEnv} = \text{Var} \xrightarrow{\text{fin}} \text{Type}.$$

We assume a basic relation IsOf $\subseteq$ Const × Type relating for instance 3 to *int*, *true* to *bool* but not 3 to *bool*, and we require that whenever $c = \text{APPLY}(c_1, c_2)$ and c_1 IsOf $(\tau_1 \to \tau_2)$ and c_2 IsOf τ_1 then c IsOf τ_2. The inference rules appear in Figure 2; they allow one to infer statements of the form $TE \vdash exp \Longrightarrow \tau$, read *exp elaborates to τ (in TE)*.

$$\frac{c \text{ IsOf } \tau}{TE \vdash c \Longrightarrow \tau} \tag{10}$$

$$\frac{x \in \text{Dom}\, TE}{TE \vdash x \Longrightarrow TE(x)} \tag{11}$$

$$\frac{TE + \{x \mapsto \tau_1\} \vdash exp \Longrightarrow \tau_2}{TE \vdash \textbf{fn } x \texttt{ => } exp \Longrightarrow \tau_1 \to \tau_2} \tag{12}$$

$$\frac{TE + \{f \mapsto \tau_1 \to \tau_2\} + \{x \mapsto \tau_1\} \vdash exp \Longrightarrow \tau_2}{TE \vdash \textbf{fix } f(x) \texttt{ = } exp \Longrightarrow \tau_1 \to \tau_2} \tag{13}$$

$$\frac{TE \vdash exp_1 \Longrightarrow \tau_1 \to \tau_2 \quad TE \vdash exp_2 \Longrightarrow \tau_1}{TE \vdash exp_1\ exp_2 \Longrightarrow \tau_2} \tag{14}$$

Figure 2: Static Semantics

4 Typing Values Using Maximum Fixed Points and Co-induction

By pointwise extension of the relation IsOf $\subseteq$ Const $\times$ Type we get a relation IsOf $\subseteq$ Env $\times$ TyEnv. We expect it to be the case, then, that if *exp* elaborates to τ in TE and and *exp* evaluates to c in E and E IsOf TE then c IsOf τ. We refer to this proposition as *the basic consistency of the static and the dynamic semantics.*

However, this proposition needs strengthening before it can be proved by induction, the reason being that evaluation resulting in constants may require evaluation resulting in closures (about which the basic consistency says nothing). Below, we first extend the relation c IsOf τ to a relation $v : \tau$, read v *has type* τ, which also says what it is for a closure to have a type; then we define the relation $E : TE$, read E *matches* TE, to be the pointwise extension of $v : \tau$ and prove that if *exp* elaborates to τ in TE and *exp* evaluates to v in E and E matches TE then v has type τ. There are variants to the $v : \tau$ relation which also are plausible definitions of what it is for a value to have a type. We shall call these relations collectively *correspondence relations*, because each of them defines a correspondence between the dynamic semantics (values) and the static semantics (types).

The consistency proof is relatively simple if we can define the correspondence

relation so that it satisfies

$$v : \tau \quad \textit{iff} \quad \begin{cases} (i) & \text{if } v = c \text{ then } v \text{ IsOf } \tau; \\ (ii) & \text{if } v = \langle x, exp, E\rangle \text{ then there exists a } TE \text{ such} \\ & \text{that } TE \vdash \textbf{fn } x \texttt{ => } exp \Longrightarrow \tau \text{ and } \mathrm{Dom}(E) = \\ & \mathrm{Dom}(TE) \text{ and } E(x) : TE(x), \text{ for all } x \in \mathrm{Dom}(E). \end{cases} \tag{15}$$

Because of the existential quantification in (15.*ii*), these bi-implications do not constitute a definition, merely a property of a correspondence relation. The reader is probably surprised to see (15.*ii*); given that we are trying to prove the soundness of the type inference system, why refer to the type inference system itself in the definition of what it is for a closure to have a type?

There are two reasons, a pragmatic one and a technical one. The pragmatic reason is that the main interest of Theorem 5.1 is the case where E is an initial environment binding pre-defined variables to constants, TE is an initial type environment binding the same variables to their types, and v is a printable value, i.e., a constant rather than a closure. In this case the theorem gives the desired result independently of (15.*ii*).

The technical reason is that (15.*ii*) leads to a simple proof of the consistency theorem since a relation satisfying (15) can be obtained as the (maximum) fixed point of a monotonic operator as follows.

Let U be the set Val $\times$ Type, let $P(U)$ be the set of subsets of U and let $F : P(U) \to P(U)$ be the function defined by

$$\begin{aligned} F(Q) = \big\{(v,\tau) \in U \;\big|\; & \text{if } v = c \text{ then } v \text{ IsOf } \tau; \\ & \text{if } v = \langle x, exp, E\rangle \text{ then there exists a } TE \text{ such that} \\ & TE \vdash \textbf{fn } x \texttt{ => } exp \Longrightarrow \tau \text{ and } \mathrm{Dom}\, E = \mathrm{Dom}\, TE \\ & \text{and } \big(E(x), TE(x)\big) \in Q, \text{ for all } x \in \mathrm{Dom}\, E\big\}. \end{aligned} \tag{16}$$

It is clear, then, that the relations satisfying (15) are precisely the fixed points of F. Notice that F is monotonic with respect to set inclusion: $Q \subseteq Q'$ implies $F(Q) \subseteq F(Q')$. Since $(P(U), \subseteq)$ is a complete lattice, it follows from Tarski's fixed point theorem that F has a largest fixed point and a smallest fixed point, namely

$$Q^{\min} = \bigcap \big\{Q \subseteq U \;\big|\; F(Q) \subseteq Q\big\}$$

and

$$Q^{\max} = \bigcup \big\{Q \subseteq U \;\big|\; Q \subseteq F(Q)\big\}. \tag{17}$$

For our particular F, the minimum fixed point $Q^{\min}$ is strictly contained in the maximum fixed point $Q^{\max}$ and the minimum fixed point is too small. To demonstrate this, let us show that the closure cl_{fact} defined in the introduction has type `int -> int` if we take the correspondence relation to be $Q^{\max}$, but not if we take it to be $Q^{\min}$. To show $(cl_{\mathrm{fact}}, \texttt{int} \to \texttt{int}) \in Q^{\max}$, let us define $Q_{\mathrm{fact}} =$

$\{(cl_{\text{fact}}, \texttt{int} \rightarrow \texttt{int})\}$. Looking at the definition of F, we can now check that $Q_{\text{fact}} \subseteq F(Q_{\text{fact}})$. First, for the sought TE, take $\{\texttt{factorial} \mapsto (\texttt{int} \rightarrow \texttt{int})\}$. Next, it is easy to show that $TE \vdash \texttt{fn n =>}\ exp \Longrightarrow \texttt{int} \rightarrow \texttt{int}$ assuming that the IsOf relation associates the constants in exp, the body of `factorial`, with the obvious types. Finally, letting $E_{fact} = \{\texttt{factorial} \mapsto cl_{\text{fact}}\}$, E_{fact} and TE are defined on `factorial` only, and the pair $(E_{fact}(\texttt{factorial}), TE(\texttt{factorial}))$ is the element of Q_{fact}. Thus $Q_{\text{fact}} \subseteq F(Q_{\text{fact}})$. But $Q^{\max}$ contains all the subsets Q of U that satisfy $Q \subseteq F(Q)$, so $Q_{\text{fact}} \subseteq Q^{\max}$. Therefore $cl_{\text{fact}} : \texttt{int} \rightarrow \texttt{int}$ if we take : to be $Q^{\max}$.

On the other hand we have $(cl_{\text{fact}}, \texttt{int} \rightarrow \texttt{int}) \notin Q^{\min}$. To see this, let us recall that there is an alternative characterization of $Q^{\min}$, namely

$$R^{\min} = \bigcup_{\lambda} F^{\lambda}, \tag{18}$$

where $F^{\lambda} = F(\cup_{\mu<\lambda} F^{\mu})$, where λ ranges over all ordinals (see [1] for an introduction to inductive definitions). In other words, one obtains $R^{\min}$ by starting from the empty set and then applying F iteratively. However, intuitively speaking, there is no first point in the chain

$$\emptyset \subseteq F(\emptyset) \subseteq F(F(\emptyset)) \subseteq \cdots$$

where the non-well-founded object $(cl_{\text{fact}}, \texttt{int} \rightarrow \texttt{int})$ could enter because, according to the definition of F, cl_{fact} cannot be typed unless E_{fact} has already been typed, i.e. unless cl_{fact} itself has already been typed. More generally, for any monotonic F which has the property that if all members of Q are well-founded then so are all members of $F(Q)$, one can prove by transfinite induction that the minimum fixed point of F contains only well-founded objects.

We say that a subset $Q \subseteq U$ is *F-consistent* if $Q \subseteq F(Q)$. This use of language is motivated as follows. Q can be seen as a set of claims, each claim being a pair (v, τ) claiming that value v has type τ. If Q is F-consistent then there is a justification for each such claim $q \in Q$, either with or without reference to claims in Q. The former is the case when q is a pair of a constant and a type, in which case the definition of F ensures that the constant is of the claimed type. The latter is the case in our example above, where the element of Q_{fact} serves as justification for itself. (The fact that claims can serve as justifications for themselves makes the use of the word consistency very appropriate.) Indeed Q_{fact} is the smallest F-consistent set containing $(cl_{\text{fact}}, \texttt{int} \rightarrow \texttt{int})$. Note that $Q^{\max}$ is the largest F-consistent subset of U.

Associated with the device of defining a relation as the maximum fixed point of a monotonic operator is the important proof technique of *co-induction*:

> *Let U be any set, let $F : P(U) \rightarrow P(U)$ be a monotonic function and let R be the maximum fixed point of F. For any $Q \subseteq U$, in order to prove $Q \subseteq R$, it is sufficient to prove that Q is F-consistent i.e., that $Q \subseteq F(Q)$.*

The point is that $R = \bigcup\{Q \subseteq U \mid Q \subseteq F(Q)\}$, so R includes all F-consistent sets.

As an example of co-induction, assume we want to prove a theorem of the form $\forall x \in A.(P(x) \Rightarrow (e[x] \in R))$, where A is a set, P is a predicate, $e[x]$ is a formula which depends on x and R is the maximum fixed point of a monotonic operator $F : P(U) \rightarrow P(U)$, where U is any set. We can then define $Q = \{q \in U \mid \exists x \in A.(P(x) \wedge (q = e[x]))\}$ and attempt to prove $Q \subseteq F(Q)$. (For if $Q \subseteq F(Q)$ then by co-induction $Q \subseteq R$, and $Q \subseteq R$ is equivalent to the desired $\forall x \in A.(P(x) \Rightarrow (e[x] \in R))$.) Sometimes the inclusion $Q \subseteq F(Q)$ does not hold, in which case one must look for a set $Q' \supset Q$ which is F-consistent. It will even suffice to prove that $Q \cup R$ is F-consistent.

5 The Consistency Theorem

We take $v : \tau$ to mean $(v, \tau) \in Q^{\max}$, where $Q^{\max}$ is the maximum fixed point of the operator F defined by (16). The relation $E : TE$ is the pointwise extension of $v : \tau$. We can now formulate and prove the consistency theorem.

Theorem 5.1 (Consistency of Static and Dynamic Semantics)

$$\textit{If } E : TE \textit{ and } E \vdash exp \longrightarrow v \textit{ and } TE \vdash exp \Longrightarrow \tau \textit{ then } v : \tau.$$

Proof. By induction on the depth of inference of $E \vdash exp \longrightarrow v$. There is one case for each rule. The cases for constants, variables and application of a constant are trivial. Of the remaining cases, the one for **fix** is the most interesting, in that it uses co-induction.

Recursion, rule 6 Here the evaluation is of the form

$$\frac{cl_\infty = \langle x, exp, E + \{f \mapsto cl_\infty\}\rangle}{E \vdash \textbf{fix } f(x) = exp \longrightarrow cl_\infty} \tag{19}$$

and the elaboration is of the form

$$\frac{TE + \{f \mapsto \tau_1 \to \tau_2\} + \{x \mapsto \tau_1\} \vdash exp \Longrightarrow \tau_2}{TE \vdash \textbf{fix } f(x) = exp \Longrightarrow \tau_1 \to \tau_2}, \tag{20}$$

where $\tau = \tau_1 \to \tau_2$. To prove $cl_\infty : \tau$ by co-induction we define $Q = Q^{\max} \cup \{(cl_\infty, \tau)\}$, and prove that Q is F-consistent.
Take a $q \in Q$.

If $q \in Q^{\max}$ then $q \in F(Q)$ because $Q^{\max} \subseteq Q$ and the monotonicity of F implies $F(Q^{\max}) \subseteq F(Q)$ i.e., $Q^{\max} \subseteq F(Q)$.

Otherwise $q = (cl_\infty, \tau)$. Let $TE' = TE + \{f \mapsto \tau\}$ and $E' = E + \{f \mapsto cl_\infty\}$. We have $TE' + \{x \mapsto \tau_1\} \vdash exp \Longrightarrow \tau_2$ by (20) so $TE' \vdash \textbf{fn } x \Rightarrow exp \Longrightarrow \tau$ by inference rule 12. Since $E : TE$ we have $\mathrm{Dom}\, E = \mathrm{Dom}\, TE$ and for all $x \in \mathrm{Dom}\, E$, $E(x) : TE(x)$. So for all $x \in \mathrm{Dom}\, E$ we have $(E(x), TE(x)) \in Q$. Moreover $(E'(f), TE'(f)) = (cl_\infty, \tau) \in Q$. Thus $\mathrm{Dom}\, E' = \mathrm{Dom}\, TE'$ and for all $x \in \mathrm{Dom}\, E'$ we have $(E'(x), TE'(x)) \in Q$. So in this case as well, we have $q \in F(Q)$.

This proves that Q is F-consistent.

Abstraction, rule 5 Here the evaluation is of the form

$$\frac{}{E \vdash \textbf{fn } x \Rightarrow exp \longrightarrow \langle x, exp, E\rangle}$$

and the conclusion of the elaboration is $TE \vdash \textbf{fn } x \Rightarrow exp \Longrightarrow \tau$. Since in addition $E : TE$, the type environment TE satisfies the requirement (15.*ii*). Hence $\langle x, exp, E\rangle : \tau$.

Application of Closure, rule 8 Here the evaluation is of the form

$$\frac{\begin{array}{rcl} E & \vdash & exp_1 \longrightarrow \langle x', exp', E' \rangle \\ E & \vdash & exp_2 \longrightarrow v_2 \\ E'+\{x' \mapsto v_2\} & \vdash & exp' \longrightarrow v \end{array}}{E \vdash exp_1\ exp_2 \longrightarrow v} \tag{21}$$

and the elaboration is of the form

$$\frac{TE \vdash exp_1 \Longrightarrow \tau' \to \tau \quad TE \vdash exp_2 \Longrightarrow \tau'}{TE \vdash exp_1\ exp_2 \Longrightarrow \tau} \tag{22}$$

for some τ'.

By induction on the first premises of (21) and (22) together with $E : TE$ we get

$$\langle x', exp', E' \rangle : \tau' \to \tau. \tag{23}$$

Similarly we get $v_2 : \tau'$ by induction on the second premises. From (23) and the fact that : is a fixed point of F, there exists a TE' with $E' : TE'$ and

$$TE \vdash \textbf{fn}\ x' \texttt{=>}\ exp' \implies \tau' \to \tau. \tag{24}$$

Take such a TE'; this type environment allows us to use induction a third time. (Indeed this is why the "$E : TE$" is important in (15.*ii*)). More precisely, since $E' : TE'$ and $v_2 : \tau'$ we have

$$E'+\{x' \mapsto v_2\} : TE'+\{x' \mapsto \tau'\}. \tag{25}$$

Moreover, (24) must be obtained from the premise

$$TE'+\{x' \mapsto \tau'\} \vdash exp' : \tau. \tag{26}$$

Noticing that the third premise of (21) was deduced in fewer steps than the conclusion, we can use induction on it together with (25) and (26) to deduce the desired $v : \tau$. ■

6 Discussion

Since the $v : \tau$ relation is an extension of the c IsOf τ relation, Theorem 5.1 implies the basic consistency result (namely that if $E \vdash exp \longrightarrow c$ and $TE \vdash exp \Longrightarrow \tau$ and E IsOf TE then c IsOf τ). However, there are other natural extensions of the IsOf relation for which one can attempt to prove the consistency result. One is

$$v :' \tau \quad \textit{iff} \quad \begin{cases} (i) & \text{if } v = c \text{ then } v \text{ IsOf } \tau; \\ (ii) & \text{if } v = \langle x, exp, E \rangle \text{ then there exist } \tau_1, \tau_2, \text{ such that} \\ & \tau = \tau_1 \to \tau_2 \text{ and for all } v_1, v_2, \text{ if } v_1 :' \tau_1 \text{ and} \\ & E+\{x \mapsto v_1\} \vdash exp \longrightarrow v_2 \text{ then } v_2 :' \tau_2. \end{cases} \tag{27}$$

Interestingly, the operator F' associated with this revised property is no longer monotonic with respect to set inclusion because of the occurrence of "$v_1 :' \tau_1$" on the lefthand side of the implication. Nevertheless, there is precisely one relation $:' \subseteq \text{Val} \times \text{Type}$ satisfying (27); this can be seen by induction on the structure of type expressions.

There are closures that have a type using $:'$ but have no type using $:$. One example is the closure $\langle \mathtt{n}, \mathtt{if\ true\ then\ 7+n\ else\ false}, \{\}\rangle$. However, we do not know whether $:$ is contained in $:'$. The consistency result can be proved using $:'$ instead of $:$; the proof we have again uses co-induction, but it is rather involved and therefore not included here.

The justification for the existence of fixed points were completely different in the two cases. Sometimes it is not at all obvious whether a given F has any fixed points at all. For example, let us extend our set v of values by *constructed values*,

$$\begin{array}{rcll} v \in \text{Val} & = & \text{Const} \uplus \text{ConVal} \uplus \text{Clos} & \text{Values} \\ c(v) \in \text{ConVal} & = & \text{Const} \times \text{Val} & \text{Constructed Values} \end{array}$$

and consider the property

$$v : \tau \quad \textit{iff} \quad \begin{cases} (i) & \text{if } v = c \text{ then } v \text{ IsOf } \tau; \\ (ii) & \text{if } v = c(v_1) \text{ then there exists a } \tau_1 \text{ such that} \\ & c \text{ IsOf } (\tau_1 \to \tau) \text{ and } v_1 : \tau_1 \\ (iii) & \text{if } v = \langle x, exp, E\rangle \text{ then there exist } \tau_1, \tau_2, \text{ such that} \\ & \tau = \tau_1 \to \tau_2 \text{ and for all } v_1, v_2, \text{ if } v_1 : \tau_1 \text{ and} \\ & E + \{x \mapsto v_1\} \vdash exp \longrightarrow v_2 \text{ then } v_2 : \tau_2. \end{cases} \tag{28}$$

The operator associated with this property is not monotonic with respect to set inclusion. Neither is this property a definition on the structure of types because of (ii). We do not see how to justify the existence of such a relation without making assumptions about the IsOf relation.

This should not leave the impression, however, that the technique of using maximum fixed points can rarely be applied. In fact, we have encountered several situations in operational semantics where the technique turns out to be very strong. In general, the technique is useful when considering consistency properties. Consistency is often of interest when one wants to relate non-well-founded objects, or more generally objects whose behaviour is in some sense infinite. Indeed, in the introduction we indicated that closures can be treated in either of these ways; in each case, typing of closures is a consistency property. Another example is the notion of observation equivalence in CCS [7] which is defined as the maximum fixed point of a certain monotonic operator. The idea is that two agents are bisimilar if the hypothesis that they are susceptible to the same observations is consistently maintained during computation. Finally the technique has been used to prove the soundness of a type discipline for polymorphic references [11, 12]. Here the need for taking the maximum fixed point in the definition of what it is for a value to

have a type arises because, when locations are values, one can create cycles in the store; since the type of a location depends on the type of the value it contains, a cycle in the store may have a consistent typing although the justification of the typing is a cyclic argument (i.e. a consistent claim rather than something that in finitely many steps can be reduced to a question of constants being typed according to the IsOf relation).

References

[1] P. Aczel, *An Introduction to Inductive Definitions*, in J. Barwise (ed.) *Handbook of Mathematical Logic*, North–Holland Publishing Company, 1977

[2] P. Aczel, *Non-Well-Founded Sets*, CSLI Lecture Notes, Number 14, LSCI/Stanford, 1988

[3] L. Damas, *Type Assignment in Programming Languages*, Ph. D. Thesis, University of Edinburgh, Department of Computer Science, CST-33-85, 1985.

[4] L. Ḍamas and R. Milner, *Principal type schemes for functional programs*, in Proceedings of the 9th ACM Symposium on the Principles of Programming Languages, pp. 207–212, 1982

[5] G. Kahn., *Natural Semantics*, Proc. of Symp. on Theoretical Aspects of Computer Science, Passau, Germany, February 1987.

[6] R. Milner, *A theory of type polymorphism in programming languages*, Journal of Computer and System Sciences, **17**, 348–375.

[7] R. Milner, *A Calculus of Communicating Systems*, Lecture Notes in Computer Science 92, ed., G. Goos and J. Hartmanis, Springer, Berlin, 1980.

[8] D. Park, *On the Semantics of Fair Parallelism,* in Abstract Software Specifications, ed. Bjørner, Springer LNCS 86.

[9] D. Park, *Concurrency and Automata on Infinite Sequences,* in Proc. 5th GI conference on TCS, Springer LNCS 104.

[10] G. Plotkin, *A Structural Approach to Operational Semantics*, Technical Report DAIMI–FN–19, Computer Science Department, Aarhus University, Denmark, 1981

[11] M. Tofte, *Operational Semantics and Polymorphic Type Inference*, Ph. D. thesis, Edinburgh University, CST-52-88, 1987.

[12] M. Tofte, *Type Inference for Polymorphic References*, under preparation.