Effect Systems with Subtyping

Yan Mei Tang
Pierre Jouvelot

Centre de Recherche en Informatique
Ecole Des Mines de Paris
E-mail: {tang,jouvelot}@cri.ensmp.fr

Abstract

Effect systems extend classical type systems with effect information. Just as types describe the possible values of expressions, effects describe their possible evaluation behaviors. Effects, which appear in function types, introduce new constraints on the typability of expressions. To increase the flexibility and accuracy of effect systems, we present a new effect system based on subtyping. The subtype relation is induced by a subsumption relation on effects. This *subtyping effect system* avoids merging effect information together, thus collecting more precise effect information. We introduce a reconstruction algorithm which for any expression already typed with classical types, reconstructs its type and effect based on the subtype relation. The reconstruction algorithm is sound and complete w.r.t. the static semantics.

1 Introduction

Effect systems extend classical type systems with effect information. Just as types describe the possible values of expressions, effects describe their possible evaluation behaviors. Effect systems allow powerful static analysis to be performed in the presence of higher-order functions, imperative constructs and separate compilation. However, effects, which appear in function types, introduce new constraints on the typability of expressions, i.e., effect checking may force the rejection of programs which would have type-checked if no effects were present.

To increase the flexibility of previous effect systems, *subeffecting* has been introduced [Gifford87, Talpin92, Tang92]. Subeffecting allows expressions to admit larger effects, thus enabling type misatches due to the introduction of effect information to be eliminated. But, subeffecting alone forces a variable to have a unique type in different occurences and thus merges effect information together; this often limits the accuracy of the effect analysis. Instead of relying on subeffecting to eliminate undue type mismatches, we use the notion of *effect-based subtyping* to improve both the flexibility and accuracy of effect systems, while preserving type safety and reconstruction.

We present a new effect system based on subtyping where the subtype relation is induced by a subsumption rule on effects. This subtyping effect system avoids merging effect information together when forcing two types to be identical, thus it collects more precise effect information. We introduce a sound and complete reconstruction algorithm for this static semantics. Since type inequalities are only introduced by effect subsumption, it is a simple extension of classical type reconstruction algorithms with effect constraints whose solutions satisfy the subtype relation. To motivate this new notion, we show how to use effect-based subtyping within the effect system for control-flow analysis presented in [Tang92], thus improving the accuracy of this control-flow analysis technique.

In the sequel, we define a type and effect static semantics based on ubtyping (Section 2), present the type and effect reconstruction algorithm and prove it is sound and complete w.r.t. the static semantics (Section 3), discuss the related work (Section 4) before concluding (Section 5). The proofs are given in appendix.

2 Static Semantics with Subtyping

2.1 Language

A simple functional language is enough to present our ideas, although our analysis can be extended to additional language constructs, such as constants, imperative operations, separate compilation (in the vein of [Tang94-1, Tang94-2]). In particular, our analysis can also deal with polymorphism as presented in [Tang94-1]. The syntax of expressions is defined as follows:

$$
\begin{array}{lll}
\mathtt{e} ::= & \mathtt{x} & \textit{value identifier} \\
 & (\lambda_{\mathtt{n}}\ (\mathtt{x})\ \mathtt{e}) & \textit{abstraction} \\
 & (\mathtt{rec}_{\mathtt{n}}\ (\mathtt{f}\ \mathtt{x})\ \mathtt{e}) & \textit{recursive function} \\
 & (\mathtt{e}\ \mathtt{e}') & \textit{application}
\end{array}
$$

where all lambda expressions, recursive or not, are explicitly given a name n (from the domain Id of identifiers) which is used to uniquely identify them. These names could be automatically assigned by the reconstruction process.

2.2 Domains

Classical types specify the data structure of expressions. A classical type τ can either be *int*, a type variable α, or a function type $\tau' \rightarrow \tau$. A classical type environment $\mathcal{T}$ is a finite map from identifiers to classical types.

$$
\begin{array}{llll}
\tau \in & \text{CType} & = int \mid \alpha \mid \tau' \rightarrow \tau & \textit{classical type} \\
\mathcal{T} \in & \text{CTEnv} & = \text{Id} \mapsto \tau & \textit{classical environment}
\end{array}
$$

Effect systems extend classical types with effect information. An *type* t is either *int*, a type variable α, or a function type $t' \xrightarrow{c} t$ with the latent effect c, which abstracts the control-flow behavior of the function body. A type environment $\mathcal{E}$ is a finite map from identifiers to types.

$$\begin{array}{lll} t \in \ \text{Type} & = int \mid \alpha \mid t' \xrightarrow{c} t & \textit{type} \\ \mathcal{E} \in \ \text{TEnv} & = \text{Id} \mapsto t & \textit{type environment} \end{array}$$

Control-flow effects record the function names that are possibly called during the evaluation of expressions. An effect c can either be the constant $\emptyset$, denoting the absence of any function call, an effect variable ζ, a singleton $\{\mathtt{n}\}$ where $\mathtt{n}$ is the name of a called function, or a union set of function names indicated by the infix union operator $\cup$.

$$c \in \ \text{Control} \ = \emptyset \mid \zeta \mid \{\mathtt{n}\} \mid c \cup c' \quad \textit{control-flow}$$

2.3 Subtype and Subeffect Relations

An effect, i.e. a set of function names, can be conservatively approximated by one of its supersets. The subeffect relation is thus the usual set inclusion relation.

The subtype relation $\leq$ is defined via effect inclusion between latent effects of function types that have the same structure. To properly define this notion, we introduce the *Struct* function which transforms types to classical types by erasing latent effects.

$$\begin{array}{ll} Struct(int) & = int \\ Struct(\alpha) & = \alpha \\ Struct(t' \xrightarrow{c} t) & = Struct(t') \rightarrow Struct(t) \end{array}$$

The *type structure* of t is $Struct(t)$. Two types t and t' have the same structure if and only if $Struct(t) = Struct(t')$.

The subtype relation $t \leq t'$ is defined whenever t and t' have the same structure. Note that the subtype relation between function types is contravariant.

Definition 1 (Subtype)

$$\begin{array}{l} \alpha \leq \alpha \\ int \leq int \\ t'_0 \xrightarrow{c_0} t_0 \ \leq \ t'_1 \xrightarrow{c_1} t_1 \quad \Leftrightarrow \quad t'_1 \leq t'_0 \ \wedge \ t_0 \leq t_1 \ \wedge c_1 \supseteq c_0 \end{array}$$

The function *Eff* generates the set of effect inequalities corresponding to a given type inequality. An effect inequality is a pair (c_i, c'_i) written $c_i \supseteq c'_i$.

$$\begin{array}{ll} \mathit{Eff}(\alpha \leq \alpha) & = \emptyset \\ \mathit{Eff}(int \leq int) & = \emptyset \\ \mathit{Eff}(t'_0 \xrightarrow{c_0} t_0 \leq t'_1 \xrightarrow{c_1} t_1) & = \{c_1 \supseteq c_0\} \cup \mathit{Eff}(t'_1 \leq t'_0) \\ & \qquad \cup \mathit{Eff}(t_0 \leq t_1) \end{array}$$

2.4 Semantics

The static semantics defines the type and control-flow effect of expressions. It is specified by a set of inference rules [Plotkin81]. Given a type environment $\mathcal{E}$, the inference rules associate an expression $\mathtt{e}$ with its type t and control-flow information c. We write :

$$\mathcal{E} \vdash \mathtt{e} : t, c$$

The crucial rules are the (abs) and (app) rules for lambda abstraction and application. In the abstraction case, the current function name is added to the functions called by the lambda body; the resulting set is the latent control-flow effect of the lambda expression. When such a function is applied, in the (app) rule, this latent control-flow information is used to determine the functions possibly called while evaluating the function body.

$$(var) : \mathcal{E}[\mathtt{x} \mapsto t] \vdash \mathtt{x} : t, \emptyset$$

$$(abs) : \frac{\mathcal{E}[\mathtt{x} \mapsto t'] \vdash \mathtt{e} : t, c}{\mathcal{E} \vdash (\lambda_{\mathtt{n}}\ (\mathtt{x})\ \mathtt{e}) : t' \xrightarrow{\{\mathtt{n}\} \cup c} t, \emptyset}$$

$$(rec) : \frac{\mathcal{E}[\mathtt{f} \mapsto t] \vdash (\lambda_{\mathtt{n}}\ (\mathtt{x})\ \mathtt{e}) : t, \emptyset}{\mathcal{E} \vdash (\mathtt{rec}_{\mathtt{n}}\ (\mathtt{f}\ \mathtt{x})\ \mathtt{e}) : t, \emptyset}$$

$$(app) : \frac{\begin{array}{l}\mathcal{E} \vdash \mathtt{e} : t' \xrightarrow{c''} t, c \\ \mathcal{E} \vdash \mathtt{e}' : t', c'\end{array}}{\mathcal{E} \vdash (\mathtt{e}\ \mathtt{e}') : t, c \cup c' \cup c''}$$

$$(sub) : \frac{\begin{array}{l}\mathcal{E} \vdash \mathtt{e} : t', c \\ t' \leq t\end{array}}{\mathcal{E} \vdash \mathtt{e} : t, c}$$

The novelty here lies in the (sub) rule where we use subtyping to allow a larger type t to be used in lieu of t'. This increases the flexibility of the static semantics by relaxing the constraint on latent effects imposed by the context of an expression. We show that this new approach performs a better analysis than the one previously introduced in [Tang92] (see an example in Section 4) which used the less precise subeffecting rule:

$$(\textit{subeffecting}) : \frac{\begin{array}{l}\mathcal{E} \vdash \mathtt{e} : t, c' \\ c \supseteq c'\end{array}}{\mathcal{E} \vdash \mathtt{e} : t, c}$$

3 Reconstruction with Subtyping

We present a new reconstruction algorithm that reconstructs types and effects of expressions based on the subtype relation. We describe the basic ideas, present the algorithm and state its correctness.

3.1 Basic Idea

The reconstruction of types and effects based on subtyping is a type inequalities solving problem. Since the subtype relation in our system is defined by the subsumption relation on effects, type inequalities amount to sets of effect inequalities when the structures of the types are known. Therefore, we define a type and effect reconstruction algorithm $\mathcal{S}$ which operates on expressions already typed with classical types. For any expression, the reconstruction algorithm $\mathcal{S}$ computes a set of type inequalities beside its type and effect. Since classical types specify type structures, solving type inequalities is reduced to solving the corresponding effect inequalities. Thus reconstruction can be viewed as an effect constraint satisfaction problem. For every expression that has a type and a control-flow effect in the static semantics, its effect constraint set must have at least one solution, which satisfies the set of type inequalities. The classical types of expressions are reconstructed by a simple type reconstruction algorithm [Milner78, Tofte87].

3.2 Algorithm $\mathcal{S}$

Given a type environment $\mathcal{E}$ and an expression e assumed priorly decorated with its classical type (we use a straightforward expression annotation mechanism to express this information in the algorithm), the reconstruction algorithm $\mathcal{S}$ computes a type t, an effect c and an effect constraint set κ. We note :

$$\mathcal{S}(\mathcal{E}, \mathtt{e}) = \langle t, c, \kappa \rangle$$

The effect constraint set is partly built by application of *Eff* to type inequalities and partly during the reconstruction of `lambda` and `rec` expressions. The function *New* transforms a classical type τ to a type t by adding fresh latent effect variables to τ. Its proper definition is:

$$\begin{array}{ll} New(int) & = int \\ New(\alpha) & = \alpha \\ New(\tau' \to \tau) & = New(\tau') \xrightarrow{\zeta} New(\tau) \quad \textit{for fresh } \zeta \end{array}$$

The inference algorithm $\mathcal{S}$ is defined as follows:

$$\begin{array}{l} \mathcal{S}(\mathcal{E}, \mathtt{x}) \Rightarrow \\ \quad let\ \ t' = \mathcal{E}(\mathtt{x}) \\ \qquad\quad t = New(Struct(t')) \\ \quad in\ \ \langle t, \emptyset, \mathit{Eff}(t' \leq t) \rangle \end{array}$$

$$\begin{array}{l} \mathcal{S}(\mathcal{E}, (\lambda_{\mathtt{n}}\ (\mathtt{x} : \tau)\ \mathtt{e})) \Rightarrow \\ \quad let\ \ t' = New(\tau) \\ \qquad\quad \zeta\ new \\ \qquad\quad \langle t, c, \kappa \rangle = \mathcal{S}(\mathcal{E}[\mathtt{x} \mapsto t'], \mathtt{e}) \\ \quad in\ \ \langle t' \xrightarrow{\zeta} t, \emptyset, \kappa \cup \{\zeta \supseteq \{\mathtt{n}\} \cup c\} \rangle \end{array}$$

$$\begin{array}{l} \mathcal{S}(\mathcal{E}, (\mathtt{rec}_{\mathtt{n}}\ (\mathtt{f} : \tau' \to \tau\ \mathtt{x} : \tau')\ \mathtt{e})) \Rightarrow \\ \quad let\ \ t' \xrightarrow{\zeta} t = New(\tau' \to \tau) \\ \qquad\quad \langle t'', c, \kappa \rangle = \mathcal{S}(\mathcal{E}[\mathtt{f} \mapsto t' \xrightarrow{\zeta} t][\mathtt{x} \mapsto t'], \mathtt{e}) \\ \quad in\ \ \langle t' \xrightarrow{\zeta} t, \emptyset, \kappa \cup \mathit{Eff}(t'' \leq t) \cup \{\zeta \supseteq \{\mathtt{n}\} \cup c\} \rangle \end{array}$$

$$\begin{array}{l} \mathcal{S}(\mathcal{E}, (\mathtt{e}\ \mathtt{e}')) \Rightarrow \\ \quad let\ \langle t'' \xrightarrow{c''} t, c, \kappa \rangle = \mathcal{S}(\mathcal{E}, \mathtt{e}) \\ \qquad\ \ \langle t', c', \kappa' \rangle = \mathcal{S}(\mathcal{E}, \mathtt{e}') \\ \quad in\ \langle t, c \cup c' \cup c'', \kappa \cup \kappa' \cup \mathit{Eff}(t' \leq t'') \rangle \end{array}$$

Subeffecting can be easily related to subtyping by noticing that its related reconstruction algorithm [Tang94-1] is similar to $\mathcal{S}$, except that $\leq$ is replaced by the more restrictive $=$, implemented by unification.

3.3 Properties of $\mathcal{S}$

The reconstruction algorithm $\mathcal{S}$ has the following properties, easily proved by induction :

Lemma 1 (Properties of $\mathcal{S}$) *For any $\mathcal{E}$, e, if $\mathcal{S}(\mathcal{E}, \mathtt{e}) = \langle t, c, \kappa \rangle$, then :*

- *t only includes fresh effect variables.*
- *All environment extensions within $\mathcal{S}$ refer to types with only fresh effect variables.*

The previous lemma implies that the constraint set computed by the reconstruction algorithm $\mathcal{S}$ has the following normal form property:

Lemma 2 (Normal Constraints) *If $\mathcal{S}(\mathcal{E}, \mathtt{e}) = \langle t, c, \kappa \rangle$, then κ has the normal form $\{\zeta_i \supseteq c_i \mid i = 1..s\}$.*

Proof See the appendix.

3.4 Constraint Satisfaction

An expression e with its type environment $\mathcal{E}$ is type and effect safe if and only if the constraint set κ computed by $\mathcal{S}(\mathcal{E}, \mathtt{e})$ admits at least one solution. A constraint set that is in normal form always has solutions, among which we are interested in the minimal one. The substitutions satisfying κ are called *effect models.*

Definition 2 (Effect Model) *A substitution μ is an* effect model *of a constraint set κ, noted as $\mu \models \kappa$, if and only if $\forall \zeta \supseteq c \in \kappa$, $\mu\zeta \supseteq \mu c$.*

The following lemma shows how to satisfy a type inequality by solving its corresponding effect constraint.

Lemma 3 (Solution of Type Inequality) *If t and t' have the same structure and satisfy Lemma 1, then*

$$\mu \models \mathit{Eff}(t' \leq t) \Leftrightarrow \mu t' \leq \mu t$$

Proof By induction of the structure of types.

Theorem 1 (Satisfaction) *Every normal form constraint set $\kappa = \{\zeta_i \supseteq c_i \mid i = 1..s\}$ admits at least one model.*

Proof $\{\zeta_i \mapsto c_i' \mid i = 1 \ldots n\}$ is an effect model of κ, where $c_i' = \cup_{i=1}^{n} c_i \setminus \cup_{i=1}^{n} \zeta_i$, where $\setminus$ is the set difference operator.

A constraint set may admit more than one effect model, among which we are interested in the minimal one. We define a function *Min* to characterize the minimal effect model of a constraint set κ. Note that the solution is independent of the order of inequalities in κ because of the algebraic properties of $\cup$: the function *Min* recursively computes an effect model by applying each solved inequation to the residual constraints.

$$\begin{array}{rl} Min(\emptyset) & = \mathrm{Id} \\ Min(\{\zeta \supseteq c\} \cup \kappa') & = let\ \mu = Min(\kappa')\ in\ \{\zeta \mapsto \mu c \setminus \zeta\}\mu \end{array}$$

The constraint set of the reconstruction algorithm always admits a unique minimal model with respect to the subsumption relation $\supseteq$ on effects.

Theorem 2 (Minimality) *Any constraint set admits a unique minimal effect model.*

Proof By induction on the constraint set.

3.5 Correctness

Since the reconstruction algorithm $\mathcal{S}$ is defined by induction on the structure of expressions, which are of finite height, it always terminates.

The reconstruction algorithm is sound and complete with respect to the static semantics. The soundness theorem states that the application of any effect model of the reconstructed type constraint set to the reconstructed type and effect satisfies the static semantics.

Theorem 3 (Soundness) *Given an expression* e *and its type environment $\mathcal{E}$, if $\mathcal{S}(\mathcal{E}, \mathtt{e}) = \langle t, c, \kappa \rangle$, then, for any effect model μ of κ, one has:*

$$\mu\mathcal{E} \vdash \mathtt{e} : \mu t, \mu c$$

Proof See the appendix.

The completeness theorem states that the reconstructed type t and the control-flow effect c are minimal with respect to any type t_1 and control-flow effect c_1 derivable from the static semantics, modulo some substitution μ that satisfies the computed constraint set κ. The substitution θ_1 ranges over the free variables of $\mathcal{E}$.

Theorem 4 (Completeness) *If* $\theta_1\mathcal{E} \vdash \mathsf{e} : t_1, c_1$ *then* $\mathcal{S}(\mathcal{E}, \mathsf{e}) = \langle t, c, \kappa\rangle$ *and there exists an effect model* μ *of* κ*, such that:*

$$\theta_1\mathcal{E} = \mu\mathcal{E} \quad \textit{and} \quad \mu t \leq t_1 \quad \textit{and} \quad c_1 \supseteq \mu c$$

Proof See the appendix.

4 Related Work

Subtyping (see e.g. [Cardelli88]) adds flexibility to type systems by allowing type coercions to be performed if necessary in the presence of type mismatches. It is often used to captures aspects of object-oriented programming [Wand87, Stansifer88]. Subtyping in effect systems has been previously introduced in explicitly typed languages [Gifford87, Consel94]. There, a subsumption rule similar to the one presented above was used, but since only type checking was performed, its treatment was simpler than ours. This paper shows that type and effect reconstruction may be performed in an implicitly typed language.

Previous implicit effect systems [Dornic91, Talpin92, Tang92] have introduced *subeffecting*, via the *subeffecting* rule (see Section 2), to increase the flexibility of the static semantics. Subeffecting allows expressions of same classical types to also have the same effect-including types by allowing such effects to be replaced by larger ones if need be. Subtyping eliminates this information loss by allowing these expressions to simply obey the subtype relation. We show below on an example how subtyping can thus be more precise than subeffecting:

$$\begin{array}{l}((\lambda \mathtt{n}_f \ \ (f) \\ \qquad\qquad (+\ (f\ (\lambda \mathtt{n}_a\ (a)\ a))_{\mathtt{l}_a} \\ \qquad\qquad\quad (f\ (\lambda \mathtt{n}_b\ (b)\ b))_{\mathtt{l}_b})) \\ \ \ (\lambda \mathtt{n}_g\ (g)\ (g\ 1)))_{\mathtt{l}_f}\end{array}$$

There, the function f is bound when performing the call $\mathtt{l}_f$ and is applied at $\mathtt{l}_a$ and $\mathtt{l}_b$ with arguments $(\lambda \mathtt{n}_a\ (a)\ a)$ and $(\lambda \mathtt{n}_b\ (b)\ b)$ respectively. We give, in the following table, the types of f at these three occurrences (t_f, t_{fa} and t_{fb}), and the types of the arguments $\lambda\mathtt{n}_a$ and $\lambda\mathtt{n}_b$ ($t_{\mathtt{n}_a}$ and $t_{\mathtt{n}_b}$) For clarity, we use i to indicates the type *int*.

	Subeffecting	Subtyping
t_f	$(i \xrightarrow{\{\mathtt{n}_a,\mathtt{n}_b\}} i) \xrightarrow{\{\mathtt{n}_g,\mathtt{n}_a,\mathtt{n}_b\}} i$	$(i \xrightarrow{\{\mathtt{n}_a,\mathtt{n}_b\}} i) \xrightarrow{\{\mathtt{n}_g,\mathtt{n}_a,\mathtt{n}_b\}} i$
t_{fa} $t_{\mathtt{n}_a}$	$(i \xrightarrow{\{\mathtt{n}_a,\mathtt{n}_b\}} i) \xrightarrow{\{\mathtt{n}_g,\mathtt{n}_a,\mathtt{n}_b\}} i$ $i \xrightarrow{\{\mathtt{n}_a,\mathtt{n}_b\}} i$	$(i \xrightarrow{\{\mathtt{n}_a\}} i) \xrightarrow{\{\mathtt{n}_g,\mathtt{n}_a,\mathtt{n}_b\}} i$ $i \xrightarrow{\{\mathtt{n}_a\}} i$
t_{fb} $t_{\mathtt{n}_a}$	$(i \xrightarrow{\{\mathtt{n}_a,\mathtt{n}_b\}} i) \xrightarrow{\{\mathtt{n}_g,\mathtt{n}_a,\mathtt{n}_b\}} i$ $i \xrightarrow{\{\mathtt{n}_a,\mathtt{n}_b\}} i$	$(i \xrightarrow{\{\mathtt{n}_b\}} i) \xrightarrow{\{\mathtt{n}_g,\mathtt{n}_a,\mathtt{n}_b\}} i$ $i \xrightarrow{\{\mathtt{n}_b\}} i$
	$t_{fa} = t_f$ $t_{fb} = t_f$	$t_f \leq t_{fa}$ $t_f \leq t_{fb}$

Notice that, when using subeffecting, all occurrences of f are forced to have the same type while, when using subtyping, they only have to obey a subtype relation, leading to more precise local control-flow information.

5 Conclusion

We presented a new effect system based on subtyping where expressions with the same structure obey a subtype relation defined by a subsumption relation on effects. This subtype effect system avoids merging effect information together, thus collects more precise effect information than effect systems with subeffecting. We designed a sound and complete reconstruction algorithm that reconstructs the types and effects of expressions in the presence of subtyping, and show that it outperforms previous systems. A natural extension of this paper is the possibility of combining subtyping and subeffecting in a single framework. This has been proved valuable in [?].

Acknowledgements

We thank Jean-Pierre Talpin for his numerous comments on this paper.

References

[Cardelli88] Cardelli, L. Structural Subtyping and the Notion of Power Type. In *ACM Symposium on Principles of Programming Languages*, pages 70-79, 1988.

[Consel93] Consel, C., and Jouvelot, P. Separate Polyvariant Binding-Time Analysis. OGI Tech. Rep. CS/E 93-006, March 1993.

[Consel94] Consel, C., Jouvelot, P., and Orbaek, P. Separate Polyvariant Binding-Time Reconstruction. Technical Report A-261, Ecole des Mines de Paris, July 1994.

[Dornic91] Dornic, V. and Jouvelot, P. Polymorphic Time Systems for Estimating Program Complexity. In *JTASPEFL'91, Bordeaux, France*, 1991.

[Gifford87] Gifford, D. K., Jouvelot, P., Lucassen, J. M., and Sheldon, M. A. FX-87 Reference Manual. *MIT/LCS/TR-407*, MIT Laboratory for Computer Science, September 1987.

[Milner78] Milner, R. A Theory for type polymorphism in programming. In *Journal of Computer and Systems Sciences*, Vol. 17, pages 348-375, 1978.

[Plotkin81] Plotkin, G. A structural approach to operational semantics. *Technical report DAIMI-FN-19*. Aarhus University, 1981.

[Stansifer88] Stansifer, R. Type Inference with Subtypes. In *ACM Symposium on Principles of Programming Languages*, 1988.

[Talpin92] Talpin, J. P., and Jouvelot, P. Polymorphic Type, Region and Effect Inference. In the *Journal of Functional Programming*, volume 2, number 3. Cambridge University Press, 1992.

[Tang92] Tang, Y. M., and Jouvelot, P. Control-Flow Effects for Closure Analysis. In *Proceedings of the 2nd Workshop on Semantics Analysis*, Bigre numbers 81-82, pages 313-321. Bordeaux, October 1992.

[Tang94-1] Tang, Y. M. Systèmes d'Effet et Interprétation Abstraite pour l'Analyse de Flot de Contrôle. Doctoral Dissertation. Ecole des Mines de Paris et Université Paris VI, March 1994.

[Tang94-2] Tang, Y. M., and Jouvelot, P. Separate Abstract Interpretation for Control-Flow Analysis. *International Symposium on Theoretical Aspects of Computer Software*, Springer Verlag, LNCS 789. Japan, April 1994.

[Tofte87] Tofte, M. Operational semantics and polymorphic type inference. *PhD Thesis*, University of Edinburgh, 1987.

[Wand87] Wand, M. Complete type inference for simple objects. In *Proceedings of the 2nd IEEE Symposium on Logic in Computer Science*, 1987, pages 37-44.

Appendix

Proof of Lemma 2

Lemma 2 (Formal Effect Constraints) If $\mathcal{S}(\mathcal{E}, \mathtt{e}) = \langle t, c, \kappa \rangle$, then κ is of the following form:

$$\{\zeta_i \supseteq c_i \mid i = 1..s\}$$

Proof By induction of the structure of expressions.

- Case of $\mathtt{x}$

  The hypothesis is
  $\mathcal{S}(\mathcal{E}, \mathtt{x}) = \langle t, \emptyset, \mathit{Eff}(t' \leq t) \rangle$

  By the definition of $\mathcal{S}$
  (1) $t' = \mathcal{E}(\mathtt{x})$
  (2) $t = \mathit{New}(\mathit{Struct}(t'))$

  From (1), by Lemma 1
  (3) t' only includes fresh effect variables

  From (2)(3), by the definition of *Eff*
  $\mathit{Eff}(t' \leq t)$ satisfies the lemma

- Case of $(\lambda_{\mathtt{n}}\ (\mathtt{x})\ \mathtt{e})$

  The hypothesis is
  $\mathcal{S}(\mathcal{E}, (\lambda_{\mathtt{n}}\ (\mathtt{x} : \tau)\ \mathtt{e})) = \langle t' \xrightarrow{\zeta} t, \emptyset, \kappa \cup \{\zeta \supseteq \{\mathtt{n}\} \cup c\} \rangle$

  By the definition of $\mathcal{S}$
  (1) ζ *new*
  (2) $\langle t, c, \kappa \rangle = \mathcal{S}(\mathcal{E}[\mathtt{x} \mapsto t'], \mathtt{e})$

  From (2), by induction
  (3) κ satisfies the lemma

  From (1)(3)
  $\kappa \cup \{\zeta \supseteq \{\mathtt{n}\} \cup c\}$ satisfies the lemma

- Case of $(\mathtt{rec_n}\ (\mathtt{f}\ \mathtt{x})\ \mathtt{e})$

  The hypothesis is
  $\mathcal{S}(\mathcal{E}, (\mathtt{rec_n}\ (\mathtt{f} : \tau' \to \tau\ \mathtt{x} : \tau')\ \mathtt{e}))=$
  $\langle t' \xrightarrow{\zeta} t, \emptyset, \kappa \cup \mathit{Eff}(t'' \leq t) \cup \{\zeta \supseteq \{\mathtt{n}\} \cup c\} \rangle$

  By the definition of $\mathcal{S}$
  (1) $t' \xrightarrow{\zeta} t = \mathit{New}(\tau' \to \tau)$
  (2) $\langle t'', c, \kappa \rangle = \mathcal{S}(\mathcal{E}[\mathtt{f} \mapsto t' \xrightarrow{\zeta} t][\mathtt{x} \mapsto t'], \mathtt{e})$

  From (1), by the definition of *New*
  (3) $t = \mathit{New}(\tau)$
  (4) ζ *new*

  From (2), by Lemma 1
  (5) t'' only includes fresh effect variables

  From (2), by induction
  (6) κ satisfies the lemma

  From (3)(5), by the definition of *Eff*
  (7) $\mathit{Eff}(t'' \leq t)$ satisfies the lemma

  From (6)(7)(4)
  $\kappa \cup \mathit{Eff}(t'' \leq t) \cup \{\zeta \supseteq \{\mathtt{n}\} \cup c\}$ satisfies the lemma

- Case of $(\mathtt{e}\ \mathtt{e}')$

  The hypothesis is
  $\mathcal{S}(\mathcal{E}, (\mathtt{e}\ \mathtt{e}')) = \langle t, c \cup c' \cup \zeta, \kappa \cup \kappa' \cup \mathit{Eff}(t' \leq t'') \rangle$

  By the definition of $\mathcal{S}$
  (1) $\langle t'' \xrightarrow{c''} t, c, \kappa \rangle = \mathcal{S}(\mathcal{E}, \mathtt{e})$
  (2) $\langle t', c', \kappa' \rangle = \mathcal{S}(\mathcal{E}, \mathtt{e}')$

  From (1)(2), by Lemma 1
  (3) $t'' \xrightarrow{c''} t$ only includes fresh effect variables
  (4) t' only includes fresh effect variables

  From (3)
  (5) t'' only includes fresh effect variables

  From (1)(2), by induction
  (6) κ and κ' satisfy the lemma

  From (6)(4)(5), by the definition of *Eff*
  $\kappa \cup \kappa' \cup \mathit{Eff}(t' \leq t'')$ satisfies the lemma $\square$

Proof of Theorem 3

Theorem 3 (Soundness) Given an expression $\mathtt{e}$ and its type environment $\mathcal{E}$, if $\mathcal{S}(\mathcal{E}, \mathtt{e}) = \langle t, c, \kappa \rangle$, then for any effect model μ of κ, one has :

$$\mu\mathcal{E} \vdash \mathtt{e} : \mu t, \mu c$$

Proof By induction on the structure of expressions

- Case of (var)

The hypotheses are
(1) $\mathcal{S}(\mathcal{E}, \mathtt{x}) = \langle t, \emptyset, \mathit{Eff}(t' \leq t)\rangle$
(2) $\mu \models \mathit{Eff}(t' \leq t)$

From (1), by the definition of $\mathcal{S}$
(3) $t' = \mathcal{E}(\mathtt{x})$, i.e. $\mu\mathcal{E}(\mathtt{x}) = \mu t'$

From (3), by (var) rule in the static semantics
(4) $\mu\mathcal{E} \vdash \mathtt{x} : \mu t', \emptyset$

From (2), by Lemma 3
(5) $\mu t' \leq \mu t$

From (4)(5), by the (sub) rule in the static semantics
$\mu\mathcal{E} \vdash \mathtt{x} : \mu t, \emptyset$

- Case of (abs)

The hypotheses are
(1) $\mathcal{S}(\mathcal{E}, (\lambda_{\mathtt{n}}\ (\mathtt{x} : \tau)\ \mathtt{e})) =$
$\langle t' \xrightarrow{\zeta} t, \emptyset, \kappa \cup \{\zeta \supseteq \{\mathtt{n}\} \cup c\}\rangle$
(2) $\mu \models \kappa \cup \{\zeta \supseteq \{\mathtt{n}\} \cup c\}$
where $t' = \mathit{New}(\tau)$ and ζ *new*

From (1), by the definition of $\mathcal{S}$
(3) $\langle t, c, \kappa\rangle = \mathcal{S}(\mathcal{E}[\mathtt{x} \mapsto t'], \mathtt{e})$

From (2), by the definition of effect models
(4) $\mu \models \kappa$
(5) $\mu \models \{\zeta \supseteq \{\mathtt{n}\} \cup c\}$ i.e. $\mu\zeta \supseteq \mu(\{\mathtt{n}\} \cup c)$

From (3)(4), by induction
(6) $\mu(\mathcal{E}[\mathtt{x} \mapsto t']) \vdash \mathtt{e} : \mu t, \mu c$

From (6), by (abs) in the static semantics
(7) $\mu\mathcal{E} \vdash (\lambda_{\mathtt{n}}\ (\mathtt{x})\ \mathtt{e}) : \mu(t' \xrightarrow{\{\mathtt{n}\}\cup c} t), \emptyset$

From (5), by the definition of subtype relation
(8) $\mu(t' \xrightarrow{\{\mathtt{n}\}\cup c} t) \leq \mu(t' \xrightarrow{\zeta} t)$

From (7)(8), by the (sub) rule in the static semantics
$\mu\mathcal{E} \vdash (\lambda_{\mathtt{n}}\ (\mathtt{x})\ \mathtt{e}) : \mu(t' \xrightarrow{\zeta} t), \emptyset$

- Case of (rec)

The hypotheses are
(1) $\mathcal{S}(\mathcal{E}, (\mathtt{rec}_{\mathtt{n}}\ (\mathtt{f} : \tau' \to \tau\ \mathtt{x} : \tau')\ \mathtt{e})) =$
$\langle t' \xrightarrow{\zeta} t, \emptyset, \kappa \cup \mathit{Eff}(t'' \leq t) \cup \{\zeta \supseteq \{\mathtt{n}\} \cup c\}\rangle$
(2) $\mu \models \kappa \cup \mathit{Eff}(t'' \leq t) \cup \{\zeta \supseteq \{\mathtt{n}\} \cup c\}$
where $t' \xrightarrow{\zeta} t = \mathit{New}(\tau' \to \tau)$

From (1), by the definition of $\mathcal{S}$
(3) $\langle t'', c, \kappa\rangle = \mathcal{S}(\mathcal{E}[\mathtt{f} \mapsto t' \xrightarrow{\zeta} t][\mathtt{x} \mapsto t'])$

From (2), by the definition of effect models
(4) $\mu \models \kappa$
(5) $\mu \models \mathit{Eff}(t'' \leq t)$
(6) $\mu \models \{\zeta \supseteq \{\mathtt{n}\} \cup c\}$, i.e. $\mu\zeta \supseteq \{\mathtt{n}\} \cup \mu c$

From (3)(4), by induction
(7) $\mu(\mathcal{E}[\mathtt{f} \mapsto t' \xrightarrow{\zeta} t][\mathtt{x} \mapsto t']) \vdash \mathtt{e} : \mu t'', \mu c$

From (5), by Lemma 3
(8) $\mu t'' \leq \mu t$

From (7)(8), by (sub) in the static semantics
(9) $\mu(\mathcal{E}[\mathtt{f} \mapsto t' \xrightarrow{\zeta} t][\mathtt{x} \mapsto t']) \vdash \mathtt{e} : \mu t, \mu c$

From (9), by (abs) in the static semantics
(10) $(\mu\mathcal{E})[\mathtt{f} \mapsto \mu(t' \xrightarrow{\zeta} t)] \vdash (\lambda_{\mathtt{n}}\ (\mathtt{x})\ \mathtt{e}) : \mu(t' \xrightarrow{\{\mathtt{n}\}\cup c} t), \emptyset$

From (6), by the definition of subtype relation
(11) $\mu(t' \xrightarrow{\{\mathtt{n}\}\cup c} t) \leq \mu(t' \xrightarrow{\zeta} t)$

From (10)(11), by the (sub) rule in the static semantics
(12) $(\mu\mathcal{E})[\mathtt{f} \mapsto \mu(t' \xrightarrow{\zeta} t)] \vdash (\lambda_{\mathtt{n}}\ (\mathtt{x})\ \mathtt{e}) : \mu(t' \xrightarrow{\zeta} t), \emptyset$

From (12), by (rec) rule in the static semantics
$\mu\mathcal{E} \vdash (\mathtt{rec}_{\mathtt{n}}\ (\mathtt{f}\ \mathtt{x})\ \mathtt{e}) : \mu(t' \xrightarrow{\zeta} t), \emptyset$

- Case of (app)

The hypotheses are
(1) $\mathcal{S}(\mathcal{E}, (\mathtt{e}\ \mathtt{e}')) = \langle t, c \cup c' \cup c'', \kappa \cup \kappa' \cup \mathit{Eff}(t' \leq t'')\rangle$
(2) $\mu \models \kappa \cup \kappa' \cup \mathit{Eff}(t' \leq t'')$

From (1), by the definition of $\mathcal{S}$
(3) $\mathcal{S}(\mathcal{E}, \mathtt{e}) = \langle t'' \xrightarrow{c''} t, c, \kappa\rangle$
(4) $\mathcal{S}(\mathcal{E}, \mathtt{e}') = \langle t', c', \kappa'\rangle$

From (2), by the definition of effect models
(5) $\mu \models \kappa$
(6) $\mu \models \kappa'$
(7) $\mu \models \mathit{Eff}(t' \leq t'')$

From (3)(5) and (4)(6), by induction
(8) $\mu\mathcal{E} \vdash \mathtt{e} : \mu(t'' \xrightarrow{c''} t), \mu c$
(9) $\mu\mathcal{E} \vdash \mathtt{e}' : \mu t', \mu c'$

From (7), by Lemma 3
(10) $\mu t' \leq \mu t''$

From (9)(10), by the (sub) rule in the static semantics
(11) $\mu\mathcal{E} \vdash \mathtt{e}' : \mu t'', \mu c'$

From (8)(11), by (app) in the static semantics
$\mu\mathcal{E} \vdash (\mathtt{e}\ \mathtt{e}') : \mu t, \mu(c \cup c' \cup c'')$ □

Proof of Theorem 4

Theorem 4 (Completeness) If $\theta_1\mathcal{E} \vdash \mathtt{e} : t_1, c_1$, then $\mathcal{S}(\mathcal{E}, \mathtt{e}) = \langle t, c, \kappa\rangle$ and there exists a effect model μ of κ, such that:

$$\theta_1\mathcal{E} = \mu\mathcal{E} \quad and \quad \mu t \leq t_1 \quad and \quad c_1 \supseteq \mu c$$

Proof By induction on the structure of expressions

- Case of (var)

  The hypothesis is
  $\theta_1\mathcal{E} \vdash \mathtt{x} : t_1, \emptyset$

  By the (var) and (sub) rules in the static semantics
  (1) $t_1' = \mathcal{E}(\mathtt{x})$
  (2) $\theta_1 t_1' \leq t_1$

  From (1), by the definition of $\mathcal{S}$
  $\mathcal{S}(\mathcal{E}, \mathtt{x}) = \langle t, \emptyset, \mathit{Eff}(t_1' \leq t)\rangle$
  where $t = \mathit{New}(\mathit{Struct}(t_1'))$

  Since t only includes fresh effect variables, we can defined θ such that:
  (3) $\theta t = t_1$

  We define the effect model μ, such that :

$$\mu v = \begin{cases} \theta v & v \in \mathit{fv}(t) \\ \theta_1 v & \mathit{otherwise} \end{cases}$$

  Note that since t only includes fresh effect variables, μ is well defined.

  From (2)(3), by the definition of μ
  (4) $\mu t_1' = \theta_1 t_1' \leq t_1$
  (5) $\mu t = \theta t = t_1$

  From (4)(5), by Lemma 3
  $\mu \models \mathit{Eff}(t_1' \leq t)$

  By the definition of μ
  $\theta_1\mathcal{E} = \mu\mathcal{E}$

  From (5)
  $\mu t \leq t_1$

- Case of (abs)

  The hypothesis is
  $\theta_1\mathcal{E} \vdash (\lambda_{\mathtt{n}}\ (\mathtt{x})\ \mathtt{e}) : t_2' \xrightarrow{c_2} t_2, \emptyset$

  By the (abs) and (sub) rules in the static semantics
  (1) $\theta_1\mathcal{E} \vdash (\lambda_{\mathtt{n}}\ (\mathtt{x})\ \mathtt{e}) : t_1' \xrightarrow{\{\mathtt{n}\}\cup c_1} t_1, \emptyset$
  (2) $t_1' \xrightarrow{\{\mathtt{n}\}\cup c_1} t_1 \leq t_2' \xrightarrow{c_2} t_2$

  From (1), by (abs) rule in the static semantics
  (3) $(\theta_1\mathcal{E})[\mathtt{x} \mapsto t_1'] \vdash \mathtt{e} : t_1, c_1$

  If $\mathtt{x}$ has classical type τ, let $t' = \mathit{New}(\tau)$.
  Then there exists a substitution θ, such that:
  (4) $t_1' = \theta t'$

  We define a substitution θ_1', such that :

$$\theta_1' v = \begin{cases} \theta v & v \in \mathit{fv}(t') \\ \theta_1 v & \mathit{otherwise} \end{cases}$$

  Note that since t' only includes fresh effect variables, θ_1' is well defined.

  From (4), by the definition of θ_1', (3) is equivalent to :
  (5) $\theta_1'(\mathcal{E}[\mathtt{x} \mapsto t']) \vdash \mathtt{e} : t_1, c_1$

  From (5), by induction
  (6) $\mathcal{S}(\mathcal{E}[\mathtt{x} \mapsto t'], \mathtt{e}) = \langle t, c, \kappa\rangle$
  there exists μ, such that :
  (7) $\mu \models \kappa$
  (8) $\theta_1'(\mathcal{E}[\mathtt{x} \mapsto t']) = \mu(\mathcal{E}[\mathtt{x} \mapsto t'])$
  (9) $\mu t \leq t_1$
  (10) $c_1 \supseteq \mu c$

  From (8)(4), by the definition of θ_1'
  (11) $\theta_1\mathcal{E} = \mu\mathcal{E}$, except on $\mathtt{x}$ which doesn't appear in $\mathcal{E}$ (alpha-renaming)
  (12) $t_1' = \mu t'$

  From (6), since $t' = \mathit{New}(\tau)$, by the definition of $\mathcal{S}$
  (13) $\mathcal{S}(\mathcal{E}, \lambda_{\mathtt{n}}\ (\mathtt{x} : \tau)\ \mathtt{e}) = \langle t' \xrightarrow{\zeta} t, \emptyset, \kappa \cup \{\zeta \supseteq \{\mathtt{n}\} \cup c\}\rangle$
  where ζ *new*

  We define an effect substitution μ' on $\mathit{fv}(\mathcal{E}, t', t, c, \kappa)$ and ζ, such that :

$$\mu' v = \begin{cases} \mu v & v \in \mathit{fv}(\mathcal{E}, t', t, c, \kappa) \\ c_2 & v = \zeta \end{cases}$$

  Note that since ζ is fresh, μ' is well defined.

  From (7), by the definition of μ'
  (14) $\mu' \models \kappa$

  By the definition of μ'
  (15) $\mu'\zeta = c_2$
  (16) $\mu'(\{\mathtt{n}\} \cup c) = \{\mathtt{n}\} \cup \mu c$

  From (10)(16), by the definition of μ'
  (17) $\{\mathtt{n}\} \cup c_1 \supseteq \mu'(\{\mathtt{n}\} \cup c)$

  From (2), by the definition of the subtype relation
  (18) $t_1 \leq t_2$
  (19) $t_2' \leq t_1'$
  (20) $c_2 \supseteq \{\mathtt{n}\} \cup c_1$

  From (20)(15)(17), by the definition of effect models
  (21) $\mu' \models \{\zeta \supseteq \{\mathtt{n}\} \cup c\}$

  From (14)(21), by the definition of effect models
  $\mu' \models \kappa \cup \{\zeta \supseteq \{\mathtt{n}\} \cup c\}$

  From (11)(12), by the definition of μ'
  $\theta_1\mathcal{E} = \mu'\mathcal{E}$

(22) $t'_1 = \mu' t'$

From (9), by the definition of μ'
(23) $\mu' t \leq t_1$

From (22)(23)(15), by the definition of subtype relation
(24) $\mu'(t' \xrightarrow{\zeta} t) \leq t'_1 \xrightarrow{c_2} t_1$

From (18)(19), by the definition of subtype relation
(25) $t'_1 \xrightarrow{c_2} t_1 \leq t'_2 \xrightarrow{c_2} t_2$

From (24)(25)
$\mu'(t' \xrightarrow{\zeta} t) \leq t'_2 \xrightarrow{c_2} t_2$

- Case of (*rec*)
The hypothesis is
$\theta_1 \mathcal{E} \vdash (\mathtt{rec_n}\ (\mathtt{f}\ \mathtt{x})\ \mathtt{e}) : t'_2 \xrightarrow{c_2} t_2, \emptyset$

By the (*rec*) and (*sub*) rules in the static semantics
(1) $\theta_1 \mathcal{E} \vdash (\mathtt{rec_n}\ (\mathtt{f}\ \mathtt{x})\ \mathtt{e}) : t'_1 \xrightarrow{\{\mathtt{n}\} \cup c_1} t_1, \emptyset$
(2) $t'_1 \xrightarrow{\{\mathtt{n}\} \cup c_1} t_1 \leq t'_2 \xrightarrow{c_2} t_2$

From (1), by the (*rec*) rule in the static semantics
(3) $(\theta_1 \mathcal{E})[\mathtt{f} \mapsto t'_1 \xrightarrow{\{\mathtt{n}\} \cup c_1} t_1][\mathtt{x} \mapsto t'_1] \vdash \mathtt{e} : t_1, c_1$

If $\mathtt{f}$ and $\mathtt{x}$ have classical types $\tau' \rightarrow \tau$ and τ' respectively, let $(t' \xrightarrow{\zeta} t) = New(\tau' \rightarrow \tau)$.
Then, there exists a substitution θ, such that:
(4) $t'_1 \xrightarrow{\{\mathtt{n}\} \cup c_1} t_1 = \theta(t' \xrightarrow{\zeta} t)$

We define a substitution on effect variables θ'_1, such that :

$$\theta'_1 v = \begin{cases} \theta v & v \in fv(t' \xrightarrow{\zeta} t) \\ \theta_1 v & otherwise \end{cases}$$

Note that since $t' \xrightarrow{\zeta} t$ only includes fresh effect variables, θ'_1 is well defined.

From (4), by the definition of θ'_1, (3) is equivalent to :
(5) $\theta'_1(\mathcal{E}[\mathtt{f} \mapsto t' \xrightarrow{\zeta} t][\mathtt{x} \mapsto t']) \vdash \mathtt{e} : t_1, c_1$

From (5), by induction
(6) $\mathcal{S}(\mathcal{E}[\mathtt{f} \mapsto t' \xrightarrow{\zeta} t][\mathtt{x} \mapsto t'], \mathtt{e}) = \langle t'', c, \kappa \rangle$
there exists μ, such that :
(7) $\mu \models \kappa$
(8) $\theta'_1(\mathcal{E}[\mathtt{f} \mapsto t' \xrightarrow{\zeta} t][\mathtt{x} \mapsto t']) =$
$\mu(\mathcal{E}[\mathtt{f} \mapsto t' \xrightarrow{\zeta} t][\mathtt{x} \mapsto t'])$
(9) $\mu t'' \leq t_1$
(10) $c_1 \supseteq \mu c$

From (6), since $t' \xrightarrow{\zeta} t = New(\tau' \rightarrow \tau)$, by the definition of $\mathcal{S}$
(11) $\mathcal{S}(\mathcal{E}, (\mathtt{rec_n}\ (\mathtt{f} : \tau' \rightarrow \tau\ \mathtt{x} : \tau')\ \mathtt{e})) =$
$\langle t' \xrightarrow{\zeta} t, \emptyset, \kappa \cup \mathit{Eff}(t'' \leq t) \cup \{\zeta \supseteq \{\mathtt{n}\} \cup c\} \rangle$

From (8)(4), by the definition of θ'_1, except on $\mathtt{f}$ and $\mathtt{x}$ which don't occur in $\mathcal{E}$ (alpha-renaming)
$\theta_1 \mathcal{E} = \mu \mathcal{E}$
(12) $t'_1 \xrightarrow{\{\mathtt{n}\} \cup c_1} t_1 = \mu(t' \xrightarrow{\zeta} t)$
(13) $t_1 = \mu t$
(14) $\{\mathtt{n}\} \cup c_1 = \mu \zeta$

From (9)(13), by Lemma 3
(15) $\mu \models \mathit{Eff}(t'' \leq t)$

From (14)(10), by the definition of effect models
(16) $\mu \models \{\zeta \supseteq \{\mathtt{n}\} \cup c\}$

From (7)(15)(16), by the definition of effect models
$\mu \models \kappa \cup \mathit{Eff}(t'' \leq t) \cup \{\zeta \supseteq \{\mathtt{n}\} \cup c\}$

From (12)(2)
$\mu(t' \xrightarrow{\zeta} t) \leq t'_2 \xrightarrow{c_2} t_2$

- Case of (*app*)
The hypotheses is
$\theta_1 \mathcal{E} \vdash (\mathtt{e}\ \mathtt{e'}) : t_2, c_1 \cup c'_1 \cup c''_1$

By the (*app*) and (*sub*) rules in the static semantics
(1) $\theta_1 \mathcal{E} \vdash (\mathtt{e}\ \mathtt{e'}) : t_1, c_1 \cup c'_1 \cup c''_1$
(2) $t_1 \leq t_2$

From (1), by the (*app*) rule in the static semantics
(3) $\theta_1 \mathcal{E} \vdash \mathtt{e} : t'_1 \xrightarrow{c''_1} t_1, c_1$
(4) $\theta_1 \mathcal{E} \vdash \mathtt{e'} : t'_1, c'_1$

From (3), by induction
(5) $\mathcal{S}(\mathcal{E}, \mathtt{e}) = \langle t'' \xrightarrow{c''} t, c, \kappa \rangle$
there exists μ, such that :
(6) $\mu \models \kappa$
(7) $\theta_1 \mathcal{E} = \mu \mathcal{E}$
(8) $\mu(t'' \xrightarrow{c''} t) \leq t'_1 \xrightarrow{c''_1} t_1$
(9) $c_1 \supseteq \mu c$

From (4), by induction
(10) $\mathcal{S}(\mathcal{E}, \mathtt{e'}) = \langle t', c', \kappa \rangle$
$\exists \mu'$, such that :
(11) $\mu' \models \kappa'$
(12) $\theta_1 \mathcal{E} = \mu' \mathcal{E}$
(13) $\mu' t' \leq t'_1$
(14) $c'_1 \supseteq \mu' c'$

From (5)(10), by the definition of $\mathcal{S}$
$\mathcal{S}(\mathcal{E}, (\mathtt{e}\ \mathtt{e'})) = \langle t, c \cup c' \cup c'', \kappa \cup \kappa' \cup \mathit{Eff}(t' \leq t'') \rangle$

We define a substitution μ'' on $fv(\mathcal{E}, t'' \xrightarrow{c''} t, c, \kappa)$, and $fv(\mathcal{E}, t', c', \kappa')$

$$\mu'' v = \begin{cases} \mu v & v \in fv(\mathcal{E}, t'' \xrightarrow{c''} t, c, \kappa) \\ \mu' v & v \in fv(\mathcal{E}, t', c', \kappa') \end{cases}$$

Note that if $v \in fv(\mathcal{E}, t', c', \kappa') \cap fv(\mathcal{E}, t'' \xrightarrow{c''} t, c, \kappa)$, then, by the definition of $\mathcal{S}$, $v \in fv(\mathcal{E})$ and thus, by (7)(12), $\mu v = \mu' v$; thus μ'' is well defined.

From (6)(11), by the definition of μ''
(16) $\mu'' \models \kappa$
(17) $\mu'' \models \kappa'$

From (8), by the definition of μ''
(18) $\mu''(t'' \xrightarrow{c''} t) = \mu(t'' \xrightarrow{c''} t) \leq t_1' \xrightarrow{c_1''} t_1$

From (18), by the definition of the subtype relation
(19) $\mu'' t \leq t_1$
(20) $t_1' \leq \mu'' t''$
(21) $c_1'' \supseteq \mu'' c''$

From (13), by the definition of μ''
(22) $\mu'' t' = \mu' t' \leq t_1'$

From (20)(22), by Lemma 3
(23) $\mu'' \models \mathit{Eff}(t' \leq t'')$

From (16)(17)(23), by the definition of effect models
$\mu'' \models \kappa \cup \kappa' \cup \mathit{Eff}(t' \leq t'')$

From (19)(2)
$\mu'' t \leq t_2$

From (9)(14)(21), by the definition of μ''
$c_1 \cup c_1' \cup c_1'' \supseteq \mu''(c \cup c' \cup c'')$ $\square$