Homework 3: Model Theory of the Typed Lambda Calculus

Consider this homework to be "due" on October 29.

1. (The following is the corrected form of Gunter's exercise 1.22.) Let $\mathcal{A} = (\{D^s\}, \{A^{s,t}\}, \mathcal{A}[\![\cdot]\!])$ and $\mathcal{B} = (\{E^s\}, \{B^{s,t}\}, \mathcal{B}[\![\cdot]\!])$ be type frames. A family $\{R^s\}$ of relations indexed over type expressions s is a *logical relation* between $\mathcal{A}$ and $\mathcal{B}$ if and only if:

   - $R^s \subseteq D^s \times E^s$, and
   - $f\ R^{s\to t}\ g$ if and only if $x\ R^s\ y$ implies $A^{s,t}(f,x)\ R^t\ B^{s,t}(g,y)$.

   (a) Show that a partial homomorphism is a special case of a logical realtion.

   (b) State and prove a version of lemma 1.17 that is appropriate to logical relations.

2. The following problem concerns the generalization of the typed lambda calculus to subtypes considered in class.

   (a) For historical reasons, in this part types are called sorts, and subtypes are called subsorts. A *signature*, Σ, consists of

   - a preordered set *SORTS* of *sorts*, a subset $V \subseteq \mathit{SORTS}$ of visible sorts such that for all $T \in V$, if $S \leq T$, then $S = T$,
   - a family *OPS* of *operation symbols*, indexed by their rank,
   - A monotonic function $\mathit{ResSort}\colon \mathit{OPS}, \mathit{SORTS}^* \to \mathit{SORTS}$, which returns an upper bound on the result sort of an operation applied to a tuple of arguments with the given sorts.

   This essentially follows Reynolds [Rey80, Page 217]. The visible sorts (e.g., `Bool`) are used below to define observations. Since *ResSort* is total, in general there will be a sort `NS` (i.e., nonsense) that is a supersort of all sorts in *SORTS*, so that if $\mathit{ResSort}(g, \langle T_1, T_2\rangle) =$ `NS`, one can consider that g is undefined on arguments of sort T_1 and T_2.

   Let Σ be a signature. A Σ-*algebra* $A = (|A|, \mathit{OPS}^A)$ consists of:

   - a *carrier*, $|A|$, which is an *SORTS*-indexed family of sets,
   - for each $n \geq 0$ and $g \in \mathit{OPS}_n$, an interpretation g^A such that for each tuple of types $\vec{S}$ and for each tuple $\vec{q} \in \vec{S}^A$, there is some sort $U \leq \mathit{ResSort}(g, \vec{S})$ such that $g^A(\vec{q}) \in U^A$.

   This is a slight generalization from Reynolds's category sorted algebras, in that the information provided by *ResSort* is regarded as an upper bound and not as giving exact information about the result sort. This generalization is appropriate for modeling object-oriented programs, where the declared sort of an operation is regarded as an upper bound.

   Let C and A be Σ-algebras. A *SORTS*-indexed family

$$\mathcal{R} = \left\{ \mathcal{R}_T \subseteq \left(\left(\bigcup_{S \leq T} S^C \right) \times \left(\bigcup_{S \leq T} S^A \right) \right) \middle| \, T \in \mathit{SORTS} \right\},$$

$$\text{[ident]} \qquad \Sigma; H, x : T, H' \vdash x : T$$

$$\text{[op-call]} \quad \frac{\Sigma; H \vdash \vec{e} : \vec{S}, \;\; \Sigma \vdash \mathit{ResSort}(g, \vec{S}) = T}{\Sigma; H \vdash g(\vec{e}) : T}$$

Figure 1: Type inference rules for operation calls

is a Σ-*homomorphic relation between* C *and* A, if and only if the following properties hold:

Substitution: for all sorts T, for all tuples of sorts $\vec{S}$, $\vec{U}$, and $\vec{W}$ such that $\vec{U} \leq \vec{S}$ and $\vec{W} \leq \vec{S}$, for all tuples $\vec{q} \in \vec{U}^C$, for all tuples $\vec{r} \in \vec{W}^A$ such that $\vec{q}\, \mathcal{R}_{\vec{S}}\, \vec{r}$, for all operation symbols $g \in \mathit{OPS}$ such that $\mathit{ResSort}(g, \vec{S}) = T$, $g^C(\vec{q})\, \mathcal{R}_T\, g^A(\vec{r})$,

Subsorting: for all sorts S and T, $(S \leq T) \Rightarrow (\mathcal{R}_S \subseteq \mathcal{R}_T)$,

Notice that the substitution property relates the results of an operation at the expected result sort (using *ResSort*). The idea is that during program verification, one reasons based on static sort information, as reflected in *ResSort*, rather than the sorts of objects used at run-time.

To state the fundamental property of such relations we use the following simplistic expression language:

$$e ::= x \mid g(e, \ldots, e)$$

where x is an identifier, g is an element of *OPS*, and $e, \ldots, e$ is a sequence of zero or more e's. The sort-checking rules for this grammar are given in Figure 1. In the rules H is a sort context (a list of identifier-sort expression pairs). Axioms of the form $\Sigma \vdash \mathit{ResSort}(g, \vec{S}) = T$, which describe the *ResSort* component of Σ, are omitted.

As in object-oriented programs, semantic environments are permitted to assign a meaning to an identifier $x : T$ that is an element of the carrier set of some $S \leq T$. Let H be a sort context. An *H-environment* ρ over an algebra A has the property for all sorts T and for all pairs $x : T$ in H, there is some sort $U \leq T$, such that $\rho(x) \in A_U$.

Homomorphic relations are extended to environments pointwise. Let H be type context and let ρ and ρ' be H-environments. Then $\rho\, \mathcal{R}\, \rho'$ if and only if for all $x : S$ in H, $\rho(x)\, \mathcal{R}_S\, \rho'(x)$.

Define the meaning of an expression e such that $H \vdash e : T$ in an H-environment ρ over A.

(b) State a prove a theorem that says that the meanings of an expression in related environmetns are related.

(c) The notion that one object "acts like" another is formalized in two ways below: using observations and using a special case of homomorphic relations. It is shown that the relational formulation implies the formalization that uses observations. This theorem can be summarized by the slogan: "subsorting means no surprises."

Observations are defined using the visible sorts of Σ. An *H-observation* is a term e such that $H \vdash e : v$, where $v \in V$. In what follows we assume that each algebra contains a fixed subalgebra the defines the meaning of the visible types.

To say that the result of an observation is surprising, one needs a notion of what results are "expected." An expected result is one that is permitted by the specification of the data types involved. For brevity, we consider the semantics of specifications, as opposed to their presentations. Let $SPEC$ be a set of Σ-algebras, which represents the "loose" semantics of a specification.

We also want to reason about programs based on nominal sort information, for example, if `s` : `Set` then the expected results for an observation involving `s` should be based on the specification of `Set`. This idea can be formalized using the semantics of specifications, by considering environments that do not exploit subsorting.

An H-environment over A, ρ is *nominal* if and only if for all $x : S$ in H, $\rho(x) \in S^A$.

The set of expected results of an observation e for $SPEC$ is the set of meanings of e in nominal environments over algebras of $SPEC$.

To show that no surprises occur in non-nominal environments, one needs something stronger than a homomorphic relation. A Σ-homomorphic relation between C and A is a Σ-*simulation relation* between C and A if and only if the following two conditions are satisfied:

Coercion: for all sorts S and T: $(S \leq T) \Rightarrow (\forall (q \in S^C) \exists (r \in T^A)\, q\, \mathcal{R}_T\, r)$, and

V-identical: for each $v \in V$, $\mathcal{R}_v$ is the identity relation on the carrier set of v (which is the same in both C and A).

Prove or disprove the following lemma. Let Σ be a signature. Let $SPEC$ be a set of Σ-algebras. Let H be a type context. Suppose that for each $C \in SPEC$ there is some $A \in SPEC$ such that there is a Σ-simulation between C and A. Then for all $C \in SPEC$, there is some $A \in SPEC$ such that for all H-environments ρ_C over C, there is some nominal H-environment ρ_A such that $\rho_C\, \mathcal{R}\, \rho_A$.

(d) State and prove a theorem that, given the hypothesis of the above lemma, says that there are no surprises that result from the use of non-nominal envrionments.

(e) In this part we start to extend the above to a typed lambda calculus. Throughout this section, let Σ be a signature, as above. The style of presentation below roughly follows [Gun90] and [Fri75].

The abstract syntax of *type expressions over* Σ is given by the following grammar

$$T ::= C \mid T \rightarrow T$$

where C is an element of $SORTS$. The preorder $\leq$ on the sorts of Σ is extended to a preorder on type expressions using the usual anti-monotonic rule for function types [Car84]

The abstract syntax of *pre-terms over* Σ is given by the following grammar:

$$e ::= x \mid g(e, \ldots, e) \mid \lambda x : T.e \mid ee$$

where x is an identifier and g is an element of OPS. Note that the operations from Σ are not first-class objects; this simplification avoids some type-checking problems. A *term over* Σ is a pre-term that has a type, according to the rules in Figures 1 and 2. Subtyping is only used for applications, hence each term has exactly one type. Axioms of the form $\Sigma \vdash S \leq T$ that describe the preorder $\leq$ on the $SORTS$ of Σ are omitted.

A Σ-*pre-frame* is a pair (D, A), where

- D is a family of sets indexed by type expressions, so that D_T is the interpretation of type T,

$$\text{[appl]} \quad \frac{\Sigma; H \vdash e_0 : S \to T,\ \ \Sigma; H \vdash e_1 : U,\ \ \Sigma \vdash U \leq S}{\Sigma; H \vdash (e_0\, e_1) : T}$$

$$\text{[abstr]} \quad \frac{\Sigma; H, x : S \vdash e : T}{\Sigma; H \vdash (\lambda x : S\, .\, e) : S \to T}$$

$$\text{[fun-}\leq\text{]} \quad \frac{\Sigma \vdash S_1 \leq T_1,\ \ \Sigma \vdash T_2 \leq S_2}{\Sigma \vdash (T_1 \to T_2) \leq (S_1 \to S_2)}$$

Figure 2: Type inference and subtyping rules

- A is a family of functions indexed by pairs of type expressions or by operation symbols.
  - At indexes that are pairs of type expressions,

    $$A_{S,T} : (D_{S \to T} \times (\bigcup_{W \leq S} D_W)) \to \bigcup_{U \leq T} D_U$$

    describes the application of an element of $D_{S \to T}$ to an element of $\bigcup_{W \leq S} D_W$ such that the following *extensionality* property holds: for all $f_1, f_2 \in D_{S \to T}$, if for all $W \leq S$ and $w \in D_W$ $A_{S,T}(f_1, w) = A_{S,T}(f_2, w)$, then $f_1 = f_2$.
  - At indexes that are operation symbols, for each $g \in \mathit{OPS}$, A_g is such that for each tuple of types $\vec{S}$ and for each tuple $\vec{q} \in D_{\vec{S}}$, there is some sort $U \leq \mathit{ResSort}(g, \vec{S})$ such that $A_g(\vec{q}) \in D_U$.

Therefore, given a Σ-pre-frame (D, A), one can extract a Σ-algebra $(D, A)_{(\Sigma)}$, where the carrier set of each sort T is D_T, and the interpretation of each $g \in \mathit{OPS}$ is A_g.

An H-environment, as above, may map $x : S$ to D_W for some $W \leq S$.

Define an appropriate notion of a Σ-*type frame.*

(f) Let $A = (D, A, \mathcal{A}[\![\cdot]\!])$ and $B = (E, B, \mathcal{B}[\![\cdot]\!])$ be Σ-type frames. A family of binary relations indexed by type expressions

$$\mathcal{R} = \left\{ \mathcal{R}_T \subseteq ((\bigcup_{U \leq T} D_T) \times (\bigcup_{U \leq T} E_T)) \right\},$$

is a Σ-*logical relation with subtypes between A and B* if and only if the following properties hold:

substitution: the restriction of $\mathcal{R}$ to a sorted family of relations between the Σ-algebras $A_{(\Sigma)}$ and $B_{(\Sigma)}$ satisfies the substitution property of a Σ-simulation relation,

logical: for all type expressions S and T:

$$f\, \mathcal{R}_{S \to T}\, g \Leftrightarrow ((x\, \mathcal{R}_S\, y) \Rightarrow A_{S,T}(f, x)\, \mathcal{R}_T\, B_{S,T}(g, y))$$

subtyping: for all type expressions S and T: $(S \leq T) \Rightarrow (\mathcal{R}_S \subseteq \mathcal{R}_T)$,

Prove the following theorem (the fundamental property of logical relations with subtypes).

Let $A = (D, A, \mathcal{A}[\![\cdot]\!])$ and $B = (E, B, \mathcal{B}[\![\cdot]\!])$ be Σ-type frames. Let $\mathcal{R}$ be a Σ-logical relation with subtypes between A and B. Let H be a type context. Let e be a term over Σ and let T be a type expression such that $\Sigma; H \vdash e : T$. For all H-environments ρ_A and ρ_B,

$$(\rho_A \,\mathcal{R}\, \rho_B) \Rightarrow (\mathcal{A}[\![H \rhd e : T]\!]\rho_A) \,\mathcal{R}_T\, (\mathcal{B}[\![H \rhd e : T]\!]\rho_B).$$

(g) The "subtyping means no surprises" slogan also holds in this setting. To show this, define an appropriate generalization of simulation relations that generalize Σ-logical relations with subtypes, and state and prove the appropriate generalization of the lemma that if there is always a "logical simulation" from any type frame, then there is always some nominal environment that a given envrionment simulates.

(h) (Extra credit) Point out any flaws in the above analysis of subtyping.

References

[Car84] Luca Cardelli. A Semantics of Multiple Inheritance. In D. B. MacQueen G. Kahn and G. Plotkin, editors, *Semantics of Data Types: International Symposium, Sophia-Antipolis, France*, volume 173 of *Lecture Notes in Computer Science*, pages 51–66. Springer-Verlag, New York, N.Y., June 1984. A revised version of this paper appears in *Information and Computation*, volume 76, numbers 2/3, pages 138–164, February/March 1988.

[Fri75] H. Friedman. Equality between Functionals. In R. Parikh, editor, *Logic Colloquium, Symposium on Logic Held at Boston, 1972-73*, volume 453 of *Lecture Notes in Mathematics*, pages 22–37. Springer-Verlag, New York, N.Y., 1975.

[Gun90] Carl A. Gunter. Structures and Techniques for the Semantics of Programming Languages. Technical report, Department of Computer and Information Science, University of Pennsylvania, September 1990.

[Rey80] John C. Reynolds. Using Category Theory to Design Implicit Conversions and Generic Operators. In Neil D. Jones, editor, *Semantics-Directed Compiler Generation, Proceedings of a Workshop, Aarhus, Denmark*, volume 94 of *Lecture Notes in Computer Science*, pages 211–258. Springer-Verlag, January 1980.