Bachelor Project in Computer Science

Danny Nygård Hansen

16th September 2023

If X and A are sets, then we write $A \subseteq_\kappa X$ if A is a subset of X with cardinality strictly less than κ. We denote the power set of X by $\mathcal{P}(X)$, and we furthermore write $\mathcal{P}_\kappa(X)$ for the collection of sets A with $A \subseteq_\kappa X$. We do not distinguish between the ordinal ω and the cardinal $\aleph_0$, so we write $A \subseteq_\omega X$ if A is a finite subset of X, and $\mathcal{P}_\omega(X)$ similarly denotes the set of finite subsets of X. If Y is another set, then we denote the set of partial maps from X to Y by $X \rightharpoonup Y$. For $f \in X \rightharpoonup Y$ we let $\operatorname{dom} f$ and $\operatorname{ran} f$ denote the domain and range of f, respectively. We also write $X \rightharpoonup_\kappa Y$ for the subset of $X \rightharpoonup Y$ of partial maps whose domain has cardinality strictly less than κ, and elements in $X \rightharpoonup_\omega Y$ will be called ***finite partial maps***. For $x_0 \in X$ and $y_0 \in Y$ we denote by $f[x_0 \mapsto y_0]$ the partial map given by

$$f[x_0 \mapsto y_0](x) = \begin{cases} y_0, & x = x_0, \\ f(x), & x \in \operatorname{dom} f \setminus \{x_0\}. \end{cases}$$

Notice in particular that $\operatorname{dom} f[x_0 \mapsto y_0] = \operatorname{dom} f \cup \{x_0\}$, so that if $f \in X \rightharpoonup_\omega Y$, then also $f[x_0 \mapsto y_0] \in X \rightharpoonup_\omega Y$.

If $\Xi \subseteq_\omega \mathit{TVar}$, Γ is a type context and Σ is a store typing, then we say that a store σ is ***well-typed*** with respect to Ξ, Γ and Σ if $\operatorname{dom} \sigma = \operatorname{dom} \Sigma$ and $\Xi \mid \Gamma \mid \Sigma \vdash \sigma(l) : \Sigma(l)$ for all $l \in \operatorname{dom} \sigma$. In this case we write $\Xi \mid \Gamma \mid \Sigma \vdash \sigma$, and if Ξ and Γ are both empty we simply write $\Sigma \vdash \sigma$.

0.a • THEOREM: *Progress.*
Let e be an expression and τ a type of $\lambda_\rightarrow$.[1] If $\Sigma \vdash e : \tau$, then either e is a value or else, for any store σ with $\Sigma \vdash \sigma$, there exists an expression e' and a store σ' such that $(\sigma, e) \rightarrow (\sigma', e')$.

Proof. The proof is by induction on the typing relation $\Xi \mid \Gamma \mid \Sigma \vdash e : \tau$, but the claim to be proved is augmented by 'or either Γ or Ξ is non-empty'.[2] Notice

---

[1] TODO note different versions of STLC with and without products/sums etc., see wikipedia

[2] This ensures that we can perform the induction on the entire 4-ary relation, which is important since this relation is the one that is defined by the inference rules, *not* the corresponding binary relation obtained by restricting the ternary relation to the subset where Ξ and Γ are empty.

that the induction step for each inference rule is trivial if Ξ or Γ is non-empty, so we need only prove each case when Ξ and Γ are empty. [TODO mention store]

T-UNIT: Since $()$ is a value, this follows.

T-VAR: Since we may assume that Γ is empty, this implication is vacuously true.[3]

T-PAIR: Assume that the claim holds for $\Sigma \vdash e_1 : \tau_1$ and $\Sigma \vdash e_2 : \tau_2$. If both e_1 and e_2 are values, then (e_1, e_2) is also a value, so assume that only $e_1 = v_1$ is a value and that $e_2 \to e_2'$. Since the only rule that generates instances of the one-step relation $\to$ is HEAD-STEP-STEP, it follows that e_2 is on the form $K[d_2]$ and e_2' is on the form $K[d_2']$, where K is an evaluation context and d_2 and d_2' are subexpressions of e_2 and e_2' respectively such that $d_2 \to_h d_2'$. Letting $K' = (v_1, K)$ it follows that $(v_1, e_2) = K'[d_2]$ and $(v_1, e_2') = K'[d_2']$, and so

$$(v_1, e_2) = K'[d_2] \to K'[d_2'] = (v_1, e_2')$$

by HEAD-STEP-STEP. If instead e_1 is not a value, then the same argument (using the evaluation context (K, e_2)) yields the same result.

T-FST: Assume that the claim holds for $\Sigma \vdash e : \tau_1 \times \tau_2$. If e is a value, then it is on the form (v_1, v_2) [TODO canonical forms], where v_1 and v_2 are values. Hence $\mathsf{fst}\, e = \mathsf{fst}\,(v_1, v_2)$, and this reduces via a head-step to v_1. Choosing the evaluation context $K = -$, HEAD-STEP-STEP implies that $\mathsf{fst}\,(v_1, v_2) \to v_1$. If instead e is not a value, then by induction there is some e' such that $e \to e'$. Hence there are subexpressions d and d' and an evaluation context K such that $e = K[d]$, $e' = K[d']$ and $d \to_h d'$. Letting $K' = \mathsf{fst}\, K$ we have

$$\mathsf{fst}\, e = K'[d] \to K'[d'] = \mathsf{fst}\, e',$$

as desired.

T-SND: Similar to T-FST.

T-INJ1: Assume that the claim holds for $\Sigma \vdash e : \tau_1$. If e is a value v, then so is $\mathsf{inj}_1 v$. If instead $e \to e'$, then as before $e = K[d]$, $e' = K[d']$ and $d \to_h d'$. Letting $K' = \mathsf{inj}_1 K$ we get $\mathsf{inj}_1 e = K'[d]$ and $\mathsf{inj}_1 e' = K'[d']$, so $\mathsf{inj}_1 e \to \mathsf{inj}_1 e'$.

T-INJ2: Similar to T-INJ1.

T-MATCH: Assume that the claim holds for $\Sigma \vdash e_1 : \tau_1 + \tau_2$. If e_1 is a value, then by [TODO canonical forms] it must be on the form $\mathsf{inj}_1 v$ or $\mathsf{inj}_2 v$ for a value v. Hence the expression $\mathsf{match}\ e_1\ \mathsf{with}\ \mathsf{inj}_1 x {\Rightarrow} e_2 | \mathsf{inj}_2 x {\Rightarrow} e_3\ \mathsf{end}$ can reduce

---

[3]In the formalism of TODO ref T-VAR would not be an inference rule, it would instead be an axiom requiring the relation to include all quadruples (Ξ, Γ, e, τ) such that $(e : \tau) \in \Gamma$ and all type variables in τ occur in Ξ. This is of course also vacuously true when Γ is empty.

via a head step by either E-MATCH-INJ1 or E-MATCH-INJ2, so it reduces by HEAD-STEP-STEP (using the evaluation context $K = -$). If instead $e_1 \to e_1'$, then by the same argument as in previous cases with $K' = \text{match } K \text{ with } \text{inj}_1\, x \Rightarrow e_2 \mid \text{inj}_2\, x \Rightarrow e_3 \text{ end}$, it follows that $\text{match } e_1 \text{ with } \text{inj}_1\, x \Rightarrow e_2 \mid \text{inj}_2\, x \Rightarrow e_3 \text{ end}$ reduces.

T-REC: This is obvious since $\text{rec } f(x) := e$ is a value.

T-APP: Assume that the claim holds for $\Sigma \vdash e_1 : \tau_1 \to \tau_2$ and $\Sigma \vdash e_2 : \tau_1$. If e_1 is a value, then by [TODO canonical forms] it must be on the form $\text{rec } f(x) := e$. If also e_2 is a value, then the claim follows by E-REC-APP. If $e_1 = v$ is a value but e_2 is not, then $e_2 \to e_2'$. The same argument as in previous cases with $K' = v\, K$ shows that $v\, e_2$ reduces. Finally, if e_1 is not a value, then $e_1 \to e_1'$, and choosing $K' = K\, e_2$ proves the claim.

T-TLAM: This is obvious since $\Lambda\, e$ is a value.

T-TAPP: Assume that the claim holds for $\Sigma \vdash e : \forall X.\tau$. If e is a value, then by [TODO canonical forms] it must be on the form $\Lambda\, e'$, so the claim follows from E-TAPP-TLAM (via HEAD-STEP-STEP using the evaluation context $K = -$). If e is not a value, then $e \to e'$ for some expression e' by induction. These expressions then have subexpressions d and d' respectively such that $d \to_h d'$, and such that $e = K[d]$ and $e' = K[d']$ for some evaluation context K. Letting $K' = K\, \_$ we thus have $e\, \_ = K'[d]$ and $e'\, \_ = K'[d']$, proving the claim.

T-PACK: Assume that the claim holds for $\Sigma \vdash e : \tau[\tau'/X]$. If e is a value, then so is $\text{pack}\, e$. Otherwise $e \to e'$ for some expression e'. The same argument as before using the evaluation context $\text{pack}\, K$ for an appropriate K yields the claim.

T-UNPACK: Assume that the claim holds for $\Sigma \vdash e_1 : \exists X.\tau$. If e_1 is a value, then it must be on the form $\text{pack}\, v$ [TODO canonical forms], so an application of E-UNPACK-PACK yields the claim. Otherwise $e \to e'$ for some e', and we use the evaluation context $\text{unpack}\, K \text{ as } x \text{ in } e_2$ for an appropriate K.

T-FOLD: TODO

T-UNFOLD: TODO

T-LOC: Locations are values, so this is obvious.

T-ALLOC: Assume that the claim holds for $\Sigma \vdash e : \tau$. If e is a value, then the claim follows by applying E-ALLOC, noting that there always exists a location $l \notin \text{dom}\, \sigma$. Otherwise given an appropriate [TODO well-typed] store σ there is an expression e' and a store σ' such that $(\sigma, e) \to (\sigma', e')$. But then there is some evaluation context K and subexpressions d and d' of e and e' respectively, such that $e = K[d]$, $e' = K[d']$ and $(\sigma, d) \to_h (\sigma', d')$. Letting $K' = \text{ref}\, K$ we have $\text{ref}\, e = K'[d]$ and $\text{ref}\, e' = K'[d']$, so HEAD-STEP-STEP implies that $(\sigma, \text{ref}\, e) \to (\sigma', \text{ref}\, e')$.

T-STORE: Assume that the claim holds for $\Sigma \vdash e_1 : \text{ref}(\tau)$ and $\Sigma \vdash e_2 : \tau$, and let σ be a store with $\Sigma \vdash \sigma$. If e_1 is a value, then by [TODO canonical forms] it is a location l, and by [TODO lemma] we have $l \in \text{dom}\,\Sigma = \text{dom}\,\sigma$. If e_2 is also a value, then the claim follows from E-STORE. If $e_1 = l$ is a value but e_2 is not, then there is some e_2' and σ' such that $(\sigma, e_2) \to (\sigma', e_2')$. Again writing $e_2 = K[d_2]$ and $e_2' = K[d_2']$ with $(\sigma, d) \to_h (\sigma', d')$, we use the evaluation context $l := K$. Finally, if e_1 is not a value, then the same argument using instead $K := e_2$ yields the claim.

T-LOAD: Assume that the claim holds for $\Sigma \vdash e : \text{ref}(\tau)$, and let σ be a store with $\Sigma \vdash \sigma$. If e is a value, then as before it is a location l, and $l \in \text{dom}\,\sigma$. It then follows from E-LOAD that $(\sigma, !l) \to_h (\sigma, v)$, where $v = \sigma(l)$. If instead $(\sigma, e) \to (\sigma', e')$ for an expression e' and a store σ', then we simply use the evaluation context $!K$ for an appropriate K. ∎

———

Let X be a set, and let P be a predicate on X (i.e., a unary relation on – or simply a subset of – X). For $x \in X$ we write $P(x)$ to mean $x \in P$. If $f : X^k \rightharpoonup X$ is a partial function for some $k \in \mathbb{N}$, then we say that P **satisfies** the inference rule R_f if $P(x_1) \wedge \cdots \wedge P(x_k)$ implies $P(f(x_1, \ldots, x_k))$ for all $(x_1, \ldots, x_k) \in \text{dom}\,f$. We write the inference rule on the form

$$\frac{P(x_1) \quad \cdots \quad P(x_k)}{P(f(x_1, \ldots, x_k))},$$

where it is implicit that $(x_1, \ldots, x_k)$ lies in $\text{dom}\,f$.

As an example, in the simply typed lambda calculus X is the set of triples (Γ, e, τ), where Γ is a type context, e an expression and τ a type. The function f might be unary and have as domain the set of triples such that τ is a product type, say $\tau_1 \times \tau_2$, and it might then map $(\Gamma, e, \tau_1 \times \tau_2)$ to $(\Gamma, \text{snd}\,e, \tau_2)$. Of course, using the conventional notation we would write e.g. the statement $P(\Gamma, e, \tau_1 \times \tau_2)$ as $\Gamma \mid \Sigma \vdash e : \tau_1 \times \tau_2$, and $P(\Gamma, \text{snd}\,e, \tau_2)$ would be written $\Gamma \mid \Sigma \vdash \text{snd}\,e : \tau_2$, so the rule R_f would then just be the rule T-SND.

We also wish to require a given predicate to satisfy a collection of axioms. As motivation, notice that the antecedent of the rule T-VAR is not on the same form as the consequent. That is, it is not an inductive rule, and it can thus be stated as an axiom. To wit, if $(e : \tau) \in \Gamma$ then $P(\Gamma, e, \tau)$. Another way to describe this is as the set $\{(\Gamma, e, \tau) \mid (e : \tau) \in \Gamma\}$. Hence we may simply use subsets of the universal set X as axioms. Of course, if $\mathcal{A}$ is a collection of axioms, then $\bigcup \mathcal{A}$ is also an axiom. Hence if given any number [TODO set many?] of axioms, we can always take their union and assume that there is only one axiom.

Given an axiom [TODO or set of axioms?] A and a collection $\mathcal{R}$ of inference rules, we first define a function $F \colon \mathcal{P}(X) \to \mathcal{P}(X)$ by

$$F(B) = \bigcup_{R \in \mathcal{R}} f_R(B^{\rho(R)}),$$

where f_R is some partial function inducing R, and $\rho(R)$ is the arity of f_R. For $A \subseteq X$ we further let $F_A(B) = F(B) \cup A$ [TODO alternatively consider sublattice of sets $\supseteq A$]. It is clear that a set P is a fixed point of F if and only if P is closed under application of the inference rules in $\mathcal{R}$. It is furthermore easy to show that P is a fixed point of F_A if and only if P is a fixed point of F which contains A [TODO need that $B \subseteq F(B)$, maybe assume that $\mathcal{R}$ contains the identity/trivial rule?]. Now notice that $\mathcal{P}(X)$ is a complete lattice (with respect to set inclusion), and that F_A is order-preserving. The Knaster–Tarski fixed point theorem [TODO ref] then implies that F_A has a least fixed point[4]. We say that this fixed point is the predicate **induced** by $\mathcal{R}$ and A.

We would also like a way to generate the predicate P induced by $\mathcal{R}$ and A. Let $A_0 = A$, and for $n \in \mathbb{N}$ define

$$A_{n+1} = F_{A_n}(A_n) = A_n \cup \bigcup_{R \in \mathcal{R}} f_R(A_n^{\rho(R)}).$$

The sequence $(A_n)_{n \in \mathbb{N}}$ is thus increasing. Letting $P = \bigcup_{n \in \mathbb{N}} A_n$, we then have the following result:

0.B • PROPOSITION. *P is the smallest predicate on X which is true on all elements in A and which satisfies all inference rules in $\mathcal{R}$.*

Proof. Clearly P is true on A, so consider an inference rule $R \in \mathcal{R}$ and let $(x_1, \ldots, x_k) \in \operatorname{dom} f_R$ be such that $P(x_1), \ldots, P(x_k)$ hold, i.e. such that $x_1, \ldots, x_k \in P$. We must then show that also $f_R(x_1, \ldots, x_k) \in P$. Then there must be indices $n_1, \ldots, n_k$ such that $x_i \in A_{n_i}$. Letting $n = \max\{n_1, \ldots, n_k\}$ we thus have[5] $x_1, \ldots, x_k \in A_n$. Hence $f_R(x_1, \ldots, x_k) \in f_R(A_n^k)$, so by definition of A_{n+1} we have $f_R(x_1, \ldots, x_k) \in A_{n+1} \subseteq P$ as desired.

Conversely we show that P is the smallest such predicate. But this is clear since it must be contained in every predicate containing A and satisfying the inference rules. ∎

0.C • COROLLARY. *Every $x \in X$ such that $P(x)$ holds is obtained from A by applying the inference rules in $\mathcal{R}$ finitely many times. In particular there is an inference rules that was applied last.* ∎

---

[4]It also has a greatest fixed point, but this is trivially $\mathcal{P}(X)$ itself and is of no interest to us.

[5]This is where the proof fails if the arity of f_R is infinite. For instance, the proof does not go through for σ-algebras – where f_R might map a sequence $(B_n)_{n \in \mathbb{N}}$ of sets to the union $\bigcup_{n \in \mathbb{N}} B_n$ – nor for topologies.

We are of course used to these kinds of results in computer science, and we rarely stop to spell out the details. Since P is the smallest predicate on X with the above properties, we obtain a notion of structural induction or rule induction, and this is often sufficient to prove properties of P. But sometimes it is at least much more convenient to appeal to the 'last inference rule applied', as we shall see.[6]

0.d • Theorem: *Preservation.*

If

$$\Xi \mid \Gamma \mid \Sigma \vdash e : \tau, \quad \Xi \mid \Gamma \mid \Sigma \vdash \sigma \quad and \quad (\sigma, e) \to (\sigma', e'),$$

then there exists some store typing Σ' with $\Sigma \subseteq \Sigma'$ such that

$$\Xi \mid \Gamma \mid \Sigma' \vdash e' : \tau \quad and \quad \Xi \mid \Gamma \mid \Sigma' \vdash \sigma'.$$

Proof. By definition of the one-step relation, there exist an evaluation context K and subexpressions d of e and d' of e' such that $e = K[d]$, $e' = K[d']$, and $(\sigma, d) \to_h (\sigma', d')$. By Lemma 0.e there is some type ρ such that $\Xi \mid \Gamma \mid \Sigma \vdash d : \rho$. Next it follows from Lemma 0.h that $\Xi \mid \Gamma \mid \Sigma' \vdash d' : \rho$ for some store typing Σ' with $\Sigma \subseteq \Sigma'$ and $\Xi \mid \Gamma \mid \Sigma' \vdash \sigma'$. By Lemma 0.f we also have $\Xi \mid \Gamma \mid \Sigma' \vdash d : \rho$, so it follows from Lemma 0.g that $\Xi \mid \Gamma \mid \Sigma' \vdash K[d'] : \tau$ as desired. ∎

0.e • Lemma. *If K is an evaluation context, e is an expression and $\Xi \mid \Gamma \mid \Sigma \vdash K[e] : \tau$, then $\Xi \mid \Gamma \mid \Sigma \vdash e : \rho$ for some type ρ.*

Proof. Every evaluation context is obtained from the hole '$-$' by finitely many applications of the productions in the grammar [TODO prove this?]. We prove the claim by induction on the length of such a sequence of productions. If $K = -$, then the claim is obvious, since then $K[e] = e$. Hence we assume that K is obtained from some evaluation context K' by some application of a production, so that the induction hypothesis holds for K'.

---

[6]As mentioned, the proof of TODO does not go through for σ-algebras. The best we can do is apparently the following: If $\mathcal{D} = \mathcal{D}_0$ is a collection of subsets of X, let $\sigma(\mathcal{D})$ denote the smallest σ-algebra on X containing $\mathcal{D}$. If α is a countable ordinal with an immediate predecessor β, then we define $\mathcal{D}_\alpha$ by transfinite induction to be the set of countable unions of elements of $\mathcal{D}_\beta$ and complements of such sets, and otherwise we let $\mathcal{D}_\alpha = \bigcup_{\beta < \alpha} \mathcal{D}_\beta$. Then one can show that $\sigma(\mathcal{D}) = \bigcup_{\alpha \in \Omega} \mathcal{D}_\alpha$, where Ω is the set of countable ordinals. Cf. Folland TODO. Note however that we by minimality of $\sigma(\mathcal{D})$ do get a notion of induction, just as for predicates defined by inference rules.

For topologies we do have a simple characterisation of the topology $\mathcal{T}$ generated by a collection of sets $\mathcal{S}$. In this case $\mathcal{S}$ is a subbasis for $\mathcal{T}$, i.e., every set in $\mathcal{T}$ is a (generally infinite) union of finite intersections of elements in $\mathcal{S}$. But notice that infinitely many set operations (finite intersections or arbitrary unions) are still required to obtain a general open set, since we take a finite intersection infinitely many times.

On the other hand, results such as TODO ref are of course standard fare in algebra, where we obtain a similar description of the subalgebra generated by a subset of some algebra. See e.g. Bergman TODO

$K = (K', e')$: Then $K[e] = (K'[e], e')$, and since this is well-typed with type τ, the inversion lemma [TODO proof] implies that $\Xi \mid \Gamma \mid \Sigma \vdash K'[e] : \tau_1$ for some type τ_1. By induction applied to K' we have $\Xi \mid \Gamma \mid \Sigma \vdash e : \rho$ for some type ρ.

$K = (v, K')$: Similar to the above.

$K = \mathsf{fst}\, K'$: Then $K[e] = \mathsf{fst}\, K'[e]$, so the inversion lemma implies that $\Xi \mid \Gamma \mid \Sigma \vdash K'[e] : \tau \times \tau_2$ for some τ_2. By induction we have $\Xi \mid \Gamma \mid \Sigma \vdash e : \rho$ for some type ρ.

$K \in \{\mathsf{snd}\, K', \mathsf{inj}_1 K', \mathsf{inj}_2 K'\}$: Similar to the above.

$K = \mathsf{match}\, K'\, \mathsf{with}\, \mathsf{inj}_1 x \Rightarrow e_1 \mid \mathsf{inj}_2 x \Rightarrow e_2\, \mathsf{end}$: The inversion lemma here implies that $\Xi \mid \Gamma \mid \Sigma \vdash K'[e] : \tau_1 + \tau_2$, so the claim follows by induction.

$K = K'\, e'$: Then $K[e] = K'[e]\, e'$, so the inversion lemma implies that $\Xi \mid \Gamma \mid \Sigma \vdash K'[e] : \tau_1 \rightarrow \tau$ for some type τ_1. The claim follows by induction as before.

$K = v\, K'$: Similar to the above.

$K = K'\,\_$: The inversion lemma implies that $\Xi \mid \Gamma \mid \Sigma \vdash K'[e] : \tau_1$ for some type τ_1, so the claim follows by induction.

$K =$: TODO

$K = \mathsf{ref}\, K'$: Then $K[e] = \mathsf{ref}\, K'[e]$, so the inversion lemma implies that $\Xi \mid \Gamma \mid \Sigma \vdash K'[e] : \tau'$. The claim follows by induction.

$K = K' := e'$: Then $K[e] = K'[e] := e'$, so the inversion lemma implies that $\Xi \mid \Gamma \mid \Sigma \vdash K'[e] : \tau'$ ... TODO

TODO rest – but they are all the same, so maybe just do one? ∎

0.f • Lemma. *If Σ and Σ' are store typings with $\Sigma \subseteq \Sigma'$ and $\Xi \mid \Gamma \mid \Sigma \vdash e : \tau$, then $\Xi \mid \Gamma \mid \Sigma' \vdash e : \tau$.*

Proof. TODO ∎

0.g • Lemma. *If $\Xi \mid \Gamma \mid \Sigma \vdash e : \tau$ and $\Xi \mid \Gamma \mid \Sigma \vdash e' : \tau$ for the same type τ, then $\Xi \mid \Gamma \mid \Sigma \vdash K[e] : \rho$ and $\Xi \mid \Gamma \mid \Sigma \vdash K[e'] : \rho$ for the same type ρ.*

Proof. The proof is by induction on K. If $K = -$, then this is obvious.

$K = (K', e'')$: Then $K'[e]$ and $K'[e']$ have the same type, so by T-PAIR, so do $K[e]$ and $K[e']$.

$K = K'\,\_$: Then $K'[e]$ and $K'[e']$ have the same type by induction, and so do $K[e] = K'[e]\_$ and $K[e'] = K'[e']\_$ by T-TAPP.[7] [TODO need lemma saying that $\tau[\tau'/X]$ is a type! Need another production that says that X is a type?]

---

[7] TODO since we just apply it to underscore, it actually has a lot of different types. But we can find one type that works for both.

TODO rest ■

0.H • LEMMA. *If $\Xi \mid \Gamma \mid \Sigma \vdash e : \tau$ and $\Xi \mid \Gamma \mid \Sigma \vdash \sigma$ and $(\sigma, e) \rightarrow_h (\sigma', e')$, then there exists a store typing Σ' such that $\Sigma \subseteq \Sigma'$, $\Xi \mid \Gamma \mid \Sigma' \vdash e' : \tau$, and $\Xi \mid \Gamma \mid \Sigma' \vdash \sigma'$.*

Proof. We simply check all cases. [TODO mention pure cases]

E-FST: In this case $e = \mathsf{fst}(v_1, v_2)$ and $e' = v_1$ for values v_1, v_2. Then e is a value, so [TODO canonical forms] implies first that $\Xi \mid \Gamma \mid \Sigma \vdash (v_1, v_2) : \tau \times \tau'$ for some type τ', and then that $\Xi \mid \Gamma \mid \Sigma \vdash v_1 : \tau$.

E-TAPP-TLAM: Write the type of $\Lambda e\_$ as $\tau[\tau'/X]$. By inversion we have $\Xi \mid \Gamma \mid \Sigma \vdash \Lambda e : \forall X.\tau$, and again by inversion this implies that $\Xi, X \mid \Gamma \mid \Sigma \vdash e : \tau$. But then it follows from [TODO lemma 0] that $\Xi \mid \Gamma[\tau'/X] \mid \Sigma \vdash e : \tau[\tau'/X]$, and since Γ does not contain X (since Ξ does not) we have $\Gamma[\tau'/X] = \Gamma$, so the claim follows.

E-ALLOC: In this case $e = \mathsf{ref}\,v$, $e' = l$, $\sigma' = \sigma[l \mapsto v]$, and $l \notin \mathrm{dom}\,\sigma$. By inversion we have $\Xi \mid \Gamma \mid \Sigma \vdash \mathsf{ref}\,v : \mathsf{ref}(\tau')$ for some τ', and we further have $\Xi \mid \Gamma \mid \Sigma \vdash v : \tau'$. Now letting $\Sigma' = \Sigma[l \mapsto \tau']$, it follows from T-LOC that $\Xi \mid \Gamma \mid \Sigma' \vdash l : \mathsf{ref}(\Sigma'(l))$, so we have both $\Xi \mid \Gamma \mid \Sigma' \vdash \sigma'$ and $\Xi \mid \Gamma \mid \Sigma' \vdash l : \mathsf{ref}(\tau')$.

E-STORE: Here $e = l := v$, $e' = ()$ and $\sigma' = \sigma[l \mapsto v]$ with $l \in \mathrm{dom}\,\sigma$. Notice first that $l := v$ and $()$ both have type Unit by inversion, so that $\Xi \mid \Gamma \mid \Sigma \vdash () : \mathsf{Unit}$ as required.

By inversion we also have $\Xi \mid \Gamma \mid \Sigma \vdash l : \mathsf{ref}(\tau')$ and $\Xi \mid \Gamma \mid \Sigma \vdash v : \tau'$ for some type τ', and another application of inversion (TODO via T-LOC, or canonical forms?) implies that $\tau' = \Sigma(l)$. Furthermore, since $\Xi \mid \Gamma \mid \Sigma \vdash \sigma$, we have $\Xi \mid \Gamma \mid \Sigma \vdash \sigma(l) : \Sigma(l)$. Since $v = \sigma'(l)$ it thus follows that $\Xi \mid \Gamma \mid \Sigma \vdash \sigma'(l) : \Sigma(l)$, so $\Xi \mid \Gamma \mid \Sigma \vdash \sigma'$.

E-LOAD: Here $e = !l$, $e' = v$ and $\sigma = \sigma'$ with $\sigma(l) = v$. By inversion we have $\Xi \mid \Gamma \mid \Sigma \vdash l : \mathsf{ref}(\tau)$, so $\tau = \Sigma(l)$ [TODO again]. But since $\Xi \mid \Gamma \mid \Sigma \vdash \sigma$ we have $\Xi \mid \Gamma \mid \Sigma \vdash \sigma(l) : \Sigma(l)$, or in other words, $\Xi \mid \Gamma \mid \Sigma \vdash v : \tau$.

TODO rest ■

— TODO substitution lemma

0.I • LEMMA. *If $\Xi, X \mid \Gamma \mid \Sigma \vdash e : \tau$, then $\Xi \mid \Gamma[\tau'/X] \mid \Sigma \vdash e : \tau[\tau'/X]$.*

Proof. T-VAR: Assume that $\Xi, X \mid \Gamma \mid \Sigma \vdash x : \tau$. By inversion [TODO ish] we thus have $(x : \tau) \in \Gamma$, so $(x : \tau[\tau'/X]) \in \Gamma[\tau'/X]$ by definition of substitution. But then T-VAR implies that $\Xi \mid \Gamma[\tau'/X] \mid \Sigma \vdash x : \tau[\tau'/X]$ (where we use that X does not occur in the context or type, so it doesn't need to appear in Ξ).

T-REC: Assume that $\Xi, X \mid \Gamma \mid \Sigma \vdash \text{rec } f(x) := e : \tau_1 \to \tau_2$. By the inversion lemma [TODO ref – also can't we just do induction??] we have $\Xi, X \mid \Gamma, f : \tau_1 \to \tau_2, x : \tau_1 \mid \Sigma \vdash e : \tau_2$, so by induction it follows that $\Xi \mid \Gamma[\tau'/X], f : \tau_1[\tau'/X] \to \tau_2[\tau'/X], x : \tau_1[\tau'/X] \mid \Sigma \vdash e : \tau_2[\tau'/X]$ [TODO by substitution on envs, function types etc.]. Applying T-REC we obtain the desired claim. ∎

TODO rest

———————-

1 Statics in general

1.1 Syntax

We describe the general framework in which we may describe various programming languages, introducing the concepts that will later be defined precisely in the concrete setting of System F.

We first fix countable sets *Var* of variables and *Loc* of locations. The ***expressions*** of the language will be a set *Exp* containing both *Var* and *Loc*, and we designate some of these expressions to be ***values***, collected in a set *Val*. We think of an expression as specifying (part of) the state of the program, and values are states in which the computation of the program has finished. All locations will also be values, and these are supposed to model memory addresses. The memory state of the program (i.e. the part of the memory that the program has access to) is modelled by a ***store***[8], which is an element of $Loc \rightharpoonup_\omega Exp$. We simply write *Sto* for this set of maps.

For $e \in Exp$ we define the set $FV(e) \subseteq Var$ of ***free variables***. There are various ways of binding variables in expressions, and the notion of free variables is supposed to capture the idea that variables can be bound, e.g. by lambda abstraction, and hence also *not* bound. If $FV(e) = \emptyset$, then we say that e is ***closed***.

1.2 Type system

We next define a set *Type* of types. This includes a countable set *TVar* of type variables, any base types (e.g. unit, integer, and boolean types) as well as more complex types that can be constructed recursively from the base types. It also includes reference types, which are the types of locations. For $\tau \in Type$ we define the set of ***free type variables*** $FTV(\tau) \subseteq TVar$. As with free variables, these can be bound in various ways. If $FTV(\tau) = \emptyset$, then we also say that τ is

---

[8]Sometimes called a ***heap***, but this has nothing to do with the heap *data structure*.

closed. A pair (e, τ) of an expression and a type is usually written $e : \tau$. An expression may also have types as subexpressions, for instance if a lambda abstraction has a type annotation on its parameter.

If an expression e has free variables, then to specify the type of e it is necessary to first specify the types of the free variables in e. This is done using a ***type context***, which is an element of $Var \rightharpoonup_\omega Type$. If e is supposed to be well-typed in a type context Γ, then we of course require that $FV(e) \subseteq \operatorname{dom}\Gamma$. That is, Γ must in fact specify the types of the variables that occur free in e.

Furthermore, if e has a location as a subexpression, then we need to be able to look up the type of the expression stored at this location in order to specify the type of e. Hence the type of e will also depend on the store in question. However, it not in general possible to deduce the type of e in this case: If σ is a store, $l_1, l_2 \in \operatorname{dom}\sigma$, and if $\sigma(l_1)$ references l_2 and $\sigma(l_2)$ references l_1, then it is impossible to deduce the type of $\sigma(l_1)$. Hence the type of e will instead depend on a given ***store typing***, which is an element of $Loc \rightharpoonup_\omega Type$. We of course require that the store typing in question actually contains in its domain all locations referenced in e, and we furthermore require that all free variables of e lie in the domain of the current type context.

It may be that the type of e or of the types of the variables in the type context Γ or of the expressions in the store typing Σ contain type variables. In order to keep track of these we collect these in a (finite) set Ξ and require that the free type variables in Γ and Σ, defined by

$$FTV(\Gamma) = \bigcup_{\tau \in \operatorname{ran}\Gamma} FTV(\tau) \quad \text{and} \quad FTV(\Sigma) = \bigcup_{\tau \in \operatorname{ran}\Sigma} FTV(\tau),$$

are contained in Ξ. A variable x is called ***fresh*** for Γ if $x \notin \operatorname{dom}\Gamma$. The finitude of $\operatorname{dom}\Gamma$ ensures that there always exist fresh variables (recall that there are countably infinitely many variables). If Δ is another type context such that $\operatorname{dom}\Gamma \cap \operatorname{dom}\Delta = \emptyset$, then instead of $\Gamma \cup \Delta$ we simply write Γ, Δ. Furthermore, if $\Delta = \{x_1 : \tau_1, \ldots, x_n : \tau_n\}$ for distinct x_i, then we omit the braces and write $\Gamma, x_1 : \tau_1, \ldots, x_n : \tau_n$. If Ξ and Φ are finite disjoint subsets of $TVar$, then we similarly write Ξ, Φ for $\Xi \cup \Phi$, and if $\Phi = \{X_1, \ldots, X_n\}$ for distinct X_i, then we also write $\Xi, X_1, \ldots, X_n$.

The semantics of the type system is captured by a five-place relation on the set

$$\mathcal{P}_\omega(TVar) \times (Var \rightharpoonup_\omega Type) \times (Loc \rightharpoonup_\omega Type) \times Exp \times Type,$$

where an element $(\Xi, \Gamma, \Sigma, e, \tau)$ of this relation is written $\Xi \mid \Gamma \mid \Sigma \vdash e : \tau$ and is called a ***type derivation***. This relation is usually defined recursively, by specifying a series of axioms and inference rules[9]. If $\Xi = \emptyset$, then we write

---

[9]Or more usually axiom and rule *schemas*. TODO Pierce

$\Gamma \mid \Sigma \vdash e : \tau$ [TODO do we need this?], and we furthermore write $\Sigma \vdash e : \tau$ if also $\Gamma = \emptyset$.

2 Dynamics in general

The operational semantics of the language is specified in a small-step style. We describe this semantics in stages, beginning with the ***pure head reductions***. These are evaluations that can be performed (1) on expressions that have no subexpressions that can be evaluated, (2) without reading or modifying the store. More precisely, we specify a binary relation $\rightarrow_p$ on *Exp*, such that $e \rightarrow_p e'$ is supposed to mean that e evaluates to or reduces to e'.

Going one level up we define the relation $\rightarrow_h$ on *Sto*×*Exp* of (not necessarily pure) ***head reductions***. These are reductions that may affect and be affected by the contents of the store. Of course, if σ is a store and $e \rightarrow_p e'$, then we have $(\sigma, e) \rightarrow_h (\sigma, e')$, but we augment the pure head reductions with reductions that e.g. read from or write to the store.

Finally we need a way to evaluate complex expressions. One way of doing this is to specify the evaluation rules for all expressions immediately instead of going through head reductions (this is the approach taken by Pierce TODO). Another is to introduce ***evaluation contexts***, which are (essentially) maps *Exp* $\rightarrow$ *Exp*. If K is an evaluation context and e is an expression, then we write $K[e]$ for the value of K at e. We then define the final reduction relation $\rightarrow$ on *Sto* $\times$ *Exp* by letting $(\sigma, K[e]) \rightarrow (\sigma', K[e'])$ if $(\sigma, e) \rightarrow_h (\sigma', e')$.

One role of evaluation contexts is to specify the evaluation order of complex expressions, e.g. if the evaluation of function applications is call-by-value or call-by-name, or if we evaluate the arguments to functions left-to-right or right-to-left. The possibilities thus depend on the available evaluation contexts.

TODO multiple threads

3   System F

The following grammar defines the syntax, the sets of values and types, and the evaluation contexts of System F:

$$x \in Var$$
$$l \in Loc$$
$$X \in TVar$$

$$Exp \quad e ::= () \mid x \mid l \mid (e,e) \mid \cdots$$
$$Val \quad v ::= () \mid l \mid (v,v) \mid \mathsf{inj}_1 v \mid \cdots$$
$$Type \quad \tau ::= \mathsf{Unit} \mid X \mid \mathsf{ref}(\tau) \mid \tau \times \tau \mid \cdots$$
$$ECtx \quad K ::= - \mid (K,e) \mid (v,K) \mid \cdots$$

Notice that $Loc \subseteq Val \subseteq Exp$ as we required in TODO ref. If K is an evaluation context and e is an expression, then we define $K[e]$ recursively by

$$-[e] = e$$
$$(K,e')[e] = (K[e],e')$$
$$(v,K)[e] = (v,K[e])$$
$$etc.$$

The symbol '$-$' is called the **hole**, and we think of the hole as the place into which we substitute the expression e. It is easy to prove (by induction in K) that $K[e]$ is an expression, so every evaluation context K can indeed be thought of as a map $Exp \to Exp$ given by $e \mapsto K[e]$.

If e is an expression, then the set $FV(e)$ of free variables in e is defined recursively as follows:

$$FV(()) = \emptyset$$
$$FV(x) = \{x\}$$
$$FV((e_1,e_2)) = FV(e_1) \cup FV(e_2)$$
$$etc.$$

Similarly, if τ is a type, then we define the set $FTV(\tau)$ of free type variables in τ as follows:

$$FTV(\mathsf{Unit}) = \emptyset$$
$$FTV(X) = \{X\}$$
$$FTV(\tau_1 \times \tau_2) = FTV(\tau_1) \cup FTV(\tau_2)$$
$$etc.$$

We are now in a position to define the typing relation. This is the smallest relation on the set

$$\mathcal{P}_\omega(\mathit{TVar}) \times (\mathit{Var} \rightharpoonup_\omega \mathit{Type}) \times (\mathit{Loc} \rightharpoonup_\omega \mathit{Type}) \times \mathit{Exp} \times \mathit{Type},$$

satisfying the following inference rules:

$$\frac{\mathit{FTV}(\Gamma) \subseteq \Xi \qquad \mathit{FTV}(\Sigma) \subseteq \Xi \qquad (x : \tau) \in \Gamma}{\Xi \mid \Gamma \mid \Sigma \vdash x : \tau} \;\; \text{T-VAR}$$

Lemma: If $\Xi \mid \Gamma \mid \Sigma \vdash e : \tau$, then $\mathit{FTV}(\Gamma) \subseteq \Xi$ and $\mathit{FV}(e) \subseteq \mathrm{dom}\,\Gamma$. In particular, if $\Xi = \emptyset$ then τ is closed, and if $\Gamma = \emptyset$ then e is closed. TODO

TODO all type variables in tau also in Xi? All locations in e also in Sigma?

TODO lemma weakening?