EcmaScript 6 Module Semantics, Take 2

Andreas Rossberg [May 2, 2012]

1 Source Language

Syntax

$$
\begin{array}{rcl}
e & ::= & x \mid e.x \mid n \mid e{+}e \mid \ldots \\
m & ::= & x \mid m.x \mid \{d\} \\
d & ::= & \mathsf{let}\ x{=}e \mid \mathsf{module}\ x{=}m \mid \\
 & & \mathsf{export}\ x \mid \mathsf{export}\ \star \mid \mathsf{import}\ m.x \mid \mathsf{import}\ m.\star \mid \epsilon \mid d;d
\end{array}
$$

Notes:

- A program is treated as a module $\{d\}$.
- The syntax for projection is slightly generalized by allowing arbitrary m.

2 Algorithmic Type-Checking

Types and Environments

$$
\begin{array}{lrcl}
\text{Signatures} & \sigma & ::= & \alpha \mid \mathsf{V} \mid \{\rho\} \\
\text{Rows} & \gamma & ::= & \rho \mid x{:}\sigma \mid \cdot \mid \gamma, \gamma \mid \gamma^\star \\
\text{Environments} & \Gamma & ::= & \cdot \mid \Gamma; \gamma \\
\text{Constraints} & C & ::= & \bot \mid C, C \mid \alpha{:=}\hat{\sigma} \mid \rho{:=}\hat{\gamma} \\
\text{Constraint Signatures} & \hat{\sigma} & ::= & \cdots \mid \bot \mid \hat{\Gamma}(x) \mid \sigma.x \\
\text{Constraint Rows} & \hat{\gamma} & ::= & \cdots \mid \bot \mid \hat{\gamma}|_X \mid \sigma.\star \\
\text{Constraint Environments} & \hat{\Gamma} & ::= & \cdot \mid \hat{\Gamma}; \hat{\gamma}
\end{array}
$$

Implicit Equivalences

$$
\begin{array}{rclrcl}
\gamma, \cdot & = & \gamma & C, \bot & = & \bot \\
\gamma_1, \gamma_2 & = & \gamma_2, \gamma_1 & C_1, C_2 & = & C_2, C_1 \\
(\gamma_1, \gamma_2), \gamma_3 & = & \gamma_1, (\gamma_2, \gamma_3) & (C_1, C_2), C_3 & = & C_1, (C_2, C_3) \\
(\gamma_1, \gamma_2)^\star & = & \gamma_1^\star, \gamma_2^\star & & & \\
(\gamma^\star)^\star & = & \gamma^\star & & &
\end{array}
$$

Notes:

- We generalize Rémy's style rows by allowing arbitrary concatenation γ_1, γ_2.
- $\gamma^\star$ marks bindings that are defined via $\mathsf{import}\ m.\star$, for which overlaps are allowed and only raise an error if used. The rules give unstarred bindings precedence over starred ones.
- Environments separate scopes by ";", which is relevant for distinguishing between shadowing and overlapping.
- Instead of using unification, we build a set of more special-cased constraints that have to be solved in a separate phase.
- We omit the hat from $\hat{\sigma}$, $\hat{\gamma}$, $\hat{\Gamma}$ in the rules.

Typing Rules

$\boxed{\Gamma \vdash e : \sigma \mid C}$

$$\frac{\alpha \text{ fresh}}{\Gamma \vdash x : \alpha \mid \alpha{:=}\Gamma(x)} \qquad \frac{\Gamma \vdash e : \sigma \mid C \qquad \alpha \text{ fresh}}{\Gamma \vdash e.x : \alpha \mid C, \alpha{:=}\sigma.x}$$

$$\frac{}{\Gamma \vdash n : \mathsf{V} \mid \top} \qquad \frac{\Gamma \vdash e_1 : \sigma_1 \mid C_1 \qquad \Gamma \vdash e_2 : \sigma_2 \mid C_2}{\Gamma \vdash e_1{+}e_2 : \mathsf{V} \mid C_1, C_2}$$

- The argument types of addition are not constrained, errors are dynamic.

$\boxed{\Gamma \vdash m : \sigma \mid C}$

$$\frac{\alpha, \rho \text{ fresh}}{\Gamma \vdash x : \alpha \mid \alpha{:=}\Gamma(x), \rho{:=}\alpha.\star} \qquad \frac{\Gamma \vdash m : \sigma \mid C \qquad \alpha, \rho \text{ fresh}}{\Gamma \vdash m.x : \alpha \mid C, \alpha{:=}\sigma.x, \rho{:=}\sigma.\star}$$

$$\frac{\Gamma; \rho \vdash d : \gamma \mid C \qquad \rho, \rho' \text{ fresh}}{\Gamma \vdash \{d\} : \{\rho'\} \mid C, \rho{:=}\gamma, \rho'{:=}\gamma|_{\text{exports}(d)}}$$

- Modules are recursive, ρ captures the local definitions.
- The constraints $\rho{:=}\sigma.\star$ in the first two rules ensure that σ is a module type (unlike $\alpha{:=}\sigma.x$, which allows $\sigma = \mathsf{V}$, for expression-level projection).
- The meta-function exports(d) gives the set of **export**-ed identifiers in d, and restricts the domain of γ to that set. We assume that **export** $\star$ yields the infinite set of all identifiers (written $\star$).

$\boxed{\Gamma \vdash d : \gamma \mid C}$

$$\frac{\Gamma \vdash e : \sigma \mid C}{\Gamma \vdash \mathsf{let}\ x{=}e : x{:}\mathsf{V} \mid C} \qquad \frac{\Gamma \vdash m : \sigma \mid C \qquad \alpha \text{ fresh}}{\Gamma \vdash \mathsf{module}\ x{=}m : x{:}\alpha \mid C, \alpha{:=}\sigma}$$

$$\frac{}{\Gamma \vdash \mathsf{export}\ x : \cdot \mid \top} \qquad \frac{}{\Gamma \vdash \mathsf{export}\ \star : \cdot \mid \top}$$

$$\frac{\Gamma \vdash m : \sigma \mid C \qquad \alpha, \rho \text{ fresh}}{\Gamma \vdash \mathsf{import}\ m.x : x{:}\alpha \mid C, \alpha{:=}\sigma.x, \rho{:=}\sigma.\star} \qquad \frac{\Gamma \vdash m : \sigma \mid C \qquad \rho \text{ fresh}}{\Gamma \vdash \mathsf{import}\ m.\star : \rho^\star \mid C, \rho{:=}\sigma.\star}$$

$$\frac{}{\Gamma \vdash \epsilon : \cdot \mid \top} \qquad \frac{\Gamma \vdash d_1 : \gamma_1 \mid C_1 \qquad \Gamma \vdash d_2 : \gamma_2 \mid C_2 \qquad \rho \text{ fresh}}{\Gamma \vdash d_1{;}d_2 : \rho \mid C_1, C_2, \rho{:=}\gamma_1, \gamma_2}$$

- All module identifiers are given a fresh type variable as their immediate type. This allows detecting cycles when solving the constraints.
- Star annotations are sticky across export boundaries. Consider:

  $A_1 = \{\mathsf{export}\ x\}; A_2 = \{\mathsf{export}\ x\}; AA = \{\mathsf{import}\ A_1.\star; \mathsf{import}\ A_2.\star; \mathsf{export}\ x\};$
  $\mathsf{let}\ y = AA.x$ // error!
  $B = \{\mathsf{import}\ AA.x; \mathsf{let}\ y = x\}$ // error!
  $C = \{\mathsf{import}\ AA.x; \mathsf{import}\ A_1.x; \mathsf{let}\ y = x\}$ // ok

Constraint Resolution

$$
\begin{array}{lll l}
(\Gamma;\gamma)(x) & \Rightarrow & (\Gamma;\gamma')(x) & (\gamma \Rightarrow \gamma') \\
(\Gamma;\gamma,x{:}\sigma)(x) & \Rightarrow & \sigma & \\
(\Gamma;x{:}\sigma^\star)(x) & \Rightarrow & \sigma & \\
(\Gamma;\gamma,x{:}\sigma_1^\star,x{:}\sigma_2^\star)(x) & \Rightarrow & \bot & \\
(\Gamma;\gamma,y{:}\sigma)(x) & \Rightarrow & (\Gamma;\gamma)(x) & \\
(\Gamma;\gamma,y{:}\sigma^\star)(x) & \Rightarrow & (\Gamma;\gamma)(x) & \\
(\Gamma;\cdot)(x) & \Rightarrow & \Gamma(x) & \\
(\cdot)(x) & \Rightarrow & \bot & \\
\{\rho\}.x & \Rightarrow & (\cdot;\rho)(x) & \\
\mathsf{V}.x & \Rightarrow & \mathsf{V} & \\
\{\rho\}.\star & \Rightarrow & \rho & \\
\mathsf{V}.\star & \Rightarrow & \bot & \\
\gamma|_\star & \Rightarrow & \gamma & \\
\gamma|_\emptyset & \Rightarrow & \cdot & \\
(x{:}\sigma,\gamma)|_X & \Rightarrow & x{:}\sigma,(\gamma|_{X-x}) & (x \in X) \\
(x{:}\sigma^\star,\gamma^\star)|_X & \Rightarrow & x{:}\sigma^\star,(\gamma^\star|_X) & (x \in X) \\
(x{:}\sigma,\gamma)|_X & \Rightarrow & \gamma|_X & (x \notin X) \\
(x{:}\sigma^\star,\gamma)|_X & \Rightarrow & \gamma|_X & (x \notin X) \\
x{:}\sigma_1,x{:}\sigma_2 & \Rightarrow & \bot & \\
x{:}\sigma_1,x{:}\sigma_2^\star & \Rightarrow & x{:}\sigma_1 & \\
\gamma_1,\gamma_2 & \Rightarrow & \gamma_1',\gamma_2 & (\gamma_1 \Rightarrow \gamma_1') \\
\alpha{:=}\sigma & \Rightarrow & \alpha{:=}\sigma' & (\sigma \Rightarrow \sigma') \\
\alpha{:=}\alpha & \Rightarrow & \bot & \\
\alpha{:=}\bot & \Rightarrow & \bot & \\
\rho{:=}\gamma & \Rightarrow & \rho{:=}\gamma' & (\gamma \Rightarrow \gamma') \\
\rho{:=}(\rho,\gamma) & \Rightarrow & \bot & \\
\rho{:=}\bot & \Rightarrow & \bot & \\
C,\alpha{:=}\sigma & \Rightarrow & C[\sigma/\alpha],\alpha{:=}\sigma & (\sigma \neq \alpha) \\
C,\rho{:=}\gamma & \Rightarrow & C[\gamma/\rho],\rho{:=}\gamma & (\gamma \neq \rho)
\end{array}
$$

- It is an error to use an ambiguous starred identifier (4th rule for $\Gamma(x)$).
- Ambiguous unstarred identifiers are always an error (1st rule for γ_1, γ_2).
- Also, unstarred (i.e. explicitly bound) identifiers hide starred ones (2nd rule for γ_1, γ_2).
- We allow $\sigma.x$ with $\sigma = \mathsf{V}$, in order to handle the $e.x$ case.
- Resolution terminates when $\bot$ arises, or when all constraints are in basic form, i.e. the r.h.s. is a proper σ/γ that is well-formed and not a variable.
- No step introduces new constraints. It might be possible to define a suitable weight function on types and constraints for proving non-divergence.
- It is non-obvious that we cannot get stuck...
- All this looks somewhat too ad-hoc and operational for my taste. How can we give a declarative semantics to this mess?