EcmaScript 6 Module Semantics

Andreas Rossberg [January 18, 2012]

1 Source Language

Syntax

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

Notes:

- A program is treated as a module $\{d\}$.
- The syntax for projection is slightly generalized in allowing arbitrary m.
- Other module syntax can be seen as sugar for the above.

2 Types and Environments

Syntax

$$\begin{array}{lrcl} \text{Signatures} & \Sigma & ::= & \mathsf{V} \mid \{\Gamma\} \mid \alpha \\ \text{Term Environments} & \Gamma & ::= & \cdot \mid \Gamma, x{:}\Sigma \\ \text{Type Environments} & \Delta & ::= & \cdot \mid \Delta, \alpha{=}\Sigma^{\pi} \\ \text{Positions} & \pi & ::= & \mathsf{L} \mid \mathsf{R} \mid \mathsf{U} \mid \mathsf{D} \end{array}$$

Notes:

- V describes let-bound values, $\{\Gamma\}$ modules.
- Type variables and type environments are utilised to represent recursive types (μ-types would be much more difficult to handle, I tried...).
- Positions describe the relative order of respective module bindings and their scopes, relative to the one 'currently' looked at by a rule:
  - L are types of modules *left* (earlier) in the same scope.
  - R are types of modules *right* (later) in the same or an outer scope.
  - U are types of modules *upward* (enclosing) or their earlier children.
  - D are types of modules *downward* (enclosed).

  We use these to abstractly simulate the algorithmic traversal order in the declarative rules.

Auxiliary Definitions

$$\begin{array}{rcll}
(\Delta, \alpha{=}\Sigma^{\pi})^{\pi'/\pi} & = & \Delta^{\pi'/\pi}, \alpha{=}\Sigma^{\pi'} & \\
(\Delta, \alpha{=}\Sigma^{\pi''})^{\pi'/\pi} & = & \Delta^{\pi'/\pi}, \alpha{=}\Sigma^{\pi''} & \text{if } \pi'' \neq \pi \\
\cdot^{\pi'/\pi} & = & \cdot & \\
\Delta^{\pi} & = & (((\Delta^{\pi/\mathsf{U}})^{\pi/\mathsf{D}})^{\pi/\mathsf{L}})^{\pi/\mathsf{R}} &
\end{array}$$

Δ^+ is defined as smallest fixpoint of the following equations:

$$\begin{array}{rcll}
\Delta^+(\alpha) & = & \{\alpha'\} \cup \Delta^+(\alpha') & \text{if } \Delta(\alpha) = \alpha' \\
\Delta^+(\alpha) & = & \{\} & \text{otherwise}
\end{array}$$

Note: The condition $\alpha \in \Delta^+(\alpha)$ thereby describes an illegal type cycle.

Well-formedness

$\boxed{\vdash \Delta}$ $\boxed{\Delta \vdash \Delta}$

$$\frac{\Delta \vdash \Delta}{\vdash \Delta} \qquad \frac{}{\Delta \vdash \cdot} \qquad \frac{\Delta \vdash \Delta' \qquad \Delta \vdash \Sigma \qquad \alpha \notin \Delta^+(\alpha)}{\Delta \vdash \Delta', \alpha{=}\Sigma^{\pi}}$$

Note: Type environments may not contain any direct cycles.

$\boxed{\Delta \vdash \Gamma}$

$$\frac{}{\Delta \vdash \cdot} \qquad \frac{\Delta \vdash \Gamma \qquad \Delta \vdash \Sigma}{\Delta \vdash \Gamma, x{:}\Sigma}$$

$\boxed{\Delta \vdash \Sigma}$

$$\frac{}{\Delta \vdash \mathsf{V}} \qquad \frac{\Delta \vdash \Gamma}{\Delta \vdash \{\Gamma\}} \qquad \frac{\alpha \in \mathrm{dom}(\Delta)}{\Delta \vdash \alpha}$$

Type Normalisation

$\boxed{\Delta \vdash \Sigma \downarrow \Sigma'}$

$$\frac{}{\Delta \vdash \mathsf{V} \downarrow \mathsf{V}} \qquad \frac{}{\Delta \vdash \{\Gamma\} \downarrow \{\Gamma\}} \qquad \frac{\Delta \vdash \Delta(\alpha) \downarrow \Sigma}{\Delta \vdash \alpha \downarrow \Sigma}$$

Note: Gives the weak-head normal form of signatures wrt δ-reduction over Δ.

3 Typing Rules

$\boxed{\Delta; \Gamma \vdash m : \Sigma}$

$$\frac{\Delta \vdash \Gamma(x) \downarrow \{\Gamma'\}}{\Delta; \Gamma \vdash x : \Gamma(x)} \qquad \frac{\Delta; \Gamma \vdash m : \Sigma \qquad \Delta \vdash \Sigma \downarrow \{\Gamma'\} \qquad \Delta \vdash \Gamma'(x) \downarrow \{\Gamma''\}}{\Delta; \Gamma \vdash m.x : \Gamma'(x)}$$

$$\frac{\Delta^{\mathsf{U/L}}, \Delta_1^{\mathsf{D}}; \Gamma, \Gamma_1 \vdash d : \Gamma_1 \qquad \vdash \Delta, \Delta_1 \qquad \Delta, \Delta_1 \vdash \Gamma, \Gamma_1}{\Delta; \Gamma \vdash \{d\} : \{\Gamma_1|_{\text{exports}(d)}\}} \qquad \frac{\Delta \vdash \Gamma'}{\Delta; \Gamma \vdash \mathsf{from}\ u : \{\Gamma'\}}$$

Notes (on the rule for module literals):

- Modules are recursive, Γ_1 captures the local definitions.
- The meta-function exports(d) gives the set of **export**-ed identifiers in d, and restricts the domain of Γ_1 to that set. (For simplicity, omitted a check for exports(d) $\subseteq$ dom(Γ_1).) The rule also allows to re-export imports.
- Δ_1 captures local types from Γ_1 that do not escape into $\Gamma_1|_{\text{exports}(d)}$.
- Because we descent into a nested module, all L-types in Δ turn to U. The types in Δ_1 otoh. are all local, and thus D.

$\boxed{\Delta; \Gamma \vdash d : \Gamma'}$

$$\frac{\Delta; \Gamma \vdash e}{\Delta; \Gamma \vdash \mathsf{let}\ x{=}e : x{:}\mathsf{V}} \qquad \frac{\Delta, \alpha{=}\Sigma^{\mathsf{L}}; \Gamma \vdash m : \Sigma}{\Delta, \alpha{=}\Sigma^{\mathsf{D}}; \Gamma \vdash \mathsf{module}\ x{=}m : x{:}\alpha}$$

$$\frac{}{\Delta; \Gamma \vdash \mathsf{export}\ x : \cdot}$$

$$\frac{\Delta, \alpha{=}\Sigma^{\mathsf{L}}; \Gamma \vdash m : \Sigma' \qquad \Delta, \alpha{=}\Sigma^{\mathsf{L}} \vdash \Sigma' \downarrow \{\Gamma'\} \qquad \Gamma'(x) = \Sigma}{\Delta, \alpha{=}\Sigma^{\mathsf{D}}; \Gamma \vdash \mathsf{import}\ m.x : x{:}\alpha}$$

$$\frac{\Delta, \Delta_1^{\mathsf{L}}; \Gamma \vdash m : \Sigma \qquad \Delta|_{\mathsf{U}} \vdash \Sigma \downarrow \{\Delta_1 \circ \Gamma'\} \qquad \Gamma' \text{ bijection}}{\Delta, \Delta_1^{\mathsf{D}}; \Gamma \vdash \mathsf{import}\ m.\star : \Gamma'}$$

$$\frac{}{\Delta; \Gamma \vdash \epsilon : \cdot} \qquad \frac{\Delta, \Delta_1^{\mathsf{D}}, \Delta_2^{\mathsf{R}}; \Gamma \vdash d_1 : \Gamma_1 \qquad \Delta, \Delta_1^{\mathsf{L}}, \Delta_2^{\mathsf{D}}; \Gamma \vdash d_2 : \Gamma_2}{\Delta, \Delta_1^{\mathsf{D}}, \Delta_2^{\mathsf{D}}; \Gamma \vdash d_1;d_2 : \Gamma_1, \Gamma_1}$$

Notes:

- All module identifiers are given a fresh type variable as their immediate type. This way, (1) type variables uniquely identify module bindings, and (2) a module X aliases Y iff $\Gamma(Y) \in \Delta^+(\Gamma(X))$. The actual Σ is assumed to be (non-deterministically) bound to that variable in Δ already.
- For "import $m.\star$" we require that its signature can be normalized by referring only to "upper" types – i.e., modules previously defined. No other restrictions on either imports or exports exist.
- A module declaration is considered left of anything on its rhs. This allows $\star$-importing a parent in a nested module, e.g.:

  module A = { . . . ; module B = { import A.$\star$ }; . . . }

- Both forms of import can import both values and modules.

4 Algorithmic Type-Checking

Types and Environments

$$\begin{array}{llcl}
\text{Signatures} & \sigma & ::= & \mathsf{V} \mid \{\gamma\} \mid \alpha \\
\text{Rows} & \gamma & ::= & \cdot \mid \gamma, x{:}\sigma \mid \rho \\
\text{Type Environments} & \delta & ::= & \cdot \mid \delta, \alpha{=}\sigma^{?} \mid \delta, \rho{=}\gamma^{?}
\end{array}$$

Notes:

- ρ represents row variables à la Rémy's polymorphic record types. Definite rows are identified up to reordering.
- δ stores unification variables; $x^{?}$ is either x or $\_$ (unbound).

Auxiliary Definitions

$$\begin{array}{rcl}
\gamma, \cdot & = & \gamma \\
\gamma, (\gamma', \alpha{=}\sigma^{?}) & = & (\gamma, \gamma'), \alpha{=}\sigma^{?}
\end{array}$$

Note: the concatenation γ, ρ is not defined.

Unification

$\boxed{\delta \vdash \sigma_1 \overset{!}{=} \sigma_2 \dashv \delta'}$

$$\frac{}{\delta \vdash \sigma \overset{!}{=} \sigma \dashv \delta} \qquad \frac{\delta \vdash \sigma_2 \overset{!}{=} \sigma_1 \dashv \delta'}{\delta \vdash \sigma_1 \overset{!}{=} \sigma_2 \dashv \delta'}$$

$$\frac{\delta \vdash \delta(\alpha) \overset{!}{=} \sigma \dashv \delta'}{\delta \vdash \alpha \overset{!}{=} \sigma \dashv \delta'} \qquad \frac{\alpha \neq \alpha' \qquad \delta(\alpha') = \_}{\delta, \alpha{=}\_ \vdash \alpha \overset{!}{=} \alpha' \dashv \delta, \alpha{=}\alpha'} \qquad \frac{\sigma \text{ not a variable}}{\delta, \alpha{=}\_ \vdash \alpha \overset{!}{=} \sigma \dashv \delta, \alpha{=}\sigma}$$

$$\frac{\delta \vdash \gamma_1 \overset{!}{=} \gamma_2 \dashv \delta'}{\delta \vdash \{\gamma_1\} \overset{!}{=} \{\gamma_2\} \dashv \delta'}$$

$\boxed{\delta \vdash \gamma_1 \overset{!}{=} \gamma_2 \dashv \delta'}$

$$\frac{}{\delta \vdash \gamma \overset{!}{=} \gamma \dashv \delta} \qquad \frac{\delta \vdash \gamma_2 \overset{!}{=} \gamma_1 \dashv \delta'}{\delta \vdash \gamma_1 \overset{!}{=} \gamma_2 \dashv \delta'}$$

$$\frac{\delta \vdash \delta(\rho) \overset{!}{=} \gamma \dashv \delta'}{\delta \vdash \rho \overset{!}{=} \gamma \dashv \delta'} \qquad \frac{\rho \neq \rho' \qquad \delta(\rho') = \_}{\delta, \rho{=}\_ \vdash \rho \overset{!}{=} \rho' \dashv \delta, \rho{=}\rho'} \qquad \frac{\gamma \text{ not a variable}}{\delta, \rho{=}\_ \vdash \rho \overset{!}{=} \gamma \dashv \delta, \rho{=}\gamma}$$

$$\frac{\delta \vdash \sigma_1 \overset{!}{=} \sigma_2 \dashv \delta' \qquad \delta' \vdash \gamma_1 \overset{!}{=} \gamma_1 \dashv \delta''}{\delta \vdash \gamma_1, x{:}\sigma_1 \overset{!}{=} \gamma_2, x{:}\sigma_2 \dashv \delta''}$$

$$\frac{x_1 \neq x_2 \qquad \delta, \rho{=}\_ \vdash \gamma_1 \overset{!}{=} \rho, x_2{:}\sigma_2 \dashv \delta' \qquad \delta' \vdash \gamma_2 \overset{!}{=} \rho, x_1{:}\sigma_1 \dashv \delta''}{\delta \vdash \gamma_1, x_1{:}\sigma_1 \overset{!}{=} \gamma_2, x_2{:}\sigma_2 \dashv \delta''}$$

Row Normalisation

$\boxed{\delta \vdash \gamma \downarrow \gamma'}$

$$\frac{}{\delta \vdash \cdot \downarrow \cdot} \qquad \frac{\delta \vdash \gamma \downarrow \gamma'}{\delta \vdash \gamma, x{:}\sigma \downarrow \gamma', x{:}\sigma} \qquad \frac{\delta \vdash \delta(\rho) \downarrow \gamma}{\delta \vdash \rho \downarrow \gamma} \qquad \frac{\delta(\rho) = \_}{\delta \vdash \rho \downarrow \rho}$$

Algorithmic Rules

$\boxed{\delta; \gamma \vdash m}$

$$\frac{\delta, \rho{=}\_; \gamma \vdash m \Leftarrow \{\rho\} \dashv \delta' \qquad \_ \notin \mathrm{ran}(\delta')}{\delta; \gamma \vdash m}$$

Notes:

- A complete program may not produce any unbound unification variables. They would correspond to a cyclic definition, e.g. $\mathsf{module\ X = X}$.
- We assume the initial δ and γ are well-formed. In particular, δ contains no direct cycles, and γ contains no row variable.

$\boxed{\delta; \gamma \vdash m \Leftarrow \sigma \dashv \delta'}$

$$\frac{\delta \vdash \sigma \overset{!}{=} \gamma(x) \dashv \delta' \qquad \delta', \rho{=}\_ \vdash \gamma(x) \overset{!}{=} \{\rho\} \dashv \delta''}{\delta; \gamma \vdash x \Leftarrow \sigma \dashv \delta''}$$

$$\frac{\delta, \rho{=}\_; \gamma \vdash m \Leftarrow \{\rho, x{:}\sigma\} \dashv \delta'}{\delta; \gamma \vdash m.x \Leftarrow \sigma \dashv \delta'}$$

$$\frac{\delta; \gamma \vdash d \Rightarrow \gamma' \dashv \delta' \qquad \delta' \vdash \sigma \overset{!}{=} \{\gamma'|_{\mathrm{exports}(d)}\} \dashv \delta'' \qquad \delta''; \gamma, \gamma' \vdash d \dashv \delta'''}{\delta; \gamma \vdash \{d\} \Leftarrow \sigma \dashv \delta'''}$$

Notes:

- Typing modules happens in analysis mode: σ is an input to unify with.
- To type a module literal, we first compute a preliminary γ' (assigning fresh type variables, except with $\star$-imports, see below), and then descent into type-checking d properly. By unifying with σ before the recursion, we enable using it for $\star$-imports inside d (corresponds to switching D to L in the declarative rule).

$\boxed{\delta; \gamma \vdash d \Rightarrow \gamma \dashv \delta'}$

$$\frac{}{\delta; \gamma \vdash \mathsf{let}\ x{=}e \Rightarrow x{:}\mathsf{V} \dashv \delta} \qquad \frac{}{\delta; \gamma \vdash \mathsf{module}\ x{=}m \Rightarrow x{:}\alpha \dashv \delta, \alpha{=}\_}$$

$$\frac{}{\delta; \gamma \vdash \mathsf{export}\ x \Rightarrow \cdot \dashv \delta} \qquad \frac{}{\delta; \gamma \vdash \mathsf{import}\ m.x \Rightarrow x{:}\alpha \dashv \delta, \alpha{=}\_}$$

$$\frac{\delta, \rho{=}\_; \gamma \vdash m \Leftarrow \{\rho\} \dashv \delta' \qquad \delta' \vdash \rho \downarrow \cdot, \gamma'}{\delta; \gamma \vdash \mathsf{import}\ m.\star \Rightarrow \gamma' \dashv \delta'}$$

$$\frac{}{\delta; \gamma \vdash \epsilon \Rightarrow \cdot \dashv \delta} \qquad \frac{\delta; \gamma \vdash d_1 \Rightarrow \gamma_1 \dashv \delta' \qquad \delta'; \gamma \vdash d_2 \Rightarrow \gamma_2 \dashv \delta''}{\delta; \gamma \vdash d_1{;}d_2 \Rightarrow \gamma_1, \gamma_1 \dashv \delta''}$$

Notes:

- This judgement precomputes a fresh γ for a module body. It assigns fresh type variables to anything that could be a module.
- However, for $\star$-imports we require that the domain of the signature of the referenced module is already fully determined at this stage (its row normalizes to $\cdot, \rho'$, which implies that ρ' does not contain a row variable).

$\boxed{\delta; \gamma \vdash d \dashv \delta'}$

$$\frac{\delta; \gamma \vdash e \dashv \delta'}{\delta; \gamma \vdash \mathsf{let}\ x{=}e \dashv \delta'} \qquad \frac{\delta; \gamma \vdash m \Leftarrow \gamma(x) \dashv \delta'}{\delta; \gamma \vdash \mathsf{module}\ x{=}m \dashv \delta'}$$

$$\frac{}{\delta; \gamma \vdash \mathsf{export}\ x \dashv \delta}$$

$$\frac{\delta, \rho{=}\_; \gamma \vdash m \Leftarrow \{\rho, x{:}\gamma(x)\} \dashv \delta'}{\delta; \gamma \vdash \mathsf{import}\ m.x \dashv \delta'} \qquad \frac{}{\delta; \gamma \vdash \mathsf{import}\ m.\star \dashv \delta}$$

$$\frac{}{\delta; \gamma \vdash \epsilon \dashv \delta} \qquad \frac{\delta; \gamma \vdash d_1 \dashv \delta' \qquad \delta'; \gamma \vdash d_2 \dashv \delta''}{\delta; \gamma \vdash d_1{;}d_2 \dashv \delta''}$$

Notes:

- This judgement merely checks the right-hand sides and unifies the pre-assigned type variables accordingly.
- For $\star$-imports, there is nothing left to do at this stage.

Other Notes:

- The types inferred by the algorithm and the ones derived by the declarative rules are equivalent only up to normalization, because unification does not give enough control over the binding order of variables (and also performs path compression).
- Hence, the previously mentioned aliasing condition "X aliases Y iff $\Gamma(Y) \in \Delta^+(\Gamma(X))$" does not hold for the algorithmic types. Aliasing has to be tracked by additional means, e.g. by giving identity to module signatures. The prototype uses straightforward physical equality on signature values, along with their physical unification.