# Semantics for Modular Analysis

Joonhyup Lee

## 1 Abstract Syntax

In this section we define the abstract syntax for a simple language that captures the essence of modules and linking. The language is basically an extension of untyped lambda calculus with modules and the linking operator.

$$
\begin{array}{lcll}
x & \in & \mathit{ExprVar} & \\
M & \in & \mathit{ModVar} & \\
m & ::= & \varepsilon & \textit{empty module} \\
 & | & M & \textit{identifier, module} \\
 & | & \texttt{let}\ x\ e\ m & \textit{let-binding, expression} \\
 & | & \texttt{let}\ M\ m\ m & \textit{let-binding, module} \\
e & ::= & x & \textit{identifier, expression} \\
 & | & \lambda x.e & \textit{function} \\
 & | & e\ e & \textit{application} \\
 & | & m!e & \textit{linked expression}
\end{array}
$$

## 2 Big-Step Operational Semantics

In this section we give the big-step operational semantics for the dynamic execution of the language defined previously. The relation which gives the semantics relates the initial state(memory and time) and configuration(the subexpression being evaluated, the list of binding times, and the surrounding context) with the resulting state and configuration.

This relation is nonstandard in that the *environment* that is often used to define closures in the call-by-value dynamics is not a finite map from variables to values. Rather, the surrounding context and the list of binding times give when the variables were bound, which can then be looked up from the memory. Concretely, the $i$-th time of the $p$ component of the configuration gives when the $i$-th variable counted from the hole upwards in $C$ was bound. $i$ is called the “De Bruijn index” of the variable.

The reason the semantics is defined as such is for the convenience and precision of abstraction. One only has to finitize the time component to make the search space finite. Furthermore, by including the $C$ component as an essential part of the configuration, one can reason precisely about how the syntactic change on the surrounding context affects the evaluation of the subexpression.

$$
[\textsc{ExprVar}]\ \dfrac{i = \mathsf{index}(C, x) \qquad p_x = \mathsf{pop}^i(p)}{(x, p, C), (\sigma, t) \Downarrow \sigma(p_x), (\sigma, t)}
$$

$$
[\textsc{Fn}]\ \dfrac{}{(\lambda x.e, p, C), (\sigma, t) \Downarrow (\lambda x.e, p, C), (\sigma, t)}
$$

$$
[\textsc{App}]\ \dfrac{\begin{array}{c}(e_1, p, C[[]\ e_2]), (\sigma, t) \Downarrow (\lambda x.e_\lambda, p_\lambda, C_\lambda), (\sigma_\lambda, t_\lambda) \\ (e_2, p, C[e_1\ []]), (\sigma_\lambda, t_\lambda) \Downarrow (a, p_a, C_a), (\sigma_a, t_a) \\ (e_\lambda, t_a :: p_\lambda, C_\lambda[\lambda x.[]]), (\sigma_a[t_a :: p_\lambda \mapsto (a, p_a, C_a)], t_a + 1) \Downarrow (v, p_v, C_v), (\sigma_v, t_v)\end{array}}{(e_1\ e_2, p, C), (\sigma, t) \Downarrow (v, p_v, C_v), (\sigma_v, t_v)}
$$

$$
[\textsc{Linking}]\ \dfrac{\begin{array}{c}(m, p, C), (\sigma, t) \Downarrow (p_m, C_m), (\sigma_m, t_m) \\ (e, p_m, C_m), (\sigma_m, t_m) \Downarrow (v, p_v, C_v), (\sigma_v, t_v)\end{array}}{(m!e, p, C), (\sigma, t) \Downarrow (v, p_v, C_v), (\sigma_v, t_v)}
$$

$$
[\textsc{Empty}]\ \dfrac{}{(\varepsilon, p, C), (\sigma, t) \Downarrow (p, C), (\sigma, t)}
$$

$$
[\textsc{ModVar}]\ \dfrac{i = \mathsf{index}(C, M) \qquad p_M = \mathsf{pop}^i(p)}{(M, p, C), (\sigma, t) \Downarrow \sigma(p_M), (\sigma, t)}
$$

$$[\textsc{LetE}]\ \frac{\begin{array}{c}(e, p, C), (\sigma, t) \Downarrow (v, p_v, C_v), (\sigma_v, t_v) \\ (m, t_v :: p, C[\texttt{let}\ x\ e\ []]), (\sigma_v[t_v :: p \mapsto (v, p_v, C_v)], t_v + 1) \Downarrow (p_m, C_m), (\sigma_m, t_m)\end{array}}{(\texttt{let}\ x\ e\ m, p, C), (\sigma, t) \Downarrow (p_m, C_m), (\sigma_m, t_m)}$$

$$[\textsc{LetM}]\ \frac{\begin{array}{c}(m_1, p, C), (\sigma, t) \Downarrow (p', C'), (\sigma', t') \\ (m_2, t' :: p, C[\texttt{let}\ M\ m_1\ []]), (\sigma'[t' :: p \mapsto (p', C')], t' + 1) \Downarrow (p_m, C_m), (\sigma_m, t_m)\end{array}}{(\texttt{let}\ M\ m_1\ m_2, p, C), (\sigma, t) \Downarrow (p_m, C_m), (\sigma_m, t_m)}$$

Before there were modules, all valid configurations $(e, p, C)$ from the initial configuration $(e_0, \mathsf{nil}, [])$ satisfied $C[e] = e_0$. However, because of the linking rule, this is no longer true. It is still, true, however, that the $C$ component of a configuration is determined from the syntax of the initial expression, and is thus always bounded by the "depth" of $e_0$. How do we express this property now?

## 3 Collecting Semantics

Now we make the semantics of a module $m$ and the semantics of an expression $e$ explicit.