Big Step Semantics

\[ e \rightarrow e' \text{ "e steps to } e'" \text{ (one step)} \]
\[ e \rightarrow^* e' \text{ "e steps to } e'" \text{ (many steps)} \]
\[ e \rightarrow^* v \text{ and } v \text{ eval } "e evaluates to } v" \]
\[ e \Downarrow v \text{ "e evaluates to } v" \]

\[ e := x!() | \lambda x.e | e_1 e_2 | (e, e) | \text{fst } e | \text{snd } e | \text{linl } e \]
\[ \text{inr } e | \text{case } e \text{ of } \lambda x.e_1 | x.e_2 | \text{fix } x=e \]

\[ t := \text{unit } | t \rightarrow t | t \times c | t + t \]

\[ v := () | \lambda x.e | (v, v) | \text{linl } v | \text{linr } v \]

Combines 3 step rules for application

\[ \frac{\nu \nu}{\nu \nu} \quad \frac{\nu \nu}{\nu \nu} \quad \frac{\nu \nu}{\nu \nu} \quad \frac{\nu \nu}{\nu \nu} \quad \frac{\nu \nu}{\nu \nu} \quad \frac{\nu \nu}{\nu \nu} \]

\[ \frac{\nu \nu}{\nu \nu} \quad \frac{\nu \nu}{\nu \nu} \quad \frac{\nu \nu}{\nu \nu} \quad \frac{\nu \nu}{\nu \nu} \quad \frac{\nu \nu}{\nu \nu} \quad \frac{\nu \nu}{\nu \nu} \]

Way less rules!

\[ \frac{\text{case } e \text{ of } \lambda x.e_1 | x.e_2 | v. v'}{\nu \nu} \]
\[ \frac{\text{case } e_1 \text{ of } \lambda x.e_2 | v. v'}{\nu \nu} \]

\[ \frac{\text{fix } x=e/x \nu \nu}{\nu \nu} \]

\[ \frac{\text{fix } x=e \nu \nu}{\nu \nu} \]
Preservation: If \( \Gamma \vdash e : \tau \) and \( \epsilon \vdash v, \Gamma_{\epsilon} \vdash v : \tau \). 

\( \vdash v \) 

\( (E-1) \) \( \checkmark \)

\( (E-2) \) Then \( e = e_1, e_2 \) and \( e_1 \downarrow \forall \Delta e \) and \( e_2 \downarrow \forall \Delta e \).

By inversion, \( \tau = \tau \) and \( \Gamma, \epsilon \vdash e_1 : \tau \) and \( \Gamma, \epsilon \vdash e_2 : \tau \).

By induction, \( \Gamma, \epsilon \vdash e_1 : \tau \) and \( \Gamma, \epsilon \vdash e_2 : \tau \).

By inversion, \( \Gamma, \epsilon \vdash e_1 : \tau \).

By substitution, \( \Gamma, \epsilon \vdash [v/x] e : \tau \). By induction, \( \Gamma, \epsilon \vdash v : \tau \).

\( (E-3) \) Then \( e = (e_1, e_2) \) and \( e_1 \downarrow \forall \Delta e \), and \( e_2 \downarrow \forall \Delta e \) and \( v = (v_1, v_2) \).

By inversion, \( \tau = \tau \times \tau \) and \( \Gamma, \epsilon \vdash e_1 : \tau \) and \( \Gamma, \epsilon \vdash e_2 : \tau \).

By induction, \( \Gamma, \epsilon \vdash e_1 : \tau \) and \( \Gamma, \epsilon \vdash e_2 : \tau \).

By typing rules, \( \Gamma, \epsilon \vdash v : \tau \times \tau \).

\( (E-4) \) Then \( e = \text{fst } e_0 \) and \( e_0 \downarrow \forall \Delta (v, v_2) \).

By inversion, \( \tau = \tau \times \tau \) and \( \Gamma, \epsilon \vdash e_0 : \tau \).

By induction, \( \Gamma, \epsilon \vdash v : \tau \).

\( (E-6) \) Then \( e = \text{inl } e_0 \) and \( e_0 \downarrow \forall \Delta v \) and \( v = \text{inl } v_1 \).

By inversion, \( \tau = \tau_1 + \tau_2 \) and \( \Gamma, \epsilon \vdash e_0 : \tau \).

By induction, \( \Gamma, \epsilon \vdash e_0 : \tau \). By typing rules, \( \Gamma, \epsilon \vdash v : \tau_1 + \tau_2 \).

\( (E-8) \) Then \( e = \text{case } e_1 \text{ of } \text{ex } e_2 \mid x \downarrow e_3 \) and \( \epsilon \vdash v, \Gamma_{\epsilon} \vdash v : \tau \).

By inversion, \( \tau = \tau_1 + \tau_2 \) and \( \Gamma, \epsilon \vdash e_1 : \tau \).

By induction, \( \Gamma, \epsilon \vdash e_1 : \tau \). By inversion, \( \Gamma, \epsilon \vdash v : \tau_1 \).

By substitution, \( \Gamma, \epsilon \vdash ([v/x] e_2) \).

By induction, \( \Gamma, \epsilon \vdash v : \tau \).

\( (E-10) \) Then \( e = \text{fix } x = e_0 \) and \( \text{fix } x = e_0(x) \).

By inversion, \( \Gamma, \epsilon \vdash e : \tau \). By substitution and induction, \( \Gamma, \epsilon \vdash v : \tau \).
Progress: ...

One option: \( \forall e \in \mathbb{E}, \exists v \in \mathbb{V} \text{ s.t. } e \Downarrow v \)

True in STLC w/o fix but not in most real languages.

\( \exists v. \ \text{fix } x = x \uparrow v. \)

Big step can't talk about non-terminating expressions

No real way to talk about progress (\( \Rightarrow \) type safety).

But:
- Don't need to worry about evaluation order
- Don't need all the search rules

Thm. \( e \rightarrow^* v \iff e \Downarrow v \)

\( \iff \) Fairly straightforward with a couple annoying lemmas

\( \Rightarrow \) Suffices to show

Lemma: If \( e \rightarrow e' \) and \( e' \Downarrow v \) then \( e \Downarrow v \).
Cost Semantics

How long does a program take to run?

\[ e \rightarrow^* v \]

\[ e \rightarrow e_1 \rightarrow e_2 \rightarrow \ldots \rightarrow e_{n+1} v \]

\[ \underbrace{\ldots_n \text{ steps}} \]

\[ e \not\vdash v \text{ not so clear} \]

\[ e \vdash v \text{ "e evaluates to } v \text{ in } \text{time } n" \]

\[ e \vdash_{nv} \text{ e.e} \vdash_{nv} \text{ v(e)} \vdash_{nv} \]

\[ e_{1, e_2} \vdash_{nv} (v, v) \]

\[ \text{ Thm: } e \rightarrow^* v \text{ iff } e \vdash_{nv} v \]

Why is this useful?

Static cost analysis

\[ \text{ o.e.tin } \text{ - has type } o \text{ takes 3n steps to run} \]

If \[ o.e.tin \text{ and } e_{1, e_2}^n \text{ then } \text{m} \leq n \]

- Hard to show

If \[ o.e.tin \text{ and } e \vdash_{nv} v \text{ then } \text{m} \leq n \text{ easier} \]

\[ o.e.tin \Rightarrow e \vdash_{nv}^n \Rightarrow e \vdash_{nv}^\infty \]

We did the static analysis right

We did the cost semantics right