Two phrases of a programming language are ("Morris style") contextually equivalent ($\simeq_{ctx}$) if occurrences of the first phrase in any program can be replaced by the second phrase without affecting the observable results of executing the program.

Gottfried Wilhelm Leibniz (1646–1716): two mathematical objects are equal if there is no test to distinguish them.
Two phrases of a programming language are ("Morris style") contextually equivalent ($\sim_{ctx}$) if occurrences of the first phrase in any program can be replaced by the second phrase without affecting the observable results of executing the program.

Gottfried Wilhelm Leibniz (1646–1716): two mathematical objects are equal if there is no test to distinguish them.

need to define these terms (for ML)
• program $\triangleq$ well-typed expression with no free identifiers

• executing program $e$ in a given state $s \triangleq$
  finding $(v,s)$ such that $s,e \Rightarrow v,s$

• observable results of execution, $\text{obs}(v,s)$:
  $\text{obs}(c,s) \triangleq c$ if $c = \text{true}, \text{false}, n, ()$
  $\text{obs}(v_1, v_2, s) \triangleq \text{obs}(v_1, s), \text{obs}(v_2, s)$
  $\text{obs}(\text{fun}(a:ty) \rightarrow e) \triangleq \langle \text{fun} \rangle$
  $\text{obs}(\text{fun } f = (a: ty) \rightarrow e) \triangleq \langle \text{fun} \rangle$
  $\text{obs}(l,s) \triangleq \{\text{contents} = n\}$ if $(l \mapsto n) \in s$

• occurrence of an expression in a program...
ML Contexts \( C[-] \)

- ML syntax trees with a single sub-tree replaced by "hole", \(-\). E.g.
  \[
  \text{fun } (x : \text{int}) \Rightarrow x + (-)
  \]

- \( C[e] \triangleq \) expression resulting from replacing hole \(-\) by \( e \) in context \( C \)
  
  E.g. when \( C[-] \) is \( \text{fun } (x : \text{int}) \Rightarrow x + (-) \)
  then \( C[x] \) is \( \text{fun } (x : \text{int}) \Rightarrow x + x \)
ML Contexts $\mathcal{C}[-]$  

- ML syntax trees with a single sub-tree replaced by "hole", $\_$. E.g.
  
  \[
  \text{fun } (x : \text{int}) \rightarrow x + (-)
  \]

- $\mathcal{C}[e] \triangleq \text{expression resulting from replacing hole } \_ \text{ by } e \text{ in context } \mathcal{C}$

  E.g. when $\mathcal{C}[-]$ is $\text{fun } (x : \text{int}) \rightarrow x + (-)$
  
  then $\mathcal{C}[ex]$ is $\text{fun } (x : \text{int}) \rightarrow x + x$ capture!

- so can't identify contexts up to \( \alpha \)-equiv.
- complicates type assignment for contexts
ML Contextual Equivalence $\Gamma \vdash e_1 =_{ctx} e_2 : ty$

is defined to hold if:

- $\Gamma \vdash e_1 : ty$ and $\Gamma \vdash e_2 : ty$

- for all contexts $C[-]$ such that $C[e_1]$ & $C[e_2]$ are programs, and for all states $s$

  if $s, C[e_1] \Rightarrow v_1, s_1$

  then $s, C[e_2] \Rightarrow v_2, s_2$ with $obs(v_1, s_1) = obs(v_2, s_2)$

and vice versa.
ML Contextual Equivalence $\Gamma \vdash e_1 =_c e_2 : \text{ty}$

is defined to hold if:

- $\Gamma \vdash e_1 : \text{ty}$ and $\Gamma \vdash e_2 : \text{ty}$
- for all contexts $C[-]$ such that $C[e_1]$ & $C[e_2]$ are programs, and for all states $s$
  - if $s, C[e_1] \Rightarrow v_1, s_1$
  - then $s, C[e_2] \Rightarrow v_2, s_2$ with $\text{obs}(v_1, s_1) = \text{obs}(v_2, s_2)$
  - and vice versa.

Simplifying assumptions:
- only consider closed expressions (can use $e[\_ / x]$) as contexts
- only observe termination (doesn’t change $\Delta x \Rightarrow \text{Ex B.3}$)
Contextual preorder / equivalence

Given \( e_1, e_2 \in \text{Prog}_{ty} \), define

\[
\begin{align*}
  e_1 =_{\text{ctx}} e_2 : ty & \triangleq e_1 \leq_{\text{ctx}} e_2 : ty \quad \& \quad e_2 \leq_{\text{ctx}} e_1 : ty \\
  e_1 \leq_{\text{ctx}} e_2 : ty & \triangleq \forall x, e, ty', s. (x : ty \vdash e : ty') \quad \& \\
  & \quad s, e[e_1/x] \Downarrow \supset s, e[e_2/x] \Downarrow
\end{align*}
\]

where \( s, e \Downarrow \) indicates termination:

\[
\begin{align*}
  s, e \Downarrow & \triangleq \exists s', v (s, e \Rightarrow v, s')
\end{align*}
\]

Other natural choices of what to observe apart from termination do not change \( =_{\text{ctx}} \).

(see Exercise B.3)
Definition of $\Downarrow$ is not syntax-directed

\[ s', e_2[v_1/x] \Downarrow \]

E.g. \[ s, \text{let } x = e_1 \text{ in } e_2 \Downarrow \]

but \[ e_2[v_1/x] \]
is not built from subphrases of \textit{let } x = e_1 \text{ in } e_2. \]

Simple example of the difficulty this causes: consider a divergent integer expression \( \bot \triangleq (\text{fun } f = (x : \text{int}) \to f \ x) \ 0 \).

It satisfies \( \bot \leq_{\text{ctx}} n : \text{int} \), for any \( n \in \text{Prog}_{\text{int}} \).

Obvious strategy for proving this is to try to show

\[ s, e \Downarrow \supset \forall x, e'. e = e'[\bot/x] \supset s, e'[n/x] \Downarrow \]

by induction on the derivation of \( s, e \Downarrow \). But the induction steps are hard to carry out because of the above problem.
Felleisen-style presentation of \( \rightarrow \)

Lemma. \((s, e) \rightarrow (s', e')\) holds iff \(e = \mathcal{E}[r]\) and \(e' = \mathcal{E}[r']\) for some evaluation context \(\mathcal{E}\) and basic reduction \((s, r) \rightarrow (s', r')\).

Evaluation contexts are closed contexts that want to evaluate their hole (\(\mathcal{E} ::= - | \mathcal{E} e | \nu \mathcal{E} | \text{let } x = \mathcal{E} \text{ in } e | \cdots\)).

\(\mathcal{E}[r]\) denotes the expression resulting from replacing the ‘hole’ \([\_]\) in \(\mathcal{E}\) by the expression \(r\).

Basic reductions \((s, r) \rightarrow (s', r')\) are the axioms in the inductive definition of \(\rightarrow \) à la Plotkin—see Sect. A.5.

see (7) on p 387 for full definition
Fact. Every closed expression not in canonical form is uniquely of the form $E[r]$ for some evaluation context $E$ and redex $r$.

Fact. Every evaluation context $E$ is a composition $F_1[F_2[\cdots F_n[-] \cdots ]]$ of basic evaluation contexts, or evaluation frames.

Hence can reformulate transitions between configurations $(s, e) = (s, F_1[F_2[\cdots F_n[r] \cdots ]])$ in terms of transitions between configurations of the form

$$\langle s, F_s, r \rangle$$

where $F_s$ is a list of evaluation frames—the frame stack.
An ML abstract machine

Transitions

\[ \langle s, Fs, e \rangle \rightarrow \langle s', Fs', e' \rangle \]

\[ \begin{cases} s, s' & = \text{states} \\ Fs, Fs' & = \text{frame stacks} \\ e, e' & = \text{closed expressions} \end{cases} \]

defined by cases (i.e. no induction), according to the structure of \( e \) and (then) \( Fs \), for example:

\[ \langle s, Fs, \text{let } x = e_1 \text{ in } e_2 \rangle \rightarrow \langle s, Fs \circ (\text{let } x = [-] \text{ in } e_2), e_1 \rangle \]

\[ \langle s, Fs \circ (\text{let } x = [-] \text{ in } e), v \rangle \rightarrow \langle s, Fs, e[v/x] \rangle \]

(See Sect. A.6 for the full definition.)

Initial configurations: \( \langle s, Id, e \rangle \)

terminal configurations: \( \langle s, Id, v \rangle \)

(\( Id \) the empty frame stack, \( v \) a closed canonical form).
Theorem. \( \langle s, F_s, e \rangle \to^* \langle s', \text{Id}, v \rangle \) iff \( s, F_s[e] \Rightarrow v, s' \).

where

\[
\begin{align*}
\text{Id}[e] & \triangleq e \\
(F_s \circ F)[e] & \triangleq F_s[F[e]].
\end{align*}
\]

(tricky) Exercise — prove the theorem.
Theorem. \( \langle s, F_s, e \rangle \rightarrow^* \langle s', I_d, v \rangle \) iff \( s, F_s[e] \Rightarrow v, s' \).

where
\[
\begin{align*}
I_d[e] & \triangleq e \\
(F_s \circ F)[e] & \triangleq F_s[F[e]].
\end{align*}
\]

Hence: \( s, e \downarrow \) iff \( \exists s', v \ (\langle s, I_d, e \rangle \rightarrow^* \langle s', I_d, v \rangle) \).

So we can express termination of evaluation in terms of termination of the abstract machine. The gain is the following simple, but key, observation:

\( \downarrow \triangleq \{ \langle s, F_s, e \rangle \mid \exists s', v \ (\langle s, F_s, e \rangle \rightarrow^* \langle s', I_d, v \rangle) \} \)

has a direct, inductive definition following the structure of \( e \) and \( F_s \)—see Sect. A.7.
The relation we are interested in is a retract of a larger one with better structural properties.

\[ (s, e) \quad \Downarrow \quad \langle s, \text{Id}, e \rangle \]

\[ (s, \mathcal{F}s[e]) \quad \leftrightarrow \quad \langle s, \mathcal{F}s, e \rangle \]