Contextual equivalence

Two phrases of a programming language are ("Morris style") contextually equivalent ($\cong_{\mathrm{ctx}}$) 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.

ML Contextual equivalence

Two phrases of a programming language are ("Morris style") contextually equivalent ($\cong_{\mathrm{ctx}}$) 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 $\triangleq$ <u>well-typed</u> 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, $obs(v,s)$:

$$obs(c,s) \triangleq c \quad \text{if } c = true, false, n, ()$$
$$obs(v_1, v_2, s) \triangleq obs(v_1, s), obs(v_2, s)$$
$$obs(\mathbf{fun}\,(x:ty) \to e) \triangleq \langle fun \rangle$$
$$obs(\mathbf{fun}\ f = (x:ty) \to e) \triangleq \langle fun \rangle$$
$$obs(\ell, s) \triangleq \{contents = n\} \quad \text{if } (\ell \mapsto n) \in s$$

- occurrence of an expression in a program ...

<u>ML Contexts</u> $\mathcal{C}[-]$

- ML syntax trees with a single sub-tree replaced by "hole", $-$. E.g.

$$\mathbf{fun}\,(x:int) \to x + (-)$$

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

E.g. when $\mathcal{C}[-]$ is $\mathbf{fun}\,(x:int) \to x + (-)$
then $\mathcal{C}[x]$ is $\mathbf{fun}\,(x:int) \to x + x$

ML Contexts $\mathcal{C}[-]$

- ML syntax trees with a single sub-tree replaced by "hole", $-$. E.g.

  $\mathrm{fun}\,(x : \mathrm{int}) \to x + (-)$

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

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

— so can't identify contexts up to α-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 $\mathcal{C}[-]$ such that $\mathcal{C}[e_1]$ & $\mathcal{C}[e_2]$ are programs, and for all states s

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

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

and vice versa.

ML Contextual Equivalence $\Gamma \vdash e_1 =_{ctx} e_2 : ty$

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.

Simplifying assumptions:
- only consider closed expressions (can use $e[-/x]$ as contexts)
- only observe termination (doesn't change $=_{ctx}$ – Ex B.3)

Contextual preorder / equivalence

Given $e_1, e_2 \in \mathbf{Prog}_{ty}$, define

$$e_1 =_{\mathrm{ctx}} e_2 : ty \quad \triangleq \quad e_1 \leq_{\mathrm{ctx}} e_2 : ty \;\; \& \;\; e_2 \leq_{\mathrm{ctx}} e_1 : ty$$

$$e_1 \leq_{\mathrm{ctx}} e_2 : ty \quad \triangleq \quad \forall x, e, ty', s \,.\, (x : ty \vdash e : ty') \;\; \& \;\; s, e[e_1/x]\Downarrow \;\supset\; s, e[e_2/x]\Downarrow$$

where $s, e\Downarrow$ indicates termination:

$$s, e\Downarrow \;\triangleq\; \exists s', v\,(s, e \Rightarrow v, s')$$

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

(See Exercise B.3)

Definition of $\Downarrow$ is not syntax-directed

E.g. $\dfrac{s', e_2[v_1/x]\Downarrow}{s, \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 \Downarrow}$ if $s, e_1 \Rightarrow v_1, s'$

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

Simple example of the difficulty this causes: consider a divergent integer expression $\bot \triangleq (\mathtt{fun}\ f = (x : \mathtt{int})\ \texttt{->}\ f\ x)\ 0$.

It satisfies $\boxed{\bot \leq_{\mathrm{ctx}} n : \mathtt{int}, \text{ for any } n \in \mathrm{Prog}_{\mathtt{int}}}$

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 $\to$

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

Evaluation contexts are closed contexts that want to evaluate their hole ($\mathcal{E} ::= - \mid \mathcal{E}\ e \mid v\ \mathcal{E} \mid \mathtt{let}\ x = \mathcal{E}\ \mathtt{in}\ e \mid \cdots$).

$\mathcal{E}[r]$ denotes the expression resulting from replacing the 'hole' $[-]$ in $\mathcal{E}$ by the expression r.

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

see (7) on p387 for full definition

Fact. Every closed expression not in canonical form is uniquely of the form $\mathcal{E}[r]$ for some evaluation context $\mathcal{E}$ and redex r.

Fact. Every evaluation context $\mathcal{E}$ is a composition $\mathcal{F}_1[\mathcal{F}_2[\cdots \mathcal{F}_n[-]\cdots]]$ of basic evaluation contexts, or evaluation frames.

Hence can reformulate transitions between configurations $(s\ ,\ e) = (s\ ,\ \mathcal{F}_1[\mathcal{F}_2[\cdots \mathcal{F}_n[r]\cdots]])$ in terms of transitions between configurations of the form

$$\boxed{\langle s\ ,\ \mathcal{F}s\ ,\ r\rangle}$$

where $\mathcal{F}s$ is a list of evaluation frames—the frame stack.

An ML abstract machine

Transitions

$$\boxed{\langle s\ ,\ \mathcal{F}s\ ,\ e\rangle \to \langle s'\ ,\ \mathcal{F}s'\ ,\ e'\rangle}$$

$$\begin{cases} s, s' & \text{= states} \\ \mathcal{F}s, \mathcal{F}s' & \text{= frame stacks} \\ e, e' & \text{= closed expressions} \end{cases}$$

defined by cases (i.e. no induction), according to the structure of e and (then) $\mathcal{F}s$, for example:

$$\langle s\ ,\ \mathcal{F}s\ ,\ \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2\rangle \to \langle s\ ,\ \mathcal{F}s \circ (\mathtt{let}\ x = [-]\ \mathtt{in}\ e_2)\ ,\ e_1\rangle$$

$$\langle s\ ,\ \mathcal{F}s \circ (\mathtt{let}\ x = [-]\ \mathtt{in}\ e)\ ,\ v\rangle \to \langle s\ ,\ \mathcal{F}s\ ,\ e[v/x]\rangle$$

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

Initial configurations: $\langle s\ ,\ \mathcal{I}d\ ,\ e\rangle$
terminal configurations: $\langle s\ ,\ \mathcal{I}d\ ,\ v\rangle$
($\mathcal{I}d$ the empty frame stack, v a closed canonical form).

$$\boxed{\textbf{Theorem. } \langle s \ , \mathcal{F}s \ , e\rangle \to^* \langle s' \ , \mathcal{I}d \ , v\rangle \textit{ iff } s, \mathcal{F}s[e] \Rightarrow v, s'.}$$

where $\begin{cases} \mathcal{I}d[e] & \triangleq e \\ (\mathcal{F}s \circ \mathcal{F})[e] & \triangleq \mathcal{F}s[\mathcal{F}[e]]. \end{cases}$

(tricky) Exercise — prove the theorem.

$$\boxed{\textbf{Theorem. } \langle s \ , \mathcal{F}s \ , e\rangle \to^* \langle s' \ , \mathcal{I}d \ , v\rangle \textit{ iff } s, \mathcal{F}s[e] \Rightarrow v, s'.}$$

where $\begin{cases} \mathcal{I}d[e] & \triangleq e \\ (\mathcal{F}s \circ \mathcal{F})[e] & \triangleq \mathcal{F}s[\mathcal{F}[e]]. \end{cases}$

Hence: $\boxed{s, e \Downarrow \textit{ iff } \exists s', v \, (\langle s \ , \mathcal{I}d \ , e\rangle \to^* \langle s' \ , \mathcal{I}d \ , v\rangle).}$

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

$$\searrow \triangleq \{ \langle s \ , \mathcal{F}s \ , e\rangle \mid \exists s', v \left(\langle s \ , \mathcal{F}s \ , e\rangle \to^* \langle s' \ , \mathcal{I}d \ , v\rangle\right) \}$$

has a direct, inductive definition following the structure of e *and* $\mathcal{F}s$—see Sect. A.7.

The relation we are interested in *is a retract of* a larger one with better structural properties.

$$
\begin{tikzcd}[column sep=huge]
\Downarrow \arrow[r, hook, shift left] \arrow[d, hook] & \searrow \arrow[l, bend left] \arrow[d, hook] \\
\text{States} \times \text{Programs} \arrow[r, hook, shift left] & \text{States} \times \text{Frame Stacks} \times \text{Programs} \arrow[l, bend left]
\end{tikzcd}
$$

$$
\begin{aligned}
(s\ ,\ e) &\longmapsto \langle s\ ,\ \mathcal{I}d\ ,\ e\rangle \\
(s\ ,\ \mathcal{F}s[e]) &\longleftarrow\!\shortmid \langle s\ ,\ \mathcal{F}s\ ,\ e\rangle
\end{aligned}
$$