CK Abstract Machine

problem with operational semantics: steps contain hidden macro-operations

```
if (head [(true and true), false]) then 3 else 4
```

one step, but hidden in there is the search for the context

```
if (head [true, false]) then 3 else 4
```

also, on next step we start over from the top level “if” and search again for the context.

CK Abstract Machine:  
- - - - - - - - -

- make the steps for the search of the context explicit
  (no hidden macro operations)
- make the context to be syntactic first-class to store and remember
  (no going back to the top level “if”)
Language Transformation for generating:
the CK abstract machine
from an operational semantics

Generate the syntax for continuations

Continuations $k = []$
  | for each (op Xs) in $E$ do:
    $E$ in $Xs$ at $n$-th position: (op$_n$ (Xs - E) k)

Example for lambda-calculus:

Continuations $k = []$ | (app$_1$ E k) | (app$_2$ V k)
Generate the reduction rules: steps has the form $e, k \rightarrow e, k$

start computing op

for each (op_n Xs k) in k do:
    if $V$ is not in Xs then:
        (op (insert e in XS at n-th position)), $k \rightarrow e$, (op_n Xs k)

switch between eval contexts

if (op (insert v in XS at n-th position)) matches (op Xs') in E then
    E is in Xs' at m-th position:
    $V$, (op_n Xs k) $\rightarrow Xs[m]$, (op_m (insert v in Xs at $n$-th position $k$)

fire the computational rule for op

for each (op_n Xs k) in k do:
    for all reduction rules r s.t.
        (op (insert v in XS at n-th position)) matches LHS of r with subst
        apply_subst($V$, subst), (op_n apply_subst(Xs,subst) k) $\rightarrow$ RHS of r, k
Example of generating the reduction rules for a CK abstract machine, for lambda-calculus with pairs

Eval CTX E := □ | (app E e) | (app v E) | (pair E e) | (pair v E) | fst E | snd E

\[\text{app (lambda } x.\ e) v \rightarrow e[v/x]\]
\[\text{fst (pair } v1\ v2) \rightarrow v1\]
\[\text{snd (pair } v1\ v2) \rightarrow v2\]

apply #1
\[
\text{(app e1 e2), k }\rightarrow\ e1, \text{(app_1 e2 k)}
\]
\[
\text{(pair e1 e2), k }\rightarrow\ e1, \text{(pair_1 e2 k)}
\]
\[
\text{(fst e), k }\rightarrow\ e, \text{(fst_1 k)}
\]
\[
\text{(snd e), k }\rightarrow\ e, \text{(snd_1 k)}
\]
rule #2
selected: (app_1 e k): when v inserted in position 1 in Xs we have (app v e)
(app v e) matches an eval context (app v E), with m = 2.
(app_2 v k) is app_m with Xs in which v is inserted a position 1 and e is deleted.
Therefore:
\[ v, (app_1 e k) \rightarrow e, (app_2 v k) \]

selected: (pair_1 e k): when v inserted in position 1 in Xs we have (pair v e)
(pair v e) matches an eval context (pair v E), with m = 2.
(pair_2 v k) is app_m with Xs in which v is inserted a position 1 and e is deleted.
Therefore:
\[ v, (pair_1 e k) \rightarrow e, (pair_2 v k) \]
rule #3:
- selected: (app_2 v k). create (app v1 v2) with a “v” inserted in position 2
  notice we had to give distinct indexes.
  the v in #3 is then v2 below.
  v in (app_2 v k) is v1 then.
(app v1 v2) matches (app (lambda x.e) v) with subst = v1 => (lambda x.e) v2 => v

Therefore:
v, (app_2 (lambda x.e) k) —> e[v/x], k

- selected: (fst_1 k). create (fst v) with a “v” inserted in position 1.
(fst v) matches (fst (pair v1 v2)) with subst = v => (pair v1 v2)
Therefore:
(pair v1 v2), (fst_1 k) —> v1, k

- (snd_1 k) is similar to (fst_1 k).