⑦

$$\underline{\mathrm{cps}}(v, v_{cont}) = v_{cont}(v)$$

$$\underline{\mathrm{cps}}(e\ e', v_{cont}) = \underline{\mathrm{cps}}(e, \lambda f.\ \underline{\mathrm{cps}}(e', \lambda u.\ f\ u\ v_{cont}))$$

$$\underline{\mathrm{cps}}(\lambda v.\ e, v_{cont}) = v_{cont}(\lambda v.\ \lambda v'_{cont}\ \underline{\mathrm{cps}}(e, v'_{cont}))$$

$$\underline{\mathrm{cps}}(n, v_{cont}) = v_{cont}(n)$$

$$\underline{\mathrm{cps}}(-e, v_{cont}) = \underline{\mathrm{cps}}(e, \lambda u.\ v_{cont}(-u))$$

$$\underline{\mathrm{cps}}(e_0 + e_1, v_{cont}) = \underline{\mathrm{cps}}(e_0, \lambda u_0.\ \underline{\mathrm{cps}}(e_1, \lambda u_1.\ v_{cont}(u_0 + u_1)))$$

$$\underline{\mathrm{cps}}(\mathbf{if}\ e\ \mathbf{then}\ e'\ \mathbf{else}\ e'', v_{cont})$$

$$= \underline{\mathrm{cps}}(e, \lambda b.\ \mathbf{if}\ b\ \mathbf{then}\ \underline{\mathrm{cps}}(e', v_{cont})\ \mathbf{else}\ \underline{\mathrm{cps}}(e'', v_{cont}))$$

$$\underline{\mathrm{cps}}(\mathbf{true}, v_{cont}) = v_{cont}(\mathbf{true})$$

$$\underline{\mathrm{cps}}(\langle e_0, e_1\rangle, v_{cont}) = \underline{\mathrm{cps}}(e_0, \lambda u_0.\ \underline{\mathrm{cps}}(e_1, \lambda u_1.\ v_{cont}\langle u_0, u_1\rangle))$$

$$\underline{\mathrm{cps}}(e.k, v_{cont}) = \underline{\mathrm{cps}}(e, \lambda u.\ v_{cont}(u.k))$$

$$\underline{\mathrm{cps}}(\mathbf{letrec}\ v \equiv \lambda u.e\ \mathbf{in}\ e', v_{cont})$$

$$= \mathbf{letrec}\ v \equiv \lambda u.\lambda v'_{cont}.\ \underline{\mathrm{cps}}(e, v'_{cont}).\ \mathbf{in}\ \underline{\mathrm{cps}}(e', v_{cont})$$

for fresh v'_{cont}.

(Sorry. I'm less certain about this case)

⑧ Note that all function calls after the cps transformation are the applications of continuation variables to parameters. Since such variables represent the rest of computation, no calls leave anything to be done after they are completed. Thus, such calls can be implemented as jump, not as procedure call, by a compiler. Also, as mentioned before, the cps-transformed programs produce the same result regardless of whether we are using eager evaluation or normal-order evaluation. These observations indicate that cps-transformed programs or expressions are simpler than the original ones.

⑨ The transformation in ⑦ can be obtained systematically from the (continuation) semantics by removing η and all the embeddings and converting κ to the variable v_{cont}. This is so because they are closely related.