Evaluation of “puzzle” example from Taha thesis

This example comes from page 65 of Taha’s thesis:

```plaintext
- val puzzle = <Fn a => ~((Fn x => <x>)(Fn y => <a>)) 0>
- (run puzzle) 5
```

The example, in a MetaML-like language, is intended to illustrate a bug in the MetaML implementation, which is also present in the evaluation function ev1 given on page 62. ev1 applies to expressions in abstract syntax (i.e. elements of data type `exp`), but are given above in a concrete syntax; I have, however, used “Fn” for lambda expressions, to distinguish the above from expressions of the form “fn => ...”, which arise during evaluation; the latter are not concrete syntax for expressions in `exp`, but rather actual functions (in the underlying MetaML system).

First, let us evaluate the expression using a substitution semantics. Although order-of-evaluation may be significant in general, it can usually be ignored, as we will do here. The rules of the substitution semantics, with this simplification, are simply:

- **(β-reduction)** (Fn x => e1) e2 → e1[e2/x]
- **(Escape)** ~<e> → e
- **(Run)** run <e> → e

(The Escape and Run rules are identical here, because the only difference is the conditions under which they are applicable; i.e. it is basically a type-checking issue, which we are also ignoring here.)

According to these rules, we get this result:

```
(run <Fn a => ~(<Fn x => <x>)(Fn y => <a>)) 0>) 5
===> (Fn a => ~(<Fn x => <x>)(Fn y => <a>)) 0) 5
===> ~(<Fn x => <x>)(Fn y => <a>)) 0
===> (Fn x => x)(Fn y => <5>) 0
===> (Fn y => <5>) 0
===> <5>
```

We now go through the evaluation using ev1 and its auxiliary function eb1. Each step of the evaluation may have sub-evaluations, and each call to ev1 or eb1 ends with a line of the form “===> v” giving the result of that call (a value in the case of ev1, an expression in the case of eb1). This last line is given at the same indentation level as the original call to ev1 or eb1, while the sub-evaluations are indented.

1. `(run <Fn a => ~(<Fn x => <x>)(Fn y => <a>)) 0>) 5`
   - `(1) ev1 {} (run <Fn a => ~(<Fn x => <x>)(Fn y => <a>)) 0>) 5`
     - `(1.1) ev1 {} (run <Fn a => ~(<Fn x => <x>)(Fn y => <a>)) 0>)`
       - `(1.1.1) ev1 {} <Fn a => ~(<Fn x => <x>)(Fn y => <a>)) 0>
         - `(1.1.1.1) eb1 1 {} (Fn a => ~(<Fn x => <x>)(Fn y => <a>)) 0)
           - `(1.1.1.1.1) eb1 1 {a -> EV a1} ~(<Fn x => <x>)(Fn y => <a>)) 0)
             - `(1.1.1.1.1.1) ev1 {a -> EV a1} (Fn x => <x>)(Fn y => <a>)
               - `(1.1.1.1.1.1.1) ev1 {a -> EV a1} (Fn x => <x>)(Fn y => <a>)
                 - `(1.1.1.1.1.1.1.1) ev1 {a -> EV a1} (Fn x => <x>) (Fn y => <a>)
                   - `===> VF (fn v => ev1 {a -> EV a1, x -> EC v} <x>)
                     - `(1.1.1.1.1.1.1.1.1) ev1 {a -> EV a1} (Fn y => <a>)
                       - `===> VF (fn v => ev1 {a -> EV a1, y -> EC v} <a>)`
                         - `(1.1.1.1.1.1.1.1) f1 (VF f2) = ev1 {a -> EV a1, x -> EC (VF f2)} <x>
                           - `(1.1.1.1.1.1.1.1.1) eb1 1 {a -> EV a1, x -> EC (VF f2)} x
                             - `===> EC (VF f2)
                               - `===> VC (EC (VF f2))
```
The result is just the quoted variable <a1>, as stated in the thesis. The problem can be seen in the function f2: the connection between variable a1 and the argument to f2 has been lost. a should have been renamed, but since this function was captured in a MetaML function, it could not be. Thus, even though the function returned by (1.1) has a1 as its formal parameter, the body of f2 still refers to a.