Dependent open terms and the evaluation contexts that bind them

Oleg Kiselyov
FNMOC
oleg@pobox.com

Chung-chieh Shan
Rutgers University
cch@cs.rutgers.edu

How should we represent open terms and their binding contexts, especially in HOAS, and especially when one binding may depend on another? We came to this problem when formalizing the small-step semantics of staging so as to combine staging with effects in a sound type system. We describe our Twelf solution and call for a more elegant and natural representation.

Code generation is the most promising approach in high-performance computing (as in SPIRAL [5]) and high-assurance embedded programming (as in Hume [3]). Staged languages such as MetaOCaml are an attractive way to express such code generation. Code generation techniques like let insertion require either programming in CPS, which is cumbersome, or using effects such as state or delimited control, which risks scope extrusion: no staged language today has a type system that is sound with effects.

Yuki Yoshi Kameyama and us are studying the combination of staging and effects. The most straightforward way to formalize a language today has a type system that is sound with effects. We convinced Twelf that $\lambda$ represents evaluation contexts outside-in. We thus take advantage of HOAS to ensure $\alpha$-conversion invariance in a staged language [6], at the cost of obscuring the connection to defunctionalized continuations. We define in LF

1. the type $exp$ of a closed expression (program) $E$,
2. the type $ga$ of a typing environment $G$,
3. the type $eqaula G$ of an open level $\alpha$ bound by $G$,
4. the type $oexp \alpha$ of an open expression $0$ at $\alpha$ bound by $G$,
5. the type $ctxb \alpha$ of a context $C$ whose hole is at $\alpha$ and bound by $G$, and finally
6. the type $zip-up 0 C E$ of plugging $G$ into $C$ to yield $E$.

We convinced Twelf that $\lambda$ is $\alpha$-conversion invariance in a staged language [6], at the cost of obscuring the connection to defunctionalized continuations. We define in LF

1. the type $exp$ of a closed expression (program) $E$,
2. the type $ga$ of a typing environment $G$,
3. the type $eqaula G$ of an open level $\alpha$ bound by $G$,
4. the type $oexp \alpha$ of an open expression $0$ at $\alpha$ bound by $G$,
5. the type $ctxb \alpha$ of a context $C$ whose hole is at $\alpha$ and bound by $G$, and finally
6. the type $zip-up 0 C E$ of plugging $G$ into $C$ to yield $E$.

We convinced Twelf that $\lambda$ is $\alpha$-conversion invariance in a staged language [6], at the cost of obscuring the connection to defunctionalized continuations. We define in LF

1. the type $exp$ of a closed expression (program) $E$,
2. the type $ga$ of a typing environment $G$,
3. the type $eqaula G$ of an open level $\alpha$ bound by $G$,
4. the type $oexp \alpha$ of an open expression $0$ at $\alpha$ bound by $G$,
5. the type $ctxb \alpha$ of a context $C$ whose hole is at $\alpha$ and bound by $G$, and finally
6. the type $zip-up 0 C E$ of plugging $G$ into $C$ to yield $E$.

We convinced Twelf that $\lambda$ is $\alpha$-conversion invariance in a staged language [6], at the cost of obscuring the connection to defunctionalized continuations. We define in LF

1. the type $exp$ of a closed expression (program) $E$,
2. the type $ga$ of a typing environment $G$,
3. the type $eqaula G$ of an open level $\alpha$ bound by $G$,
4. the type $oexp \alpha$ of an open expression $0$ at $\alpha$ bound by $G$,
5. the type $ctxb \alpha$ of a context $C$ whose hole is at $\alpha$ and bound by $G$, and finally
6. the type $zip-up 0 C E$ of plugging $G$ into $C$ to yield $E$.

We convinced Twelf that $\lambda$ is $\alpha$-conversion invariance in a staged language [6], at the cost of obscuring the connection to defunctionalized continuations. We define in LF

1. the type $exp$ of a closed expression (program) $E$,
2. the type $ga$ of a typing environment $G$,
3. the type $eqaula G$ of an open level $\alpha$ bound by $G$,
4. the type $oexp \alpha$ of an open expression $0$ at $\alpha$ bound by $G$,
5. the type $ctxb \alpha$ of a context $C$ whose hole is at $\alpha$ and bound by $G$, and finally
6. the type $zip-up 0 C E$ of plugging $G$ into $C$ to yield $E$.

We convinced Twelf that $\lambda$ is $\alpha$-conversion invariance in a staged language [6], at the cost of obscuring the connection to defunctionalized continuations. We define in LF

1. the type $exp$ of a closed expression (program) $E$,
2. the type $ga$ of a typing environment $G$,
3. the type $eqaula G$ of an open level $\alpha$ bound by $G$,
4. the type $oexp \alpha$ of an open expression $0$ at $\alpha$ bound by $G$,
5. the type $ctxb \alpha$ of a context $C$ whose hole is at $\alpha$ and bound by $G$, and finally
6. the type $zip-up 0 C E$ of plugging $G$ into $C$ to yield $E$.

We convinced Twelf that $\lambda$ is $\alpha$-conversion invariance in a staged language [6], at the cost of obscuring the connection to defunctionalized continuations. We define in LF

1. the type $exp$ of a closed expression (program) $E$,
2. the type $ga$ of a typing environment $G$,
3. the type $eqaula G$ of an open level $\alpha$ bound by $G$,
4. the type $oexp \alpha$ of an open expression $0$ at $\alpha$ bound by $G$,
5. the type $ctxb \alpha$ of a context $C$ whose hole is at $\alpha$ and bound by $G$, and finally
6. the type $zip-up 0 C E$ of plugging $G$ into $C$ to yield $E$.