Taming Nontermination

Recovering Free Theorems with Linearity

Nick Rioux Amal Ahmed

July 20, 2017

College of Computer and Information Science
Northeastern University

Free Theorems

Free Theorems

$$\forall\alpha.\, \alpha \to \alpha$$

e.g. $\mathrm{id} = \Lambda\alpha.\, \lambda(x : \alpha).\, x$

Free Theorems

$$\forall\alpha.\,\alpha \to \alpha$$

e.g. $\mathrm{id} = \Lambda\alpha.\,\lambda(x : \alpha).\,x$

In fact, id is the only inhabitant of this type!

Free Theorems

$$\forall \alpha.\, \alpha \to \alpha \to \alpha$$

Free Theorems

$$\forall \alpha.\, \alpha \to \alpha \to \alpha$$

- $\Lambda \alpha.\, \lambda(x : \alpha).\, \lambda(y : \alpha).\, x$

Free Theorems

$$\forall \alpha.\, \alpha \to \alpha \to \alpha$$

- $\Lambda\alpha.\, \lambda(x : \alpha).\, \lambda(y : \alpha).\, x$
- $\Lambda\alpha.\, \lambda(x : \alpha).\, \lambda(y : \alpha).\, y$

Linear Types

Linear Types

- A *linear* function uses its argument exactly once.

Linear Types

- A *linear* function uses its argument exactly once.
- We can enforce this with a linear function type:

$$\tau \multimap \tau'$$

Linear Types

- A *linear* function uses its argument exactly once.
- We can enforce this with a linear function type:

$$\tau \multimap \tau'$$

- For example:
  - $\vdash (\lambda(f : \mathrm{bool} \multimap \mathrm{Int}).\, f\ \mathrm{true}) : (\mathrm{bool} \multimap \mathrm{Int}) \multimap \mathrm{Int}$ but

Linear Types

- A *linear* function uses its argument exactly once.
- We can enforce this with a linear function type:

$$\tau \multimap \tau'$$

- For example:
  - $\vdash (\lambda(f : \mathrm{bool} \multimap \mathrm{Int}).\, f\ \mathrm{true}) : (\mathrm{bool} \multimap \mathrm{Int}) \multimap \mathrm{Int}$ but
  - $\lambda(f : \mathrm{bool} \multimap \mathrm{Int}).\, f\ \mathrm{true}; f\ \mathrm{false}$ **cannot** be given a $\multimap$ type.

Linear Types: Preservation of Resources

- Linear functions preserve 'resources' (*i.e.* free variables)
- Zhao, Zhang, and Zdancewic '10 (ZZZ) show this can be useful.

Linear Types: Preservation of Resources

- Linear functions preserve 'resources' (*i.e.* free variables)
- Zhao, Zhang, and Zdancewic '10 (ZZZ) show this can be useful.

Suppose

$$f\ v \longmapsto \cdots \longmapsto e$$

Then

free linear variables in $f\ v$ = free linear variables in e

Linear Free Theorems

$$\forall \alpha .\, \alpha \multimap \alpha \multimap \alpha$$

There are no values of this type!

$$\forall \alpha.\, \alpha \multimap \alpha \multimap \alpha$$

There are no values of this type! But how do we prove this?

Linear Free Theorems

ZZZ approach:

1. Prove the usual free theorem (ignoring linearity).
   Any inhabitants of $\forall\alpha.\,\alpha \multimap \alpha \multimap \alpha$ must be equivalent to:
   - $f_1 \approx \Lambda\alpha.\,\lambda(x:\alpha).\,\lambda(y:\alpha).\,x$ or
   - $f_2 \approx \Lambda\alpha.\,\lambda(x:\alpha).\,\lambda(y:\alpha).\,y$

Linear Free Theorems

ZZZ approach:

1. Prove the usual free theorem (ignoring linearity).
   Any inhabitants of $\forall\alpha.\,\alpha\multimap\alpha\multimap\alpha$ must be equivalent to:
   - $f_1 \approx \Lambda\alpha.\,\lambda(x:\alpha).\,\lambda(y:\alpha).\,x$ or
   - $f_2 \approx \Lambda\alpha.\,\lambda(x:\alpha).\,\lambda(y:\alpha).\,y$
2. Identify inhabitants that don't preserve free linear variables.
   Applying f_1 to free variables x_1 and x_2,

   $$f_1\ x_1\ x_2 \approx x_1$$

   But x_2 was lost! (Similar problem with f_2.)

Linear Free Theorems

ZZZ approach:

1. Prove the usual free theorem (ignoring linearity).
   Any inhabitants of $\forall\alpha.\, \alpha \multimap \alpha \multimap \alpha$ must be equivalent to:
   - $f_1 \approx \Lambda\alpha.\, \lambda(x : \alpha).\, \lambda(y : \alpha).\, x$ or
   - $f_2 \approx \Lambda\alpha.\, \lambda(x : \alpha).\, \lambda(y : \alpha).\, y$
2. Identify inhabitants that don't preserve free linear variables.
   Applying f_1 to free variables x_1 and x_2,
   $$f_1\ x_1\ x_2 \approx x_1$$
   But x_2 was lost! (Similar problem with f_2.)
3. **Contradiction**: since linear functions must preserve resources,
   f_1 and f_2 cannot be well-typed linear functions.

Compositional Compiler Correctness

CPS

We're interested in the correctness of CPS translations.

$$(e \ : \ \tau) \rightsquigarrow (e_{CPS} \ : \ \forall \alpha. (\tau \to \alpha) \to \alpha)$$

CPS

We're interested in the correctness of CPS translations.

$$(e \ : \ \tau) \rightsquigarrow (e_{CPS} \ : \ \forall\alpha.\,(\tau \to \alpha) \to \alpha)$$

e.g. $(\mathrm{true} : \mathrm{bool}) \rightsquigarrow (\Lambda\alpha.\,\lambda(k : \mathrm{bool} \to \alpha).\,k\ \mathrm{true}$
$: \forall\alpha.\,(\mathrm{bool} \to \alpha) \to \alpha)$

Continuation Shuffling

- Correctness depends on a key free theorem for the type $\forall\alpha.\,(\tau \to \alpha) \to \alpha$ called *continuation shuffling*.

Continuation Shuffling

- Correctness depends on a key free theorem for the type $\forall\alpha.\,(\tau \to \alpha) \to \alpha$ called *continuation shuffling*.
- **Problem:** Continuation shuffling breaks in languages with nontermination.

Continuation Shuffling

- Correctness depends on a key free theorem for the type $\forall\alpha.\,(\tau \to \alpha) \to \alpha$ called *continuation shuffling*.
- **Problem:** Continuation shuffling breaks in languages with nontermination.
  - Reason: a term of type $\forall\alpha.\,(\tau \to \alpha) \to \alpha$ may use its continuation multiple times.

Continuation Shuffling

- Correctness depends on a key free theorem for the type $\forall\alpha.\,(\tau \to \alpha) \to \alpha$ called *continuation shuffling*.
- **Problem:** Continuation shuffling breaks in languages with nontermination.
  - Reason: a term of type $\forall\alpha.\,(\tau \to \alpha) \to \alpha$ may use its continuation multiple times.
  - But our translation never produces such terms!

Continuation Shuffling

- Correctness depends on a key free theorem for the type $\forall\alpha.\,(\tau \to \alpha) \to \alpha$ called *continuation shuffling*.
- **Problem:** Continuation shuffling breaks in languages with nontermination.
  - Reason: a term of type $\forall\alpha.\,(\tau \to \alpha) \to \alpha$ may use its continuation multiple times.
  - But our translation never produces such terms!
- **Solution:** Change the CPS target type to $\forall\alpha.\,(\tau \to \alpha) \multimap \alpha$

Continuation Shuffling

"Solution": Change the CPS target type to $\forall\alpha.\,(\tau \to \alpha) \multimap \alpha$

- Now we need a proof technique that can take advantage of the linearity to prove continuation shuffling.

Continuation Shuffling

“Solution”: Change the CPS target type to $\forall\alpha.\,(\tau \to \alpha) \multimap \alpha$

- Now we need a proof technique that can take advantage of the linearity to prove continuation shuffling.
- But this is hard!

Continuation Shuffling

“Solution”: Change the CPS target type to $\forall\alpha.\,(\tau \to \alpha) \multimap \alpha$

- Now we need a proof technique that can take advantage of the linearity to prove continuation shuffling.
- But this is hard!
- “Resources in = resources out” isn’t enough.
    - ZZZ approach requires a non-linear free theorem as a starting point, but we don’t even have that!

Continuation Shuffling

"Solution": Change the CPS target type to $\forall\alpha.\,(\tau \to \alpha) \multimap \alpha$

- Now we need a proof technique that can take advantage of the linearity to prove continuation shuffling.
- But this is hard!
- "Resources in = resources out" isn't enough.
  - ZZZ approach requires a non-linear free theorem as a starting point, but we don't even have that!
- We need to model what happens when free variables are used.
  - Then we can reason about the state of the program at the point that a continuation is used.

Conclusion

- There is a deep interaction between parametricity, linearity, and effects that we don't fully understand.

Conclusion

- There is a deep interaction between parametricity, linearity, and effects that we don't fully understand.
- We would like to be able to reason about how linearity can tame effects because, in many situations, this is what linearity is all about.

Qed.