Mechanical proof of type soundness for L^3

L^3: *The Linear Language with Locations*

Michael Sproul

Supervisor: Ben Lippmeier
University of New South Wales

Introduction

What?

1. Model the *type system* and *semantics* of a small programming language as a formal mathematical system.
2. Prove that the programming language is "well-behaved" – *sound*.

Motivation

Programming languages are complicated, lots of interacting features.

We want to:

- Eliminate or minimise undefined behaviour.
- Ensure that language features interoperate without creating unsoundness.
- Provide a solid foundation for software verification.

Motivation

Formalised language semantics are a prerequisite for:

- Proofs about programs. *seL4*, verified crypto.
  - Security, correctness.
- Trust-worthy compilers. *CompCert*.
- Sanity whilst programming?

Background

Contraction and Weakening

In regular* logic there are two rules that allow assumptions to be discarded, or introduced from nowhere:

$$\frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B}\ \text{Contraction} \qquad \frac{\Gamma \vdash B}{\Gamma, A \vdash B}\ \text{Weakening}$$

*regular = classical/intuitionistic/constructive

Contraction and Weakening

Every logic corresponds to a type system, so we have...

$$\frac{\Gamma, y : A, z : A \vdash u :: B}{\Gamma, x : A \vdash u[x/y, x/z] :: B}\ \text{Contraction} \qquad \frac{\Gamma \vdash y :: B}{\Gamma, x : A \vdash y :: B}\ \text{Weakening}$$

Without contraction, variables can only appear once in a term.

Without weakening, all variables in the context must be used in the term.

[Wadler 1991, 1993]

Linear and affine type systems

- **Linear type system:** Contraction and weakening banned – every value used *exactly* once.
- **Affine type system:** Contraction banned, weakening allowed – every value used *at most* once.

From linear logic [Girard 1986] and affine logic respectively.

Retaining intuitionistic behaviour

Linear type systems allow some terms to be treated in the normal intuitionistic way, via the bang operator, (!).

$$\varnothing \vdash \lambda\langle x' \rangle.\ \mathsf{case}\ x' \mathsf{of}\ !x \to \langle !x, !x \rangle :\ \ !A \multimap (!A \otimes !A)$$

Derivable using a version of contraction that requires *non-linear* assumptions [Wadler 1993].

Why bother?

Several languages use type systems based on linear logic to provide useful features:

Mutation without sacrificing referential transparency (Clean).

$$
\begin{array}{lcl}
\text{qs} :: \underline{[\ \text{T}\ ]} \to \underline{[\ \text{T}\ ]} & & \\
\text{qs}\ [\ ] & = & [\ ] \\
\text{qs}\ [\ \text{hd} \mid \text{tl}\ ] & = & (\text{qs left}) +\!\!+ [\ \text{hd} \mid \text{qs right}\ ] \\
 & & \text{where} \\
 & & (\text{left}, \text{right}) = \text{split tl hd}
\end{array}
$$

$$
\begin{array}{lcl}
\text{split} :: \underline{[\ \text{T}\ ]} \to \text{T} \to \underline{([\ \text{T}\ ], [\ \text{T}\ ])} & & \\
\text{split}\ [\ ]\ \text{p} & = & ([\ ], [\ ]) \\
\text{split}\ [\ \text{hd} \mid \text{tl}\ ]\ \text{p} & = & ([\ \text{hd} \mid \text{left}\ ], \text{right}), \quad \text{if p} \geq \text{hd} \\
 & = & (\text{left}, [\ \text{hd} \mid \text{right}\ ]) \\
 & & \text{where} \\
 & & (\text{left}, \text{right}) = \text{split tl p}
\end{array}
$$

Why bother?

Memory safety without garbage collection –
Cyclone [Grossman et al. 2005], Rust.

```
fn main() {
    // Heap allocated vector - but we don't need to free it!
    let mut v = Vec::new();

    for i in 0 .. 200 {
        v.push(i * i);
    }

    println!("{:?}", v);
}
```

Uniqueness typing

Linear (and affine) typing aren't quite enough to guarantee unique references to values.

Dereliction allows a non-linear value to be used as a linear one.

$$\lambda x \cdot x : \quad \alpha^{\times} \to \alpha^{\odot} \quad \text{(linear dereliction)}$$
$$\lambda x \cdot x : \quad \alpha^{\odot} \to \alpha^{\times} \quad \text{(uniqueness removal)}$$

In a pure linear (or affine) type system, uniqueness of linear values therefore isn't guaranteed [de Vries 2008].

Uniqueness typing

- **Linear/affine typing**: Linear values *will not* be duplicated at any point in the future.
- **Uniqueness typing**: Linear values *have not* been duplicated at any point in the past.

Uniqueness typing via attributes

Edsko de Vries designed a system that uses a uniqueness *kind*, and a type constructor `Attr` which annotates a type as either unique or non-unique.

Kind language

κ	$::=$		kind
		$\mathcal{T}$	base type
		$\mathcal{U}$	uniqueness attribute
		$*$	base type together with a uniqueness attribute
		$\kappa_1 \rightarrow \kappa_2$	type constructors

Type constants

`Int`, `Bool`	$::$	$\mathcal{T}$	base type
$\rightarrow$	$::$	$* \rightarrow * \rightarrow \mathcal{T}$	function space
$\bullet$, $\times$	$::$	$\mathcal{U}$	unique, non-unique
$\vee$, $\wedge$	$::$	$\mathcal{U} \rightarrow \mathcal{U} \rightarrow \mathcal{U}$	disjunction, conjunction
$\neg$	$::$	$\mathcal{U} \rightarrow \mathcal{U}$	negation
`Attr`	$::$	$\mathcal{T} \rightarrow \mathcal{U} \rightarrow *$	combine a base type and attribute

Syntactic conventions

t^u	$\equiv$	`Attr` t u
$a \xrightarrow{u} b$	$\equiv$	`Attr` $(a \rightarrow b)$ u

Uniqueness typing formalisation

de Vries used Coq to prove soundness for his attribute-based type system.

- Locally nameless approach used for variable naming – de Bruijn indices for bound variables and names for free variables.
- *The use of the locally nameless approach, and in particular the use of the Formal Metatheory library, meant that little of our subject reduction proof needs to be concerned with alpha-equivalence or freshness.*

The L^3 language

Based on the linear lambda calculus, supporting *strong updates* via a system of capabilities [Ahmed, Fluet, Morrisett 2001].

```
let val r = ref () in
   r := true;
   if (!r) then
     r := 42
   else
     r := 15;
   !r + 12
end
```

The L^3 language

Pointers can be shared, but require a *capability* to be read, written or discarded. The capability is treated *linearly*.

$$(\sigma, \mathsf{new}\ v) \Rightarrow (\sigma \uplus \{l \mapsto v\}, \ulcorner l, \langle \mathsf{cap}, \mathsf{ptr}\ l \rangle \urcorner)$$
$$(\sigma, \mathsf{let}\ \ulcorner \rho, x \urcorner = \ulcorner l, v \urcorner\ \mathsf{in}\ e) \Rightarrow (\sigma, e[l/\rho][v/x])$$

Different from de Vries' system in that neither *dereliction* nor removal of uniqueness is allowed.

The L^3 language

Swapping a value can change its type:

$$(\sigma \uplus \{l \mapsto v_1\}, \mathsf{swap\ cap\ (ptr}\ l)\ v_2) \Rightarrow (\sigma \uplus \{l \mapsto v_2\}, \langle \mathsf{cap}, v_1 \rangle)$$

$$(\mathsf{Swap})\ \frac{\Delta; \Gamma_1 \vdash e_1 : \mathsf{Cap}\ \rho\ \tau_1 \quad \Delta; \Gamma_2 \vdash e_2 : \mathsf{Ptr}\ \rho \qquad \Delta; \Gamma_2 \vdash e_3 : \tau_3}{\Delta; \Gamma_1, \Gamma_2, \Gamma_3 \vdash \mathtt{swap}\ e_1\ e_2\ e_3 : \mathsf{Cap}\ \rho\ \tau_3 \otimes \tau_1}$$

L^3 has a successor...

[Ahmed, Fluet and Morrisett 2006] verified a system similar to L^3, with regions added on, **but without strong updates**.

- Twelf proof assistant, rather than Coq.
- Higher-order abstract syntax – host language binders used to describe object language binders.

Their conclusion from 2006 still seems to be true, **nobody has verified a system with strong updates yet**.

Proposal

Proposal

Model the operational semantics and type system of L^3 core using Coq, and prove *progress* and *preservation*.

$$\frac{\varnothing \vdash x : \tau}{\text{value } x \vee \exists x'.\ x \Rightarrow x'} \text{ Progress}$$

$$\frac{\varnothing \vdash x : \tau \quad x \Rightarrow x'}{\varnothing \vdash x' : \tau} \text{ Preservation}$$

What have I done so far?

- Background reading on semantics and type theory.
- Literature review - mostly papers on linear-logic based type systems.
- Coq proofs of type soundness for simple variants of the lambda calculus, via *Software Foundations* [Pierce 2015].

Software Foundations

- STLC with let-bindings, fixed points, sum types and product types.
- *Globally shared names for variables* rather than HOAS, de Bruijn indices or locally nameless.

```
Lemma substitution_preserves_typing :
  forall Gamma x U t v T,
    extend Gamma x U |- t \in T ->
    empty |- v \in U   ->
    Gamma |- [x:=v]t \in T.
```

Approaches to naming

SF avoids name capture by only permitting substitution of closed terms.

For L^3 I'll need to pick a better approach to variable naming.

- Higher-order Abstract Syntax (HOAS).
- de Bruijn indices [de Bruijn 1972].
- Locally nameless, as used by [de Vries 2008].

Schedule

Before Week 12 (literature review deadline):

- Investigate approaches to variable binding (3 weeks).
- Finish reading and summarising papers (also 3 weeks).

Schedule

Before Week 12 (literature review deadline):

- Investigate approaches to variable binding (3 weeks).
- Finish reading and summarising papers (also 3 weeks).

Over summer (3 months $\approx$ 12 weeks total):

- L^3's operational semantics and type system in Coq (2 weeks).
- Prove progress (4 weeks).
- Prove preservation, and related lemmas (6 weeks).

Definitions may need tweaking along the way.

Schedule

Next semester (12 weeks):

- Finalise proofs completed over the summer break (6 weeks).
- Write-up results and findings (6 weeks).

References

- *Lambda Calculus Notation with Nameless Dummies* - N. G. de Bruijn (1972).
- *Linear Logic* - J. Girard (1986).
- *Higher-order abstract syntax* - F. Pfenning and C. Elliot (1988).
- *Linear types can change the world!* - P. Wadler (1991).
- *A taste of linear logic* - P. Wadler (1993).
- *Guaranteeing Safe Destructive Updates through a Type System with Uniqueness Information for Graphs* - S. Smetsers et al (1994).
- L^3*: A Linear Language with Locations* - A. Ahmed, M. Fluet, G. Morrisett (2001).
- *Linear Regions Are All You Need* - A. Ahmed, M. Fluet, G. Morrisett (2006).
- *Uniqueness Typing Simplified* - E. de Vries (2008).
- *Making Uniqueness Typing Less Unique* - E. de Vries (2008).
- *Software Foundations* - B. Pierce (2015).

Thanks!