Bachelor Proposals

Programming Languages, Logic, and Software Security

Andreas Pavlogiannis

November, 2025

AARHUS UNIVERSITY

Programming Languages, Logic, and Software Security

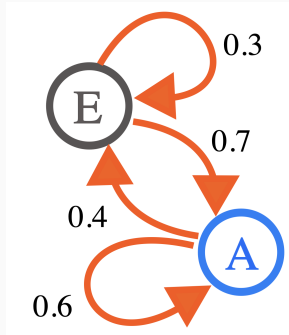
The Overarching Question

How do we make computer systems provably safe, reliable, secure and performant?

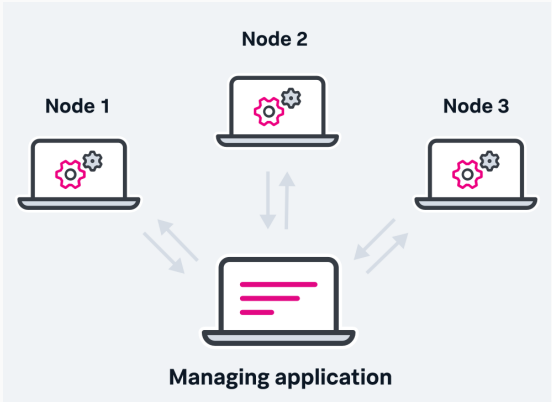
The Overarching Question

How do we make computer systems provably safe, reliable, secure and performant?

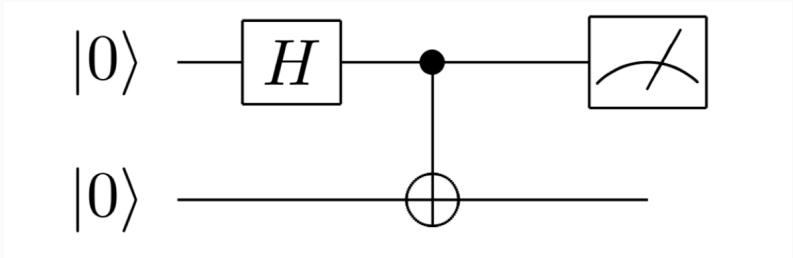
Programming language design



Probabilistic systems



Concurrency



Quantum circuits & programs

Research Areas

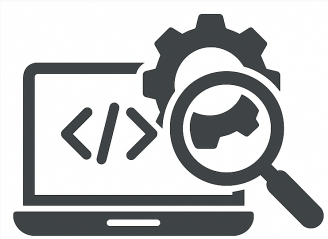
$\Gamma \vdash e : \tau$

Type systems

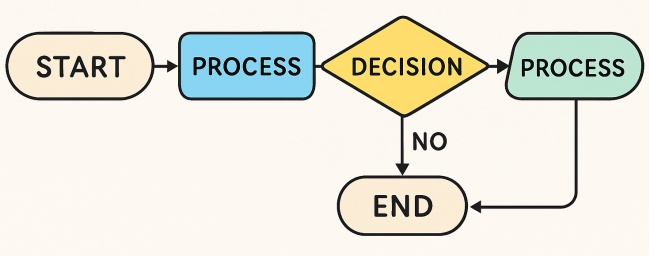
Iris

Logics

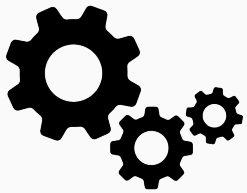
Proof assistants



Static program analysis



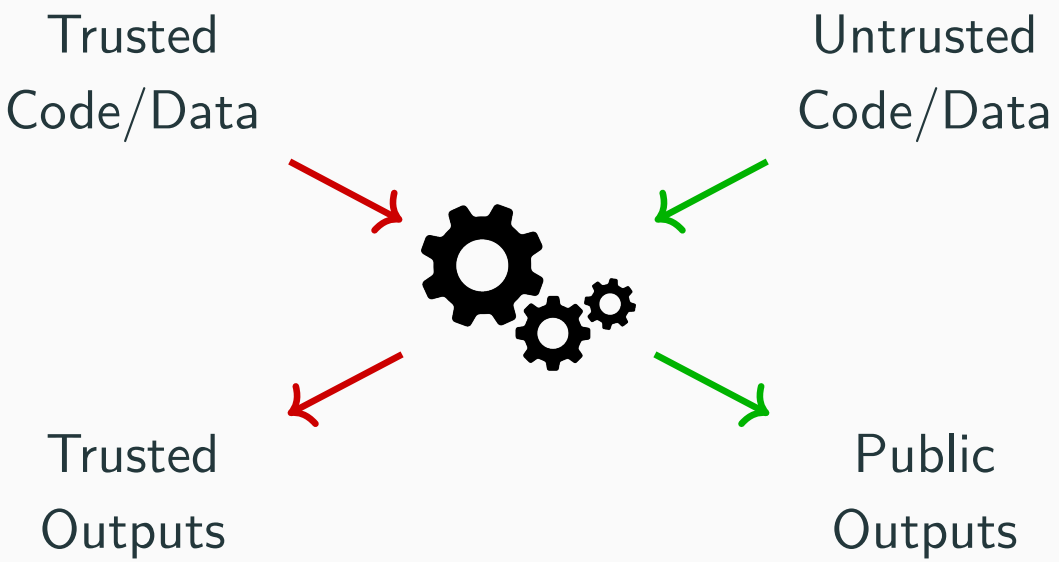
Algorithms & complexity



Automated reasoning (SAT/SMT)

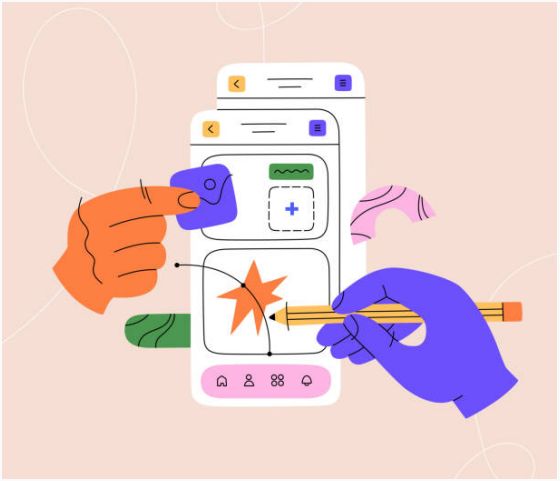
Troupe: A Privacy-Aware Programming Language

A language with runtime security monitoring



Example Projects in Troupe

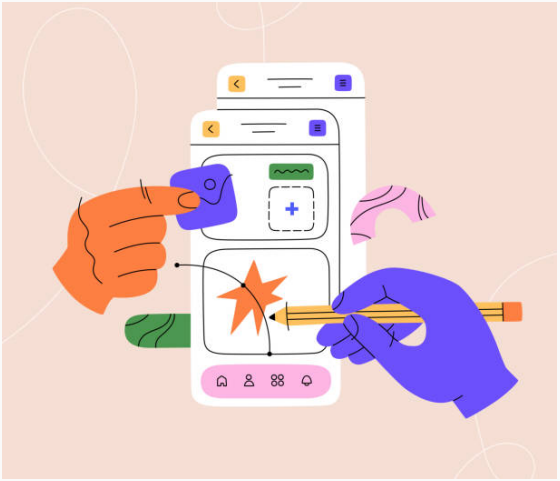
Privacy Aware User Interface



How do we make UIs
adaptive to privacy environments?

Example Projects in Troupe

Privacy Aware User Interface

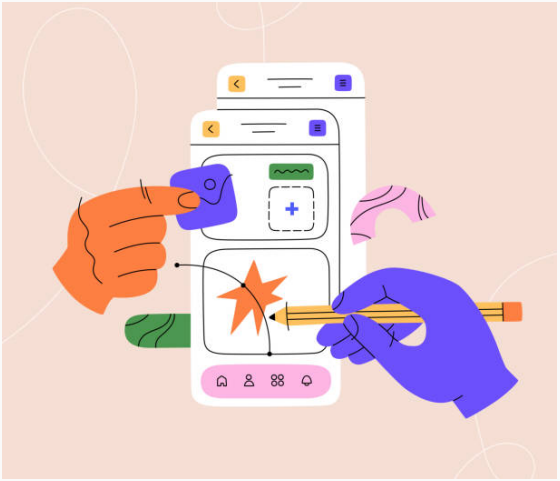


How do we make UIs
adaptive to privacy environments?

- Develop a UI library in Troupe
- Context-aware privacy
- Showcase on a messaging app

Example Projects in Troupe

Privacy Aware User Interface



How do we make UIs
adaptive to privacy environments?

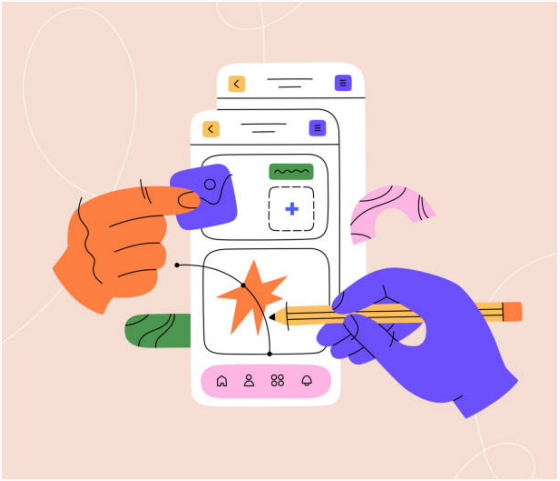
Secure Agentic Coding

Can AI generate code preserving
security/privacy constraints?

- Develop a UI library in Troupe
- Context-aware privacy
- Showcase on a messaging app

Example Projects in Troupe

Privacy Aware User Interface



How do we make UIs
adaptive to privacy environments?

- Develop a UI library in Troupe
- Context-aware privacy
- Showcase on a messaging app

Secure Agentic Coding

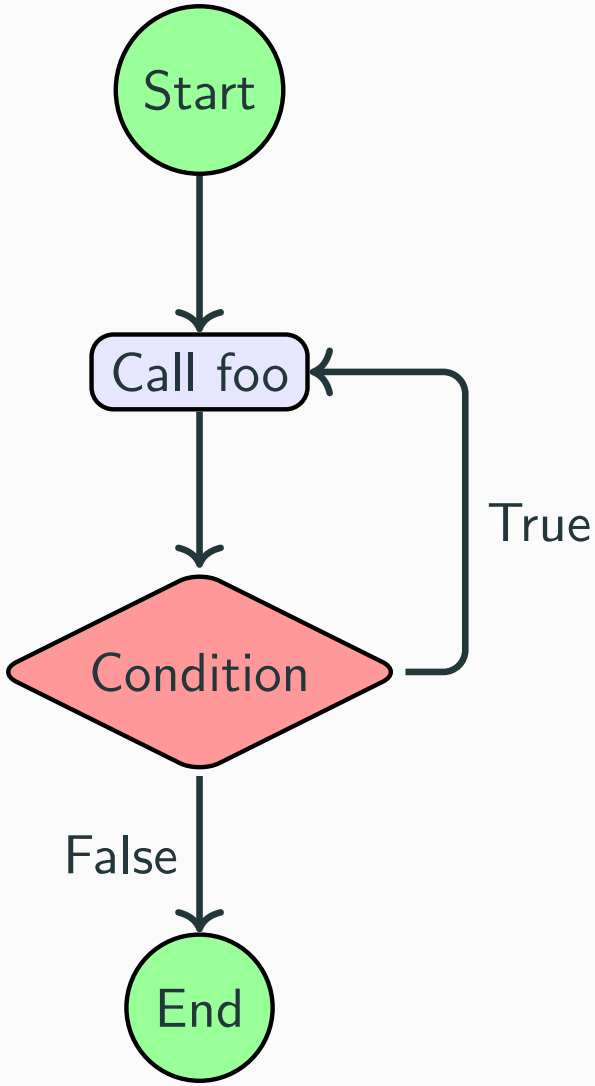
Can AI generate code preserving
security/privacy constraints?

- LLM code generation for Troupe
- Incorporate security concepts lacking in training

Termination Analysis

Termination Analysis

Termination Analysis



Termination Analysis in Flix

- Algebraic data types
- Type inference
- Polymorphic effects
- First-class Datalog constraints

```
// Termination via recursion on structurally smaller data
def map(f: a->b, l: List[a]): List[b] =
    match l {
        case Nil     => Nil
        case x :: xs => f(x) :: map(f,xs)
    }
```

- What methods exist for termination checking?
- How to implement in the Flix compiler?

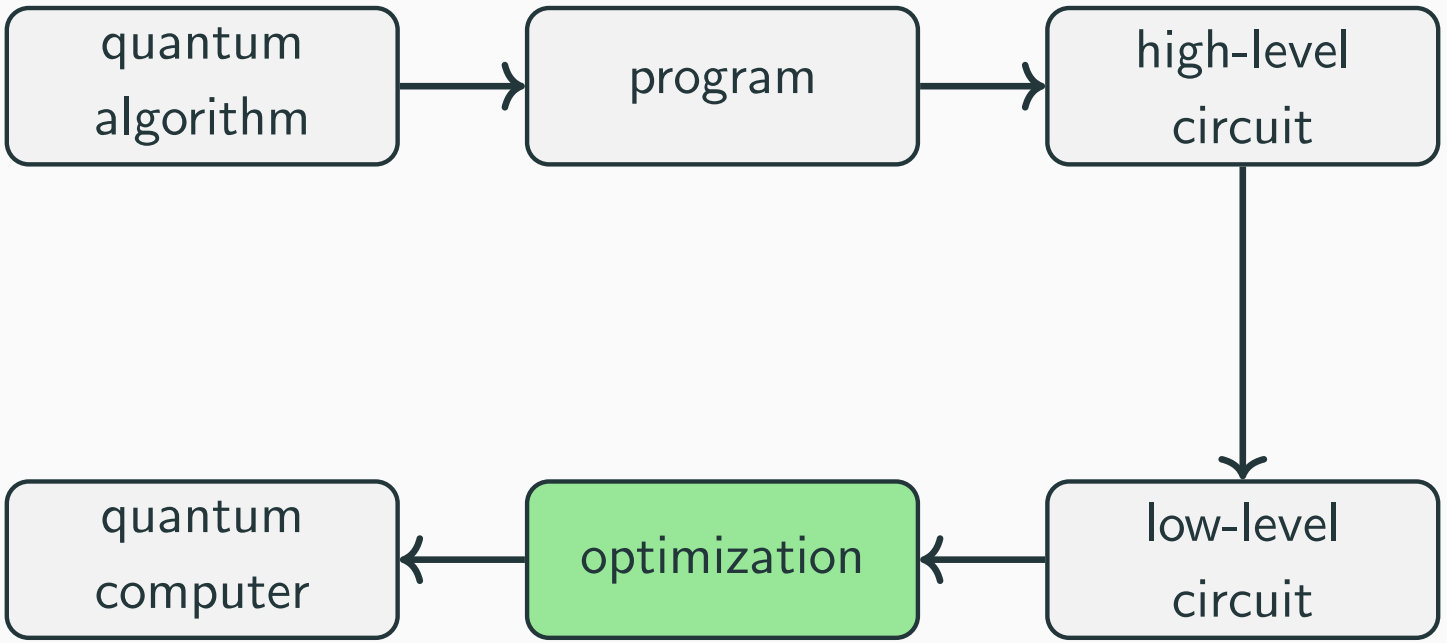
Other Projects in Flix

- Algebraic data types
- Type inference
- Polymorphic effects
- First-class Datalog constraints

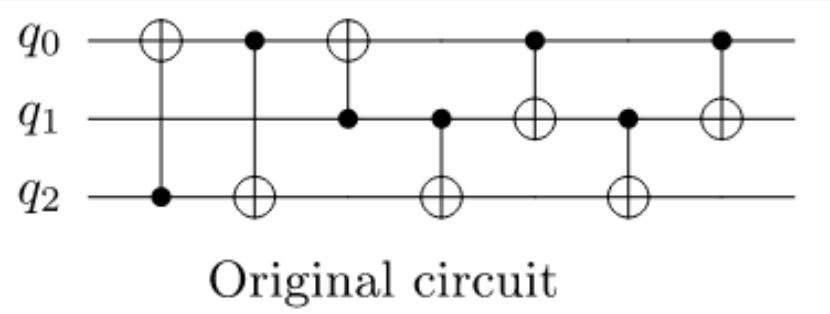
Contribute to the growth of Flix:

- Code formatter
- Program optimizations
- Faster type inference
- Extend the standard library
- ...

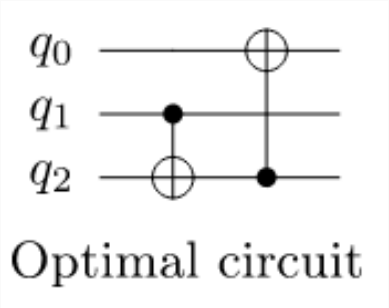
Quantum Circuit Synthesis



Quantum Circuit Optimization

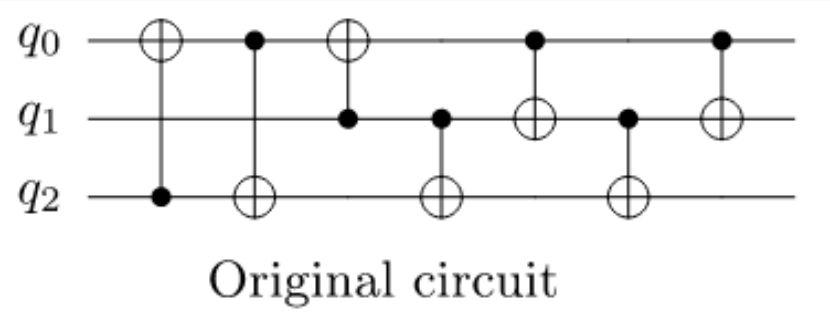


Original circuit

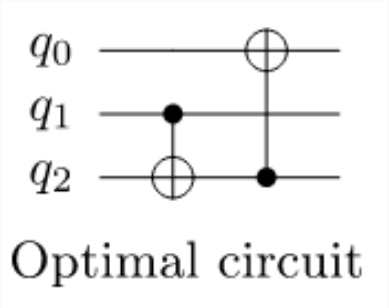


Optimal circuit

Quantum Circuit Optimization



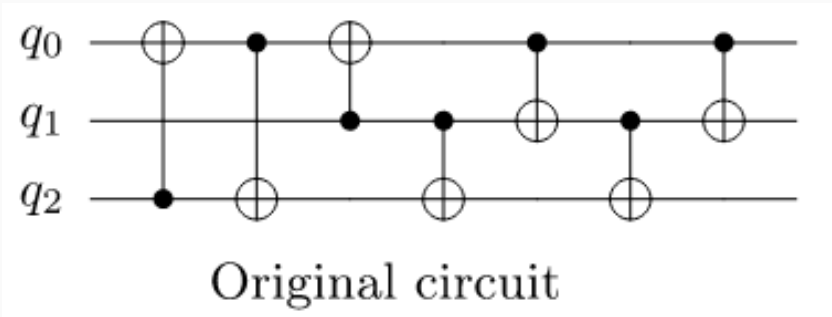
Original circuit



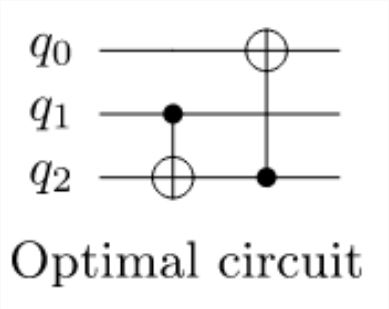
Optimal circuit

- How can we optimize circuits efficiently?
- What quantum gate sets can we target?
- What metrics can we optimize for? E.g.
  - Circuit Size
  - Circuit Depth
  - Gate count
  - Noise

Quantum Circuit Optimization

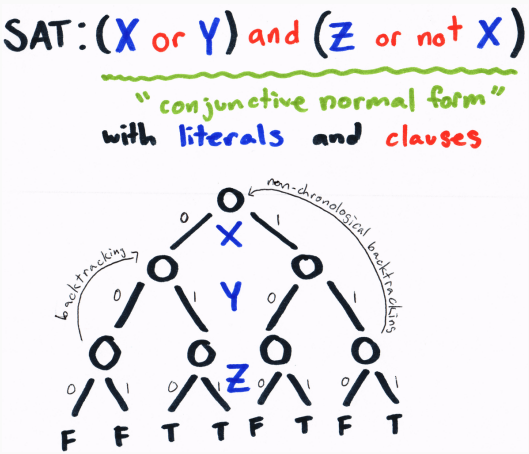


Original circuit



Optimal circuit

- How can we optimize circuits efficiently?
- What quantum gate sets can we target?
- What metrics can we optimize for? E.g.
  - Circuit Size
  - Circuit Depth
  - Gate count
  - Noise



QSynth!

Safe Compilation to WebAssembly and LLVM

Safe Compilation to WebAssembly and LLVM


Safe Compilation to WebAssembly and LLVM


- How do I compile between mainstream languages?
- How do I prove that my compilation is correct?
  - Property based testing
  - Mechanization
  - ...

Dependently Typed Programming

"Well-typed programs cannot go wrong." – Robin Milner

Well...

```
var x:Int;
...
//what if x is 0?
println(42/x);
```

```
def get(a:Array[Int], i:Int):Int={
   //What if i>=a.len?
 return a(i);
}
```

Dependently Typed Programming

"Well-typed programs cannot go wrong." – Robin Milner

Well...

```
var x:Int;
...
//what if x is 0?
println(42/x);
```

```
def get(a:Array[Int], i:Int):Int={
   //What if i>=a.len?
 return a(i);
}
```

Q: How can we prevent such errors at compile time?

Dependently Typed Programming

A: Dependent Types!

```
var x:Int[v|v!=0];
...
//x != 0
println(42/x);
```

```
def get(a:Array[Int], i:Int[v|v<a.len]):Int={
    // i<a.len
 return a(i);
}
```

Verified Functional Programming in Agda

- A dependently-typed programming language
- A proof assistant

Verified Functional Programming in Agda

- A dependently-typed programming language
- A proof assistant

Curry–Howard correspondence!

Propositions $\iff$ Types

Proofs $\iff$ Programs

Verified Functional Programming in Agda

- A dependently-typed programming language
- A proof assistant

Curry–Howard correspondence!

Propositions $\iff$ Types

Proofs $\iff$ Programs

- What are the foundations of dependently-typed programming?
- How do we express program correctness as dependent types?
- How can we verify simple algorithms using Agda?

What next?

- Meet us after this lecture
- Check our project descriptions on brightspace
- Email us, drop by our offices

What next?

- Meet us after this lecture
- Check our project descriptions on brightspace
- Email us, drop by our offices

Also reach out if

- You like a project but are in doubt
- You are passionate about an idea/project not listed