安全なコード生成を提供するプログラミング言語の理論について

On Foundations of Safe Code Genration

Yuito Murase
Graduate School of Informatics
Kyoto University

I work on *theoritical perspectives of code generation*.
My ambition is to establish design principles of
programming languages that suit for development of platforms.

1. Extensible Programming Languages

Motivation:

Domain-Specific Lang.

It would ideal if we can develop **a tailor-made programming language** to describe domain logic for each platform.

Big Data Platform


```
let num_workers(age_max) =
  select count()
  from residence_info_db
  where has_job = true
        and age <= age_max
```

PL with built-in query lang

IoT Platform

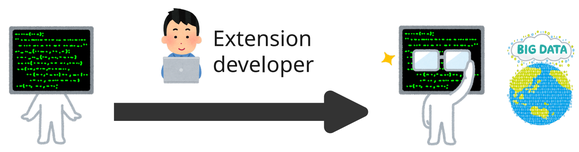
```
@sensor { send ch getTemp }
@controller {
  receive ch -> x
  store(db, x)
}
```

PL to describe business logic across multiple devices

Approach:

Syntactic Extensions

Our final goal is to design a extensible language that users can provide **domain-specific syntactic extensions for each platform** (a.k.a, macros)



Core Lang → **Query Lang Ext.**

```
let num_workers(age_max) =
  buildQuery(
    "SELECT count() ...",
    residence_tbl,
    age_max)
  |> submit(DB_CONNECTION)
  |> parseInt()
```

Compile

```
let num_workers(age_max) =
  select count()
  from residence_tbl
  where has_job = true
        and age <= age_max
```

Issue to resolve:

Static safety of exts.

We want to statically detect bugs in progams, but it is not straightforward in those programs written in an extended langauge.

```
let num_workers(age_max:int):int =
  select count()
  from residence_tbl
  where has_job = age_max
        Error: expected boolean,
        but got integer
```

Based upon

2. Type-Safe Code Generation

Background:

Multi-Stage Programming

... provides **langauge-level support for generating code**, which makes development of syntactic extensions much more efficient

Code Generator

```
let x: int code = `{ 10 } in
let y: int code = `{ 20 } in
`{ print (~x + ~y) }
```

Generated Program

```
print (10 + 20)
```

Type system can **find errors of generated code without generating it** (which should be applicable to the goal of 1)

```
let x: int code = `{ "hello" } in
let y: int code = `{ 20 } in
`{ print (~x + ~y) }
```

```
Error: expected integer,
but got string
```

Ongoing Project (joint work with Atushi Igarashi)**:**

Type System for Flexible Code Gen.

We work on a novel type system λ∀γ for code generation that ensure safety for code generation with more flexible operations.

```
let x:int@g = 10 in
let y:int g code = `{@g x + 1 } in
run y
```

scope information captures which variables can be used

code type captures scope information

First-place award winner @APLAS 2024 Student research competition

Based upon

3. Logical Foundation of Code Generation

Background:

Correspondance between Modal Logic and Code Generation

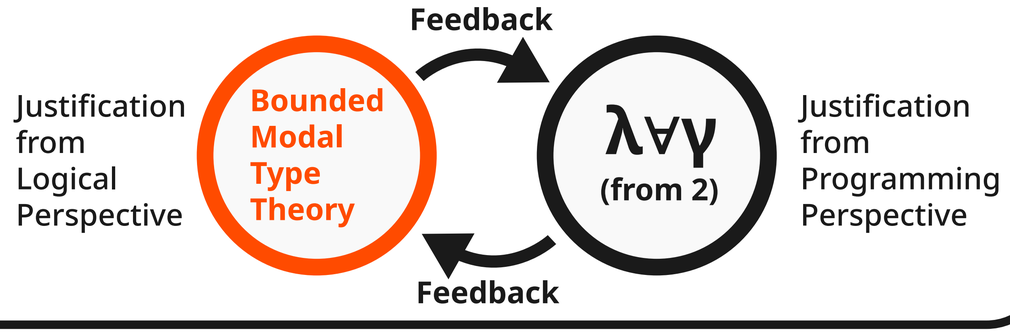
It is widely known that there is a correspondance between logic and programming languages, called *the Curry-Howard correspondance*.
And, **code generation is considered to correspond to modal logic**.

Logic	Programming Language
proofs	programs
propositions (subject of proofs)	types
implication (A → B)	function type (A → B)
modality (□A, necessarily A)	**code type (A code)**

Ongoing Project (joint work with Akinori Maniwa)**:**

Novel Modal Logic for Safe yet Flexible Code Genneration

We work on novel extension of modal logic, which we call ***Bounded Modal Type Theory***. We utilize BMTT to establish a design printicple of λ∀γ in 2.



Bottom-up Design