Goals of the Luau Type System

Lily Brown Andy Friesen Alan Jeffrey

ROBLOX

Human Aspects of Types and Reasoning Assistants 2021

Creator Goals

Roblox

A platform for creating shared immersive 3D experiences:

- **Many**: 20 million experiences, 8 million creators.
- **At scale**: e.g. *Adopt Me!* has 10 billion plays.
- **Learners**: e.g. 200+ kids' coding camps in 65+ countries.
- **Professional**: 345k creators monetizing experiences.

A very heterogeneous community.

Roblox developer community

All developers are important:

- **Learners**: energetic creative community.
- **Professionals**: high-quality experiences.
- **Everyone inbetween**: some learners become professionals!

Satisfying everyone is sometimes challenging.

Roblox Studio

Demo time!

Learners have immediate goals

E.g. “when a player steps on the button, advance the slide”.

- **3D scene editor** meets most goals, e.g. model parts.
- **Programming** is needed for reacting to events, e.g. collisions.
- **Onboarding** is very different from “let’s learn to program”.
- **Google Stack Overflow** is a common workflow.
- **Type-driven tools** are useful, e.g. autocomplete or API help.
- **Type errors** may be useful (e.g. catching typos) but some are not.

Type systems should help or get out of the way.

Professionals have long-term goals

E.g. “decrease user churn” or “improve frame rate”.

- **Code planning**: programs are incomplete.
- **Code refactoring**: programs change.
- **Defect detection**: programs have bugs.

Type-driven development is a useful technique!

Luau Type System

Infallible types

Goal: *support type-driven tools (e.g. autocomplete) for all programs.*

- **Traditional typing judgment** says nothing about ill-typed terms.
- **Infallible judgment**: every term gets a type.
- **Flag type errors**: elaboration introduces *flagged* subterms.

Related work:

- Type error reporting, program repair.
- Typed holes (e.g. in Hazel).

Strict types

Goal: *no false negatives*.

- **Strict mode** enabled by developers who want defect detection.
- **Business as usual** soundness via progress + preservation.
- **Gradual types** for programs with flagged type errors.

Related work:

- Lots and lots for type safety.
- Gradual typing, blame analysis, migratory types…

Nonstrict types

Goal: *no false positives*.

- **Nonstrict mode** enabled by developers who want type-drive tools.
- **Victory condition** does not have an obvious definition!
- **A shot at it**: a program is *incorrectly flagged* if it contains a flagged value (i.e. a flagged program has successfully terminated).
- **Progress + correct flagging** is what we want???

Related work:

- Success types (e.g. Erlang Dialyzer).
- Incorrectness Logic.

Mixing types

Goal: *support mixed strict/nonstrict development*.

- **Per-module** strict/nonstrict mode.
- **Combined** progress + preservation with progress + correct flagging?

Related work:

- Some on mixed languages, but with shared safety properties.

Type inference

Goal: *provide benefits of type-directed tools to everyone*.

- **Infer types** for all variables. Resist the urge to give up and ascribe a top type when an error is encountered.
- **System F** is in Luau, so everything is undecidable. Yay heuristics!
- **Different modes** currently infer different types. Boo!

Related work:

- Lots, though not on mixed modes.

Thank you!
Roblox is hiring!