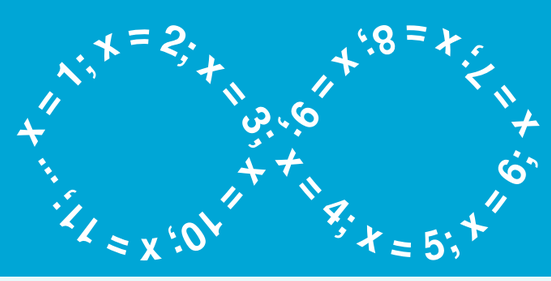
Stuck in a (While) Loop

Assessing Coinduction in Agda Using Cyclic Program Traces



TU Delft

1. Background

- **Program Trace**: describes progression of states a program passes through

$$x = 0 \;\Rightarrow\; x = 1 \;\Rightarrow\; x = 2 \;\Rightarrow\; \ldots$$

- Cycles (e.g. infinite loops) → infinite traces
- Applications of non-terminating programs — operating systems, industrial control
- Verifying program correctness → proof assistants
- Proof assistants (e.g. **Agda**) are often *total*, guaranteeing programs terminate
- Totality + reasoning about infinite structures? → **coinduction**

Three methods of coinduction in Agda, each guaranteeing **productivity**:

1. **Musical** uses musical symbols to represent "delay" (♯) and "force" (♭) operations.
2. **Guarded** relies on coinductive records and copattern matching.
3. **Sized** introduces "sizes" into types to help guide the termination checker.

Research Question

What are the different ways to model (potentially infinite) program traces which are suitable for use in the Agda proof assistant?

- How do these approaches compare in their abilities and limitations?
- What improvements can be made to coinduction support in Agda?

2. Encodings

- While: simple imperative language (while loops, variable assignment, conditional statements)
- Traces and relational semantics implemented for While using each of Agda's coinduction techniques
- Trace: non-empty colist of states
- Semantics: connects traces to programs
- Coinductive records: special care needs to be taken to encode "choice" between terminating and continuing → more complex definition

```
data Trace₁ : Set where
  tnil  : State → Trace₁
  tcons : State → ∞ Trace₁ → Trace₁
```

```
data _≈_ : Rel Trace₁ Level.zero where
  tnil  : ∀ {st} → (tnil st) ≈ (tnil st)
  tcons : ∀ {st tr₁ tr₂}
    → ∞ (♭ tr₁ ≈ ♭ tr₂)
    → (tcons st tr₁) ≈ (tcons st tr₂)
```

```
data rTrace₂ : Set where
  tnil  : State → rTrace₂
  tcons : State → Trace₂ → rTrace₂

record Trace₂ : Set where
  coinductive
  constructor mkTr
  field
    out : rTrace₂
```

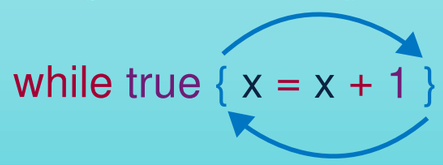
```
data _r≈_ : Rel rTrace₂ Level.zero where
  tnil  : ∀ {st}
    → (tnil st) r≈ (tnil st)
  tcons : ∀ {st tr₁ tr₂} → tr₁ ≈ tr₂
    → (tcons st tr₁) r≈ (tcons st tr₂)

record _≈_ (tr₁ tr₂ : Trace₂) : Set where
  coinductive
  constructor mkBisim
  field
    p : (out tr₁) r≈ (out tr₂)
```

```
data exec : Stmt → State → Traceₙ → Set
data execseq : Stmt → Traceₙ → Traceₙ → Set
```

```
data Trace₄ (i : Size) : Set where
  tnil  : State → Trace₄ i
  tcons : State
    → Thunk Trace₄ i
    → Trace₄ i

data Bisim (i : Size) : Rel (Trace₄ ∞) Level.zero where
  tnil  : ∀ {st} → Bisim i (tnil st) (tnil st)
  tcons : ∀ {st tr₁ tr₂}
    → Thunk (λ s → Bisim s (force tr₁) (force tr₂)) i
    → Bisim i (tcons st tr₁) (tcons st tr₂)
```



5. Future Research

- Verification of identified limitations
- Documentation and error messages → reduce learning curve
- Colist representations in Guarded coinduction
- More complex language → verification of practical programs

3. Experiments

Three types of proof, which combine to guarantee behavior of a program:

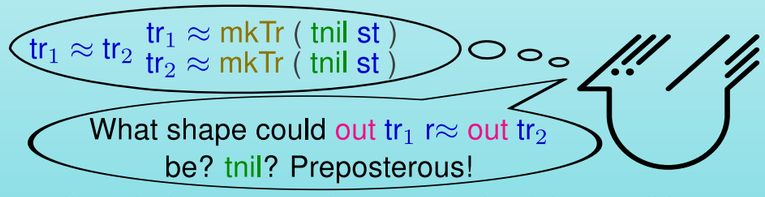
- **Trace proofs**: Proofs that a variable follows an increasing progression in an infinite trace
- **Program proofs**: Proofs that programs satisfy traces, including for infinite traces
- **Language proof**: Proof of determinism of the language — for any state and statement, two traces arising from execution of the statement in the state must be bisimilar

Type	Encoding	Proof		
		Trace	Program	Language
Musical	Trace_1	✓	✓	✓
Guarded	Trace_2	✓	✓	✗
	Trace_3	✗	✗	✗
Sized	Trace_4	✓	✓	✓

Success of experiments for each encoding

4. Agda Limitations

- Lack of documentation → steep learning curve
- Error messages obscure the root cause of problems
- Unification issues under function application ↓



- Guardedness too strict as a productivity condition → battles with the termination checker
- Coinductive records model concept of "choice" poorly

Claire Stokka — C.C.Stokka@student.tudelft.nl

CSE3000 — Research Project

Supervisors: Bohdan Liesnikov, Jesper Cockx