The μ-calculus, type-theoretically

Sean Watters

University of Strathclyde

30/08/2024

The Propositional Modal μ-calculus

Given a set of variable names P with $p, x \in P$:

$$L_\mu \ := \ \top \mid \bot \mid p \mid \neg p \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \Box\varphi \mid \Diamond\varphi \mid \mu x.\varphi \mid \nu x.\varphi$$

Essentially, this is:

- Propositional logic,
- Plus the modal operators $\Box$ and $\Diamond$,
- Plus the least and greatest fixpoint operators μ and ν.

Fixpoint Unfolding

For any fixpoint formula of the form $\mu x.\varphi$, we have:

$$\mu x.\varphi \equiv \varphi[x \,/\, \mu x.\varphi]$$

(And dually for ν.)

eg:

$$\begin{aligned} & \nu x.p \wedge \Box x \\ \equiv\ & p \wedge \Box(\nu x.p \wedge \Box x) \\ \equiv\ & p \wedge \Box(p \wedge \Box(\nu x.p \wedge \Box x)) \\ \equiv\ & \dots \end{aligned}$$

Surprising(?) fact: For any formula φ, take the set of formulas closed under direct subformulas of propositional and modal formulas, and unfoldings of fixpoints. This set is always finite.

The Fischer-Ladner Closure

Definition (The Closure of ϕ)

The least set of formulas which contains ϕ and is closed under taking direct subformulas of propositional and modal subformulas, and unfolding fixpoint subformulas.

The closure is highly important in the study of the μ-calculus, but...

It is not invarient under α-equivalence!

For example, we have:

$$\begin{aligned} & \mu x.(\Diamond x \wedge \Box(\nu y.\mu z.(\Diamond z \wedge \Box y))) \\ \equiv\ & \mu x.(\Diamond x \wedge \Box(\nu y.\mu x.(\Diamond x \wedge \Box y))) \end{aligned}$$

But their closures are different!

De Bruijn to the Rescue?

```
data WST (At : Set) (n : ℕ) : Set where
  tt ff   : WST At n
  at ¬at  : At → WST At n
  and or  : (φ ψ : WST At n) → WST At n
  box dia : (φ : WST At n) → WST At n
  mu nu   : (φ : WST At (suc n)) → WST At n
  var     : Fin n → WST At n
```

```
data Scope (At : Set) : ℕ → Set where
  [] : Scope At zero
  _-,_ : ∀ {n} (Γ₀ : Scope At n) {φ : WST At n}
       → (Γ₀ : IsFP φ) → Scope At (suc n)
```

~~Well~~ *Sublimely*-Scoped Formulas

```
mutual
  data SST (At : Set) {n : ℕ} (Γ : Scope At n) : Set where
    -- other constructors here...
    var : (x : Fin n) → SST At Γ
    mu : {ψ : WST At (suc n)}
         → (φ : SST At (Γ -, mu ψ))
         → ψ ≈ φ
         → SST At Γ

  data _≈_ {At : Set} {n : ℕ} {Γ : Scope At n}
    : WST At n → SST At Γ → Set where
    -- other constructors here...
    var : (x : Fin n) → (var x) ≈ (var x)
    mu : {φ : WST At (suc n)}
         → {φ' : SST At (Γ -, mu φ)}
         → (p : φ ≈ φ')
         → mu φ ≈ mu φ' p
```