Correct-by-Construction Type-Checking for Algebraic Data Types

Miloš Ristić: m.ristic-1@student.tudelft.nl

1. Motivation

- Static type systems prevent wide range of bugs.
- Type-checkers should never accept faulty programs.
- Avoiding bugs through intrinsic verification.
  - Intrinsic verification: inherent verification by specifying properties at the type-level of a program [1].
- Uncertain whether the benefits of correct-by-construction type-checking outweigh the challenges for different language features, e.g. algebraic data types (ADTs).
- ADTs expand expressive power by adding support for common data types like lists, trees, etc.
- **How can correct-by-construction programming be used to increase the trustworthiness of type-checkers for ADTs?**

```haskell
data List a = Nil | Cons a (List a)
```

Figure 1: ADT declaration in Haskell.

2. Agda

- Functional programming language and proof assistant.
- Allows encoding invariants and relations between values at type level using dependent types.

3. Toy Language

- Polymorphic and recursive ADTs.
- Extension of polymorphic lambda calculus (System F) [2] with Haskell-like ADTs and pattern matching.

```
Λ ƛ "x" : T "List" (TVar 0 :: []) ⇒
  ‘case ‘ "x" # {-} of[
    (‘ "Cons" # {-} : "v" :: "t" :: []
      → ‘ "Just" # {-} ∘ TVar 0 · ‘ "v" # {-}) ::
    (‘ "Nil" # {-} : []
      → ‘ "Nothing" # {-} ∘ TVar 0)
    :: []
  ]
```

Figure 2: Abstract syntax tree for an implementation of “head”.

4. Implementing the Type-Checker

- Typing relation where constructors are the typing rules.
  - *“Given context Γ and Δ, term e has type t”*

```
data _⨾_⊢_:_ (Γ : Context α) (Δ : TyContext n) : Term α → Type → Set where
  ⊢‘
    : (p : x ∈ α)
    ---------------------------
    → Γ ⨾ Δ ⊢ ‘ x # p : lookupVar Γ x p
  ⊢ƛ
    : Γ , x : t₁ ⨾ Δ ⊢ e : t₂
    → Δ ⊢ᵏ t₁
    ---------------------------
    → Γ ⨾ Δ ⊢ (ƛ x : t₁ ⇒ e) : t₁ ⇒ t₂
  ⊢·
    : Γ ⨾ Δ ⊢ e₁ : (t₁ ⇒ t₂)
    → Γ ⨾ Δ ⊢ e₂ : t₁
    ---------------------------
    → Γ ⨾ Δ ⊢ e₁ · e₂ : t₂
```

Figure 3: Subset of the typing rules as a relation in Agda.

- Type-checker constructs an instance of the relation which serves as proof the term is well-typed.
- Impossible to create an instance of a relation if the relation does not hold > Impossible to create proof for ill-typed term.
- Unsound type-checker will not compile.

```
inferTerm : ∀ {α : Scope} {n : ℕ} (Γ : Context α) (Δ : TyContext n)
  (u : Term α) ⟶ Evaluator (Σ[ t ∈ Type ] Γ ⨾ Δ ⊢ u : t)
checkTerm : ∀ {α : Scope} {n : ℕ} (Γ : Context α) (Δ : TyContext n)
  (u : Term α) (ty : Type) ⟶ Evaluator (Γ ⨾ Δ ⊢ u : ty)
```

Figure 4: Signatures for type-checker. Both return instance of the typing relation or an error mesage, *inferTerm* returns it paired with the inferred type.

- Type inference allows fewer explicit type annotations while maintaining type safety.

5. Evaluating the Type-Checker

- Type-checker is sound by construction, i.e. can not accept ill-typed terms.
- Type-checker may reject correct programs, i.e. incomplete.
  - Should return proof that term is ill-typed instead of only an error message for completeness.
- Correctly handled all 19 test cases, 6 well-typed, 13 ill-typed.

6. Discussing the CbC Approach

- Advantages
  - Guarantee of termination and no run-time errors in Agda.
  - Soundness by construction.
- Challenges
  - Added complexity.
  - Extrinsic verification may still be required.
  - Termination check can reject terminating programs.
  - Agda standard library [4] can be hard to navigate.

7. Future Work

- Research correct-by-construction type-checking for further language features.
  - Generalized algebraic data types.
- Research integration with correct-by-construction compilers and interpreters.

8. References

[1] J. Cockx. (2019). Correct-by-construction programming in Agda: indexed datatypes and dependent pattern matching [Online]. Available: https://jespercockx.github.io/ohrid19-agda/slides/slides2.html#/title-slide.

[2] J. Girard, “The system F of variable types, fifteen years later,” *Theor. Comput. Sci.*, vol. 45, no. 2, pp. 159–192, Sep. 1986.

[4] Agda standard library. “Documentation for the Agda standard library.” agda.github.io. Accessed: Jun. 5, 2024. [Online] Available: https://agda.github.io/agda-stdlib/

Supervisor: **Sára Juhošová** Responsible Professor: **Jesper Cockx**