GADTs for Reconstruction of Invariants and Postconditions

Doctoral Dissertation Defense

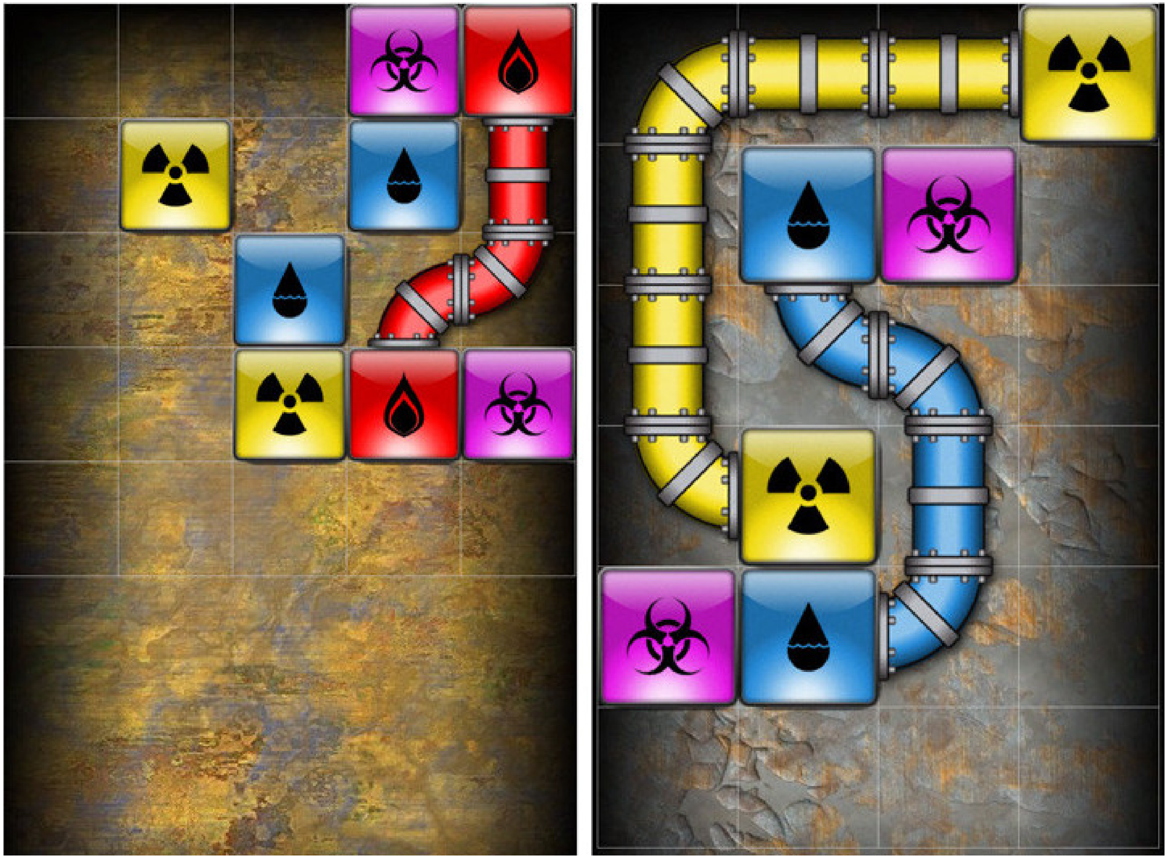
Łukasz Stafiniak

Advisor: Leszek Pacholski

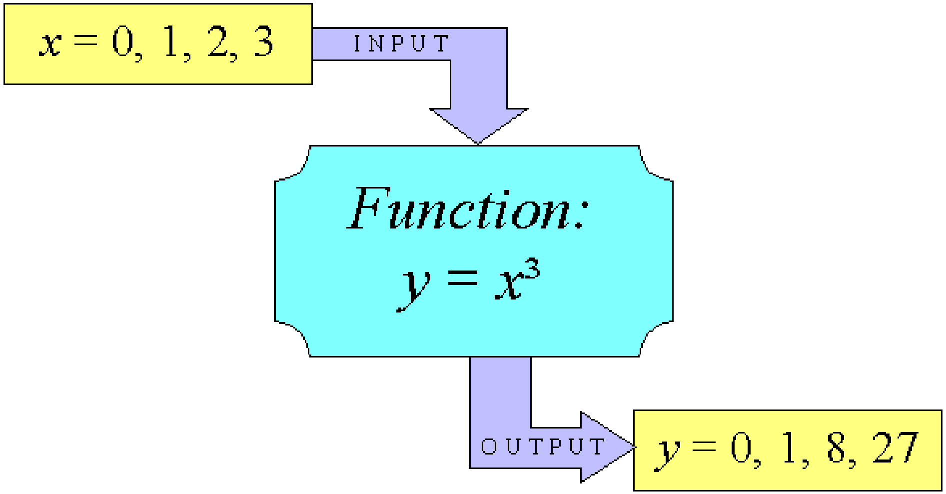
October 2, 2015

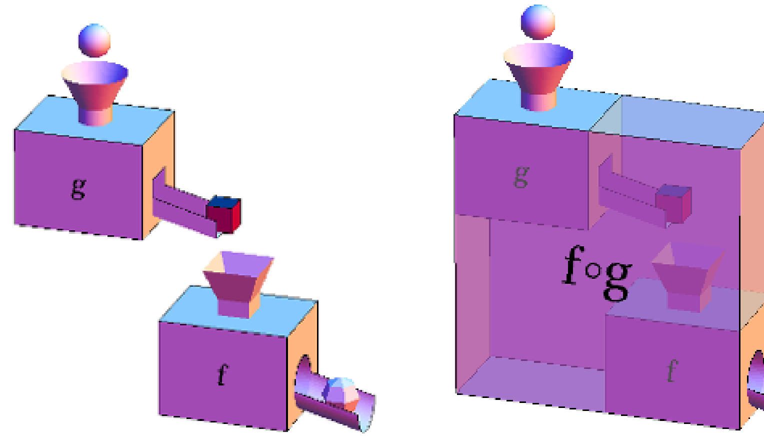
Types in Programming are Helpful

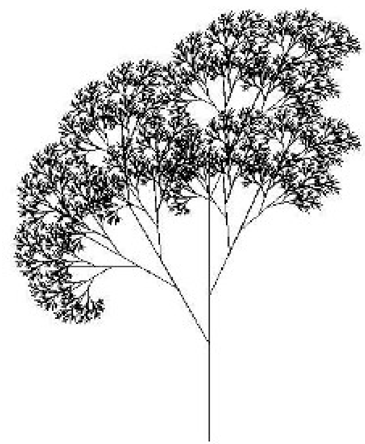
- *Type systems* $\approx$ detailed grammars
- Static checking $\Rightarrow$ no more program crashes!
- Help structure programs.
- Help evolve programs.



Functional Programming



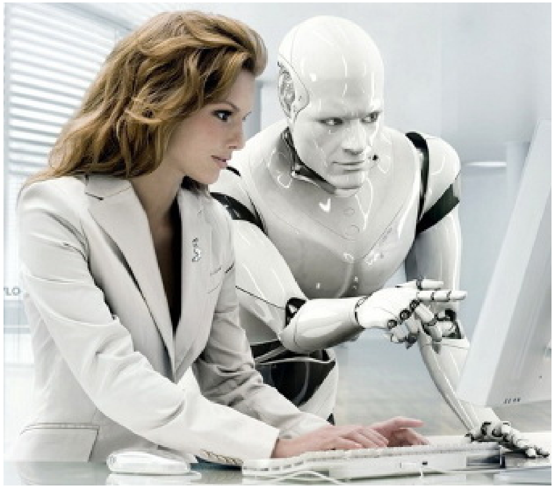




Thesis: Why

- **Fully automated specification generation**:
  - Overcome the quick-and-dirty mindset.
  - Fast evolution of specifications.
  - Comparative advantage wrt. other projects.

Core ML



InvarGenT

Coq

Generalized Algebraic Data Types

Example: encoding computation

```
datacons Lit : Int ⟶ Term Int
datacons Pair : ∀a, b. Term a * Term b
                          ⟶ Term (a, b)
```

specific types
for specific cases

```
let rec eval = function
    | Lit i -> i                    ∀a. Term a → a
     Term Int → Int
    | Pair (x, y) -> eval x, eval y
     Term (a, b) → (a, b)
```

computing / reasoning by cases

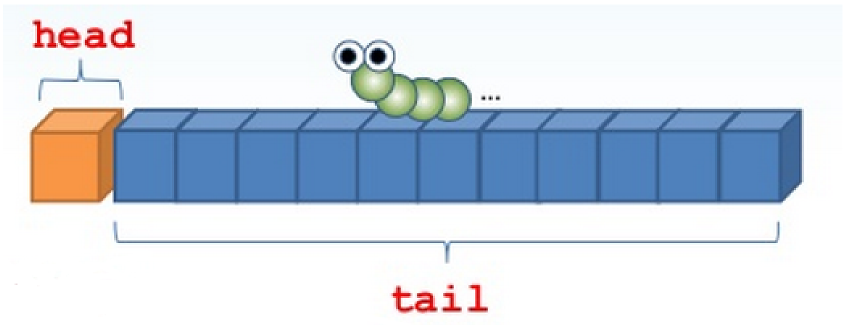
Examples of Numerical Properties

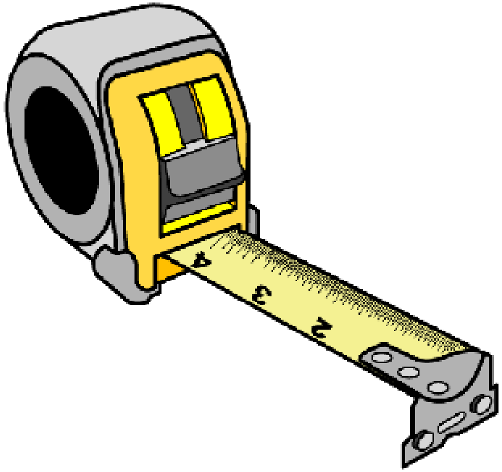
- Binary representation of natural numbers:

```
datacons POne : ∀n [0≤n].
  Binary n ⟶ Binary(2 n + 1)
```

- Lists with length:

```
datacons LCons : ∀n, a [0≤n].
  a * List(a, n) ⟶ List(a, n+1)
```

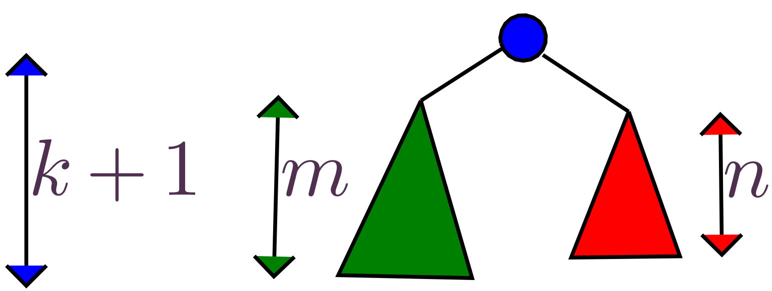


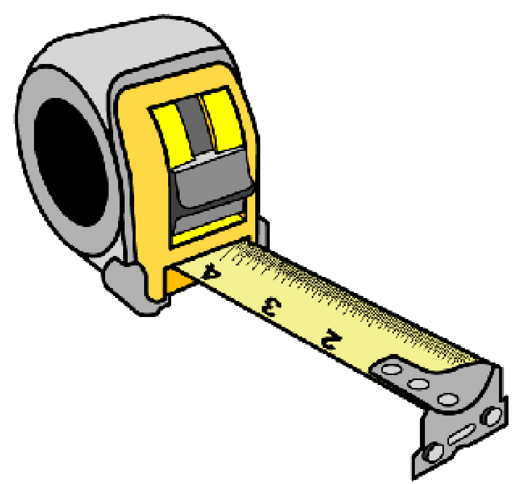


Examples of Numerical Properties

- AVL trees with imbalance of at most 2:

```
datacons Node :
  ∀a,k,m,n [k=max(m,n) ∧ 0≤m ∧ 0≤n ∧
              n≤m+2 ∧ m≤n+2].
  Avl (a, m) * a * Avl (a, n) * Num (k+1)
                           ⟶ Avl (a, k+1)
```





Existential Types

Without existential types, every case has the same type (as function of input type).
''We only accept standard size boxes!''

With inferred existential types, a range of parameters is automatically generated.
''Let me package this for you!''

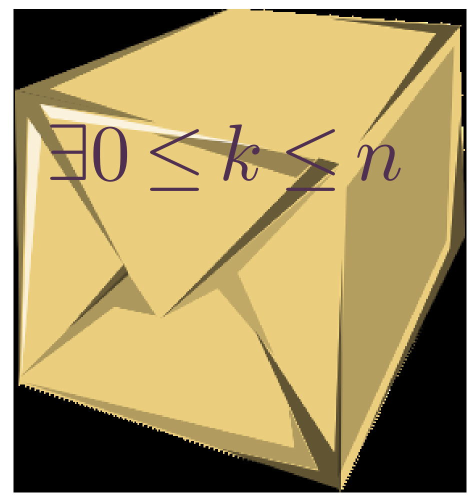
Invariants, Preconditions, Postconditions

Invariants: types and properties of arguments (i.e. *preconditions*) and of results required to call a function.

`head : ∀n, a[1 ≤ n].`
`        List (a, n) → a`

Postconditions: existential types for results of functions, capturing properties of results.

`filter: ∀n, a.(a → Bool)`
`→ List (a, n)→ ∃k[k ≤ n`
`   ∧ 0 ≤ k].List (a, k)`

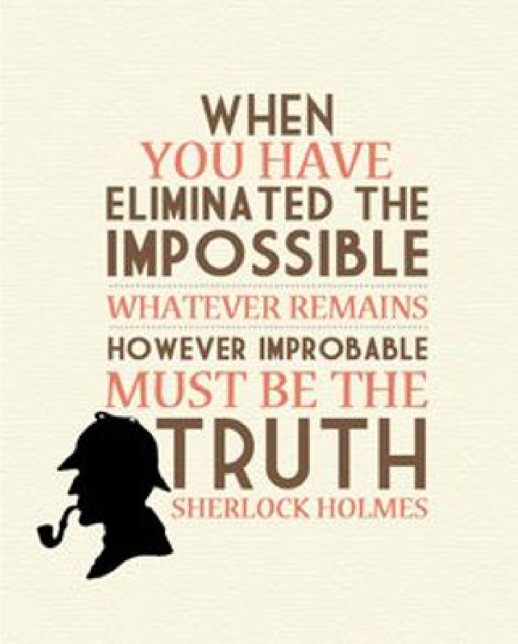


Thesis: What

Fully automated reconstruction of invariants and postconditions for recursive functions can be formulated and achieved as type inference for type systems with GADTs.

Abduction

- Abduction = reasoning to the best explanation.
- Constraint Abduction: given conjunctions of atoms D, C solve $D \Rightarrow C$ by finding good A such that $\mathcal{M} \vDash A \wedge D \Rightarrow C$.
- *Joint Constraint Abduction*: multiple $D_i \Rightarrow C_i$, single A.

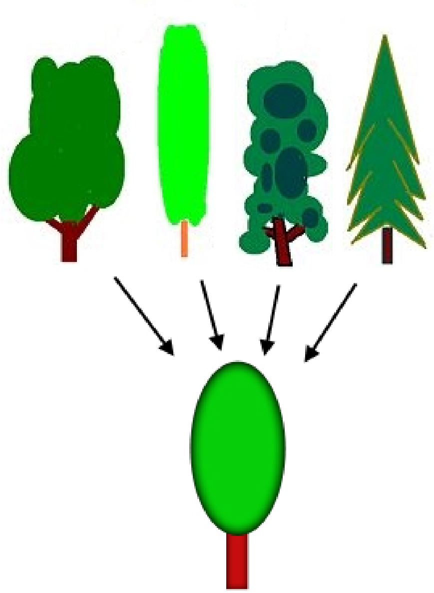


Constraint Abduction Ideas

- For type shapes, guess equality between compatible variables.
- For linear inequalities, take each $d \in D, c \in C$, and solve $d \Rightarrow c$ exactly. In particular:
  - $d \Leftrightarrow \alpha \leqslant d_\alpha$ and $c \Leftrightarrow \alpha \leqslant c_\alpha$: the abduction answers are c and $d_\alpha \leqslant c_\alpha$,
  - $d \Leftrightarrow d_\alpha \leqslant \alpha$ and $c \Leftrightarrow c_\alpha \leqslant \alpha$: the abduction answers are c and $c_\alpha \leqslant d_\alpha$.

Generalization

- *Constraint Generalization*: given conjunctions of atoms D_i, solve $D_1 \vee \ldots \vee D_n$ by finding a conjunction G such that $\mathcal{M} \models D_i \Rightarrow G$ for all i.
- = *anti-unification* for type shapes.
- = *convex hull* for numerical properties.



Finding Invariants and Postconditions

1. Start with trivial preconditions (no properties).
2. Use constraint generalization to find **strongest postconditions** – on initial iterations, from base cases only.
3. Use joint constraint abduction to update **maximally weak preconditions**.
4. Go to (2) if either invariants or postconditions change.

Thesis: How

Fully automated reconstruction of invariants and postconditions for recursive functions can be achieved by joint constraint abduction and constraint generalization.

Example: Binary Addition

```
let rec plus =
  function CZero ->
    (function
      | Zero ->
        (function Zero -> Zero
          | PZero _ as b -> b
          | POne _ as b -> b)
      | PZero a1 as a ->
        (function Zero -> a
          | PZero b1 -> PZero (plus CZero a1 b1)
          | POne b1 -> POne (plus CZero a1 b1))
...
```

$\texttt{plus} : \forall \texttt{i}, \texttt{k}, \texttt{n}.\texttt{Carry i} \to \texttt{Binary k} \to \texttt{Binary n} \to \texttt{Binary (n + k + i)}$

Example: AVL Trees merge

```
let merge = efunction
   | Empty, Empty -> Empty
   | Empty, (Node (_,_,_,_) as t) -> t
   | (Node (_,_,_,_) as t), Empty -> t
   | (Node(_,_,_,_) as t1),(Node(_,_,_,_) as t2) ->
     let x = min_binding t2 in
     let t2' = remove_min_binding t2 in
     eif height t1 <= height t2' + 2
     then create t1 x t2'
     else rotr t1 x t2'
```

$$\texttt{merge} : \forall k, n, a[k \leq n + 2 \wedge n \leq k + 2].$$
$$(\texttt{Avl}\ (a, n), \texttt{Avl}\ (a, k)) \to$$
$$\exists i[n \leq i \wedge k \leq i \wedge i \leq n + k \wedge$$
$$i \leq \max\ (k + 1, n + 1)].\texttt{Avl}\ (a, i)$$

Contributions

- Formalized finding invariants and postconditions as "single-stage" constraint-based type inference.
  - Alternative to *refinement types*.
- Weakly multi-sorted joint constraint abduction and constraint generalization algorithms.
- Best GADTs inference (vs. Chuan-kai Lin).
- Competitive, faster reconstruction.
  - Complementary to learning approaches coming from Suresh Jagannathan's group.
- `github.com/lukstafi/invargent`