A Reconstruction of a Types-and-Effects Analysis by Abstract Interpretation

Letterio Galletta

Dipartimento di Informatica - Università di Pisa

19/09/2012 - ITCTCS 2012

Outline

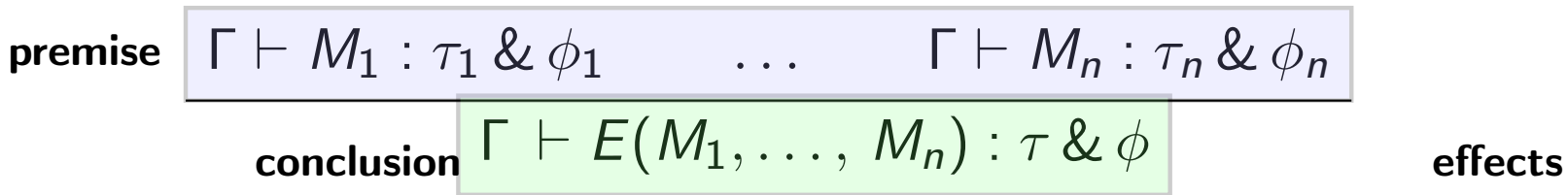
Type and Effect Systems

A powerful type systems allowing one to reason about program's execution

$$\frac{\Gamma \vdash M_1 : \tau_1 \,\&\, \phi_1 \qquad \ldots \qquad \Gamma \vdash M_n : \tau_n \,\&\, \phi_n}{\Gamma \vdash E(M_1, \ldots, M_n) : \tau \,\&\, \phi}$$

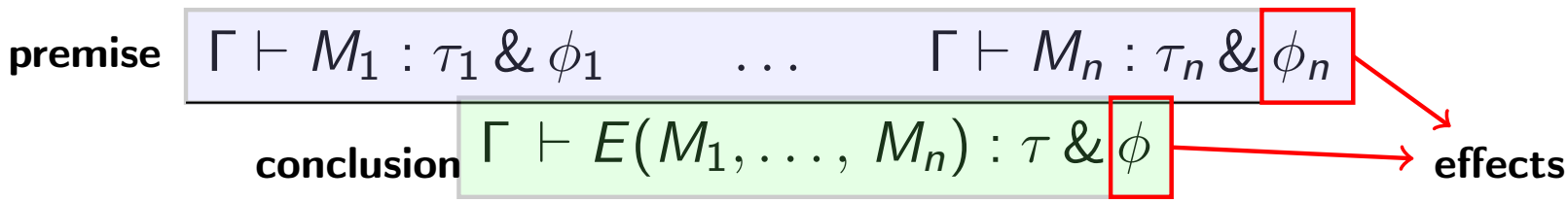
Type and Effect Systems

A powerful type systems allowing one to reason about program's execution



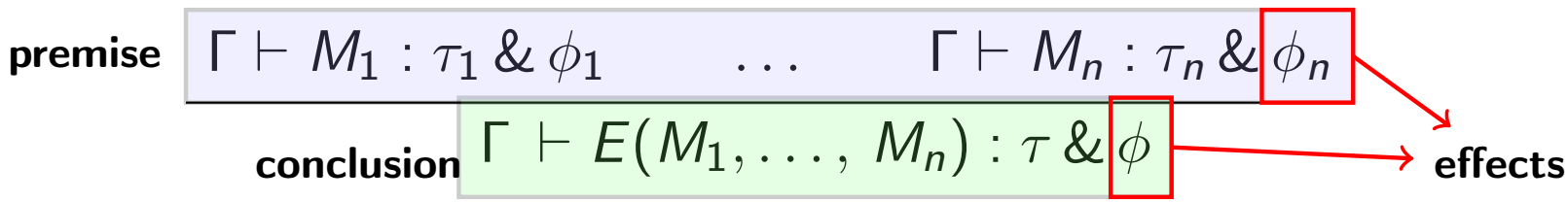
Type and Effect Systems

A powerful type systems allowing one to reason about program's execution



Type and Effect Systems

A powerful type systems allowing one to reason about program's execution



If the premise holds then $E(M_1, \ldots, M_n)$

- has type τ
- enjoys the semantic property ϕ

Application: communication, security properties, resource usage, secure information flow, atomicity, etc.

Type and Effect Systems

Methodology

The definition of an analysis requires

1. Typing rules
2. State and prove the soundness theorem
3. Inference algorithm
4. Prove that the algorithm is correct (soundness/completeness)

Drawbacks

1. A new analysis requires performing all previous steps
2. No general theory exists to systematically build-up an analysis from existent ones
   - g.e. how can we systematically compose analyses for resource usage [Bartoletti and et.] and secure flow [Volpano and et.]?

Abstract Interpretation

A general theory introduced by Patrick and Radhia Cousot for approximating the semantics of dynamic systems

- specify static analyses
- prove their correctness
- construct systematically hierarchies of related semantics
- derive correct analysis algorithms

Key Ideas

- Every property of a program can be observed in its semantics and computed as an approximation
- The analysis can be systematically derived by throwing away superfluous information from the semantics

Abstract Interpretation Advantages

soundness if the abstract domain is a sound approximation of the concrete one, the analysis is sound by construction

analysis algorithm if the abstract semantics is decidable, it is an sound analysis algorithm

Domain construction

- We can systematically specify a novel analysis by transforming/composing existent abstract domains
- A large number of domain operators to define novel domains (see the literature)

Abstract Interpretation and Type and Effect

Type Systems

Relevant literature and results (Monsuez, Cousot, Gori & Levi)

What about type and effect systems?

Our long term goal

Definition of a abstract interpretation-based framework for type and effect providing "primitive"

- abstract domains
- domain operators

First Step

Reconstruction of well known analysis by abstract interpretation

- LINKS
- Call-Tracking Analysis

Results

The definition of a preliminary methodology and a preliminary abstract domain

- methodology => extension of the Cousot's one
- abstract domain => extension of the Gori & Levi's abstract domain (Hindley's monotypes, idempotent substitution)

A workbench: Call-Tracking Analysis

For each program phrase

- its type
- a over approximation of function applications occurring during the evaluation

Example

```
let f = fun[f_point] x -> x 2 in
  f (fun[y_point] y -> 1)
```

- the type is `integer`
- the effect is $\{\texttt{f\_point}, \texttt{y\_point}\}$

TinyML: Syntax

A core of ML with labelled function abstraction

$$\begin{aligned}\mathtt{E} ::= {} & \mathtt{n} \mid \mathtt{b} \mid \mathtt{x} \mid \lambda^{\mathtt{l}}\mathtt{x}.\,\mathtt{E} \mid \mathtt{E}_1\,\mathtt{E}_2 \mid \mu\mathtt{f}.\,\lambda^{\mathtt{l}}\mathtt{x}.\,\mathtt{E} \mid \mathtt{let\ x} = \mathtt{E}_1\ \mathtt{in}\ \mathtt{E}_2 \mid \\ & \mathtt{E}_1\ \mathtt{op_a}\ \mathtt{E}_2 \mid \mathtt{E}_1\ \mathtt{op_l}\ \mathtt{E}_2 \mid \mathtt{E}_1\ \mathtt{op_r}\ \mathtt{E}_2 \mid \mathtt{iszero(E)} \mid \mathtt{not(E)} \mid \\ & \mathtt{if}\ \mathtt{E}_1\ \mathtt{then}\ \mathtt{E}_2\ \mathtt{else}\ \mathtt{E}_3\end{aligned}$$

where

- $\mathtt{b} \in \{\mathtt{true}, \mathtt{false}\}$
- $\mathtt{op_a} \in \{+, \star, -\}$
- $\mathtt{op_l} \in \{\mathtt{and}, \mathtt{or}\}$
- $\mathtt{op_r} \in \{<, >\}$
- $\mathtt{l} \in \mathtt{Point}$ — function labels

Concrete Semantics

Following the Cousot's methodology the concrete semantics is a denotational semantics considering

- TinyML a untyped λ-calculus
- the labels of the applied function $\mathcal{P}(Point)$ (effects store)

Semantic values domain

$$Eval = (\mathbb{Z} + T + S + (\mathcal{P}(Point) \to Eval \to (Eval \times \mathcal{P}(Point))))_{\perp}$$

Concrete Semantics

The semantic function is

$$[\![-]\!] : \texttt{EXP} \to \mathit{Env} \to \mathcal{P}(\mathit{Point}) \to (\mathit{Eval} \times \mathcal{P}(\mathit{Point}))$$

Examples of semantic equation

Concrete Semantics

The semantic function is

$$[\![-]\!] : \texttt{EXP} \to \mathit{Env} \to \mathcal{P}(\mathit{Point}) \to (\mathit{Eval} \times \mathcal{P}(\mathit{Point}))$$

Examples of semantic equation

$$\begin{aligned}
[\![\texttt{E}_1\ \texttt{E}_2]\!]\rho\ ps = {} & \mathit{let}\ (v_1,\ ps_1)\ =\ [\![\texttt{E}_1]\!]\rho\ ps \\
& \mathit{let}^\star\ v'\ =\ v_1\ \mathit{in} \\
& \mathit{let}\ (v_2,\ ps_2)\ =\ [\![\texttt{E}_2]\!]\rho\ ps_1 \\
& \mathit{let}^\star\ v''\ =\ v_2\ \mathit{in} \\
& \mathit{case}\ v'\ \mathit{of} \\
& \quad \mathit{Fun}(f) \to f\ \lfloor\ v''\ \rfloor\ ps_2 \\
& \quad _ \to (\lfloor\ \mathit{WrongValue}()\ \rfloor,\ \emptyset)
\end{aligned}$$

Concrete Semantics

The semantic function is

$$[\![-]\!] : \texttt{EXP} \to \mathit{Env} \to \mathcal{P}(\mathit{Point}) \to (\mathit{Eval} \times \mathcal{P}(\mathit{Point}))$$

Examples of semantic equation

$$\begin{aligned}
[\![\texttt{if}\ \texttt{E}_1\ \texttt{then}\ \texttt{E}_2\ \texttt{else}\ \texttt{E}_3]\!]\rho\ ps = \mathit{let}\ (v_1,\ ps_1)\ =\ & [\![\texttt{E}_1]\!]\rho\ ps\ \mathit{in} \\
& \mathit{let}^{\star}\ v'\ =\ v_1\ \mathit{in} \\
& \quad \mathit{case}\ v'\ \mathit{of} \\
& \qquad \mathit{Bool}(b) \to \mathit{if}\ b\ \mathit{then}\ [\![\texttt{E}_2]\!]\rho\ ps_1\ \mathit{else}\ [\![\texttt{E}_3]\!]\rho\ ps_1 \\
& \qquad _ \to (\lfloor\ \mathit{WrongValue}()\ \rfloor, \emptyset)
\end{aligned}$$

Abstract domain

Values

- types have annotations

$$\texttt{integer} \overset{\{f,g\}}{\rightarrow} \texttt{integer}$$

- an extension of the Gori & Levi's abstract domain to deal with annotations

Our proposal

- annotation variables in types
- constraint to restrict annotation variables values

$$\texttt{integer} \overset{\gamma}{\rightarrow} \texttt{integer} \qquad \gamma \supseteq \{f, g\}$$

Abstract domain

Values

- types have annotations

$$\texttt{integer} \overset{\{f,g\}}{\rightarrow} \texttt{integer}$$

- an extension of the Gori & Levi's abstract domain to deal with annotations

The resulting domain

$$\mathit{TypeA} = \mathit{TypeS} \times \mathit{Constr}$$

- *TypeS* => a pair (Hindley's monotypes, idempotent substitution)
- *Constr* => set of pair (annotation variable, function label) representing constraints

Galois Connection

The relationship between the abstract domain and the concrete one is built-up in two steps

1. Definition of some representation functions
   - values (β_v)
   - environments (β_ρ)
   - others auxiliary (i.e. β_{vt}, β_{f_1})

2. By using the representation functions and some standard lemmas we can define

$$(CD, \alpha, \gamma, AD)$$

Abstract Semantics

The abstract semantic function is

$$[\![-]\!]^a : \texttt{EXP} \to \mathit{AEnv} \to \mathcal{P}(\mathit{Point}) \to (\mathit{TipoA} \times \mathcal{P}(\mathit{Point}))$$

Both semantics have been implemented (a unique semantic function parametrized by the primitive operation and semantic domain)

Example 1

Expression

```
let f = fun[f_point] x -> x 2 in
  f (fun[y_point] y -> 1)
```

Abstract semantics

```
(type - : Integer [(_annvar3_,f_point),
           (_annvar2_,y_point)] & {f_point, y_point})
```

Example 2

Expression

```
let a = fun[a_point] x -> true in
let b = fun[b_point] x -> false in
  (a 1) or (b 1)
```

Abstract semantics

```
(type - : Boolean [(_annvar2_,a_point),
                   (_annvar3_,b_point)] & {a_point, b_point})
```

Conclusions

First steps towards an abstract interpretation-based framework to express type and effect systems

Future work

- different style of concrete semantics (e.g. structural operational semantics)
- more interesting types (e.g. polymorphism, subtyping)
- different kind of effects (e.g. history expressions)
- more expressive constraints and hierarchy of constraints