Behavioural Type Systems

Algorithms Analysing Algorithms

Dr. Maxime Gamboni

Instituto de Telecomunicações, Instituto Superior Técnico, Portugal

December 5, 2011

Plan

- Behavioural
- Type Systems:
- Algorithms Analysing
- Algorithms

Behavioural Properties

"Behavioural Type Systems: Algorithms Analysing Algorithms"

```
public PType prod(PType that) throws IllegalArgumentExcep
  Map n = Tools.union(this.names,that.names);
  Map nli = new HashMap(), // new local inputs
    nlo = new HashMap(), // new local outputs
    nri = new HashMap(), // new remote inputs
    nro = new HashMap(); // you probably got the idea by 

  Mult al,ar,bl,br,m; // names as in Mult.radd
  for (Iterator i = n.keySet().iterator();i.hasNext();) {
    Var v = (Var)i.next();
    al = this.getMult(true, v,true);
    ar = this.getMult(false,v,true);
    bl = that.getMult(true, v,true);
```

Behavioural Properties

"Behavioural Type Systems: Algorithms Analysing Algorithms"

```
public PType prod(PType that) throws IllegalArgumentExcep
  Map n = Tools.union(this.names,that.names);
  Map nli = new HashMap(), // new local inputs
    nlo = new HashMap(), // new local outputs
    nri = new HashMap(), // new remote inputs
    nro = new HashMap(); // you probably got the idea by 

  Mult al,ar,bl,br,m; // names as in Mult.radd
  for (Iterator i = n.keySet().iterator();i.hasNext();) {
    Var v = (Var)i.next();
    al = this.getMult(true, v,true);
    ar = this.getMult(false,v,true);
    bl = that.getMult(true, v,true);
```

Behavioural Properties

"Behavioural Type Systems: Algorithms Analysing Algorithms"

```
public PType prod(PType that) throws IllegalArgumentExcep
  Map n = Tools.union(this.names,that.names);
  Map nli = new HashMap(), // new local inputs
    nlo = new HashMap(), // new local outputs
    nri = new HashMap(), // new remote inputs
    nro = new HashMap(); // you probably got the idea by

  Mult al,ar,bl,br,m; // names as in Mult.radd
  for (Iterator i = n.keySet().iterator();i.hasNext();) {
    Var v = (Var)i.next();
    al = this.getMult(true, v,true);
    ar = this.getMult(false,v,true);
    bl = that.getMult(true, v,true);
```

Behavioural Properties

- Deadlock-freedom
- Termination
- Isolation
- Determinism

Determinism Examples

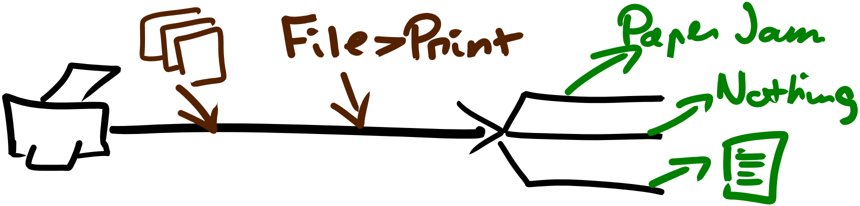
Deterministic System: Coffee Machine

Non-Deterministic System: Printer

Determinism Examples

Deterministic System: Coffee Machine

Non-Deterministic System: Printer



Behavioural Types

"Behavioural Type Systems: Algorithms Analysing Algorithms"

Who provides the types?

- Type *Checking*: The programmer
- Type *Inference*: The type system

When to type?

- *Static* Analysis: "Compile time"
- *Dynamic* Checking: Run time

Behavioural Types

"Behavioural Type Systems: Algorithms Analysing Algorithms"

Who provides the types?

- Type *Checking*: The programmer
- Type *Inference*: The type system

When to type?

- *Static* Analysis: "Compile time"
- *Dynamic* Checking: Run time

Behavioural Types

"Behavioural Type Systems: Algorithms Analysing Algorithms"

Who provides the types?

- Type *Checking*: The programmer
- Type *Inference*: The type system

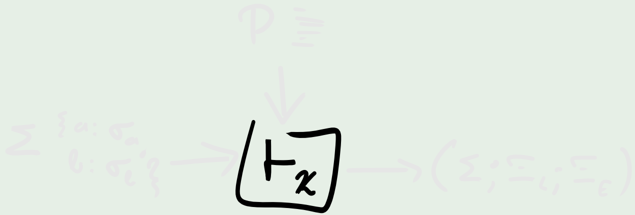
When to type?

- *Static* Analysis: "Compile time"
- *Dynamic* Checking: Run time

Type Systems

Finding/Verifying properties without running the program

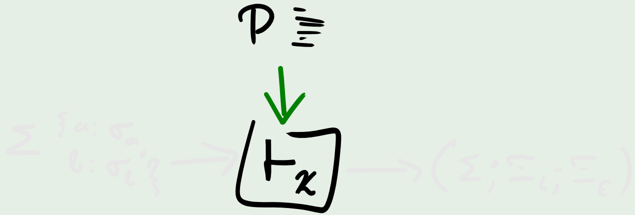
My Type Inference System



Type Systems

Finding/Verifying properties without running the program

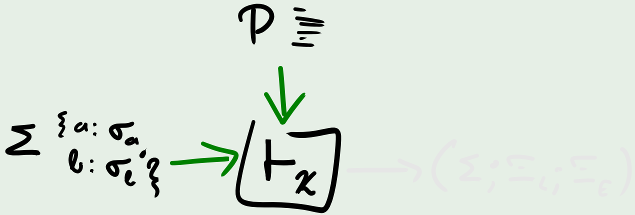
My Type Inference System



Type Systems

Finding/Verifying properties without running the program

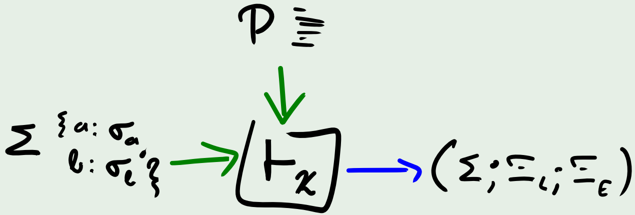
My Type Inference System



Type Systems

Finding/Verifying properties without running the program

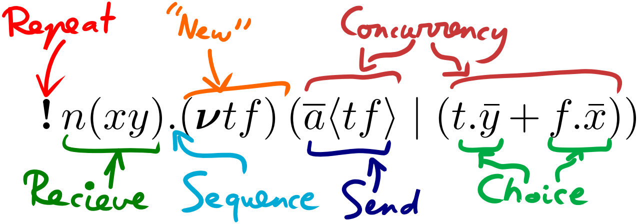
My Type Inference System



Process Calculi

"Behavioural Type Systems: Algorithms Analysing Algorithms"

The π-calculus: a tiny concurrent "programming language".



$$!\, n(xy).(\boldsymbol{\nu} t f)\,(\overline{a}\langle t f\rangle \mid (t.\bar{y} + f.\bar{x}))$$

Dependency Analysis (1)

"Behavioural Type Systems: Algorithms Analysing Algorithms"

- *Parallel composition* fundamental to the π-calculus

$$P = P_1 \mid P_2 \mid \ldots \mid P_n$$

- Need to analyse one component at a time

$$\{\Gamma_i \vdash P_i\} \mapsto (\Gamma \vdash P)$$

- Need to make assumptions on the *environment*

$$\Gamma_i = (\Xi_L \blacktriangleleft \Xi_E)$$

Dependency Analysis (1)

"Behavioural Type Systems: Algorithms Analysing Algorithms"

- *Parallel composition* fundamental to the π-calculus

$$P = P_1 \mid P_2 \mid \ldots \mid P_n$$

- Need to analyse one component at a time

$$\{\Gamma_i \vdash P_i\} \mapsto (\Gamma \vdash P)$$

- Need to make assumptions on the *environment*

$$\Gamma_i = (\Xi_L \blacktriangleleft \Xi_E)$$

Dependency Analysis (1)

"Behavioural Type Systems: Algorithms Analysing Algorithms"

- *Parallel composition* fundamental to the π-calculus

$$P = P_1 \mid P_2 \mid \ldots \mid P_n$$

- Need to analyse one component at a time

$$\{\Gamma_i \vdash P_i\} \mapsto (\Gamma \vdash P)$$

- Need to make assumptions on the *environment*

$$\Gamma_i = (\Xi_L \blacktriangleleft \Xi_E)$$

Dependency Analysis (2)

Definition (Behavioural Statements)

$$\Xi \quad ::= \quad (\gamma \lhd \Xi) \quad | \quad (\Xi \vee \Xi) \quad | \quad (\Xi \wedge \Xi) \quad | \quad \top \quad | \quad \bot$$

$$a_{\mathbf{D}} \lhd (b_{\mathbf{D}} \wedge c_{\mathbf{D}}) \quad \vdash \quad A \;=\; !\, a(tf).\overline{b}(\nu t' f').(t'.\overline{c}\langle tf\rangle + f'.\overline{f})$$

Dependency Analysis (2)

Definition (Behavioural Statements)

$$\Xi \quad ::= \quad (\gamma \lhd \Xi) \quad \mid \quad (\Xi \vee \Xi) \quad \mid \quad (\Xi \wedge \Xi) \quad \mid \quad \top \quad \mid \quad \bot$$

$$a_{\mathbf{D}} \lhd (b_{\mathbf{D}} \wedge c_{\mathbf{D}}) \quad \vdash \quad A \;=\; \mathbf{!}\, a(tf).\overline{b}(\boldsymbol{\nu} t'f').(t'.\overline{c}\langle tf\rangle + f'.\overline{f})$$

Dependency Analysis (2)

Definition (Behavioural Statements)

$$\Xi \;::=\; (\gamma \lhd \Xi) \;\mid\; (\Xi \vee \Xi) \;\mid\; (\Xi \wedge \Xi) \;\mid\; \top \;\mid\; \bot$$

$$a_{\mathbf{D}} \lhd (b_{\mathbf{D}} \wedge c_{\mathbf{D}}) \quad \vdash \quad A \;=\; \mathbf{!}\, a(tf).\overline{b}(\boldsymbol{\nu} t' f').(t'.\overline{c}\langle tf \rangle + f'.\overline{f})$$

$$b_{\mathbf{D}} \lhd \top \quad \vdash \quad B \;=\; \mathbf{!}\, b(tf).\overline{t}$$

Dependency Analysis (2)

Definition (Behavioural Statements)

$$\Xi \ ::= \ (\gamma \lhd \Xi) \ \mid \ (\Xi \vee \Xi) \ \mid \ (\Xi \wedge \Xi) \ \mid \ \top \ \mid \ \bot$$

$$\begin{array}{rcl} a_{\mathbf{D}} \lhd (b_{\mathbf{D}} \wedge c_{\mathbf{D}}) & \vdash & A = \,!\, a(tf).\overline{b}(\nu t' f').(t'.\overline{c}\langle tf \rangle + f'.\overline{f}) \\ b_{\mathbf{D}} \lhd \top & \vdash & B = \,!\, b(tf).\overline{t} \\ b_{\mathbf{D}} \lhd \bot & \vdash & C = \,!\, c(tf).((\nu q)\, \overline{q} \mid q.\overline{t} \mid q.\overline{f}) \end{array}$$

Dependency Analysis (2)

Definition (Behavioural Statements)

$$\Xi \quad ::= \quad (\gamma \lhd \Xi) \quad \mid \quad (\Xi \vee \Xi) \quad \mid \quad (\Xi \wedge \Xi) \quad \mid \quad \top \quad \mid \quad \bot$$

$$\begin{array}{rcl} a_{\mathbf{D}} \lhd (b_{\mathbf{D}} \wedge c_{\mathbf{D}}) & \vdash & A \;=\; \mathbf{!}\, a(tf).\overline{b}(\boldsymbol{\nu} t' f').(t'.\overline{c}\langle tf \rangle + f'.\overline{f}) \\ b_{\mathbf{D}} \lhd \top & \vdash & B \;=\; \mathbf{!}\, b(tf).\overline{t} \\ b_{\mathbf{D}} \lhd \bot & \vdash & C \;=\; \mathbf{!}\, c(tf).((\boldsymbol{\nu} q)\, \overline{q} \mid q.\overline{t} \mid q.\overline{f}) \end{array}$$

$$a_{\mathbf{D}} \lhd (\top \wedge c_{\mathbf{D}}) \cong a_{\mathbf{D}} \lhd c_{\mathbf{D}} \quad \vdash \quad A \mid B$$

Dependency Analysis (2)

Definition (Behavioural Statements)

$$\Xi \quad ::= \quad (\gamma \lhd \Xi) \quad | \quad (\Xi \vee \Xi) \quad | \quad (\Xi \wedge \Xi) \quad | \quad \top \quad | \quad \bot$$

$$a_{\mathbf{D}} \lhd (b_{\mathbf{D}} \wedge c_{\mathbf{D}}) \quad \vdash \quad A = \boldsymbol{!}\, a(tf).\overline{b}(\boldsymbol{\nu} t'f').(t'.\overline{c}\langle tf\rangle + f'.\overline{f})$$

$$b_{\mathbf{D}} \lhd \top \quad \vdash \quad B = \boldsymbol{!}\, b(tf).\overline{t}$$

$$b_{\mathbf{D}} \lhd \bot \quad \vdash \quad C = \boldsymbol{!}\, c(tf).((\boldsymbol{\nu} q)\,\overline{q} \mid q.\overline{t} \mid q.\overline{f})$$

$$a_{\mathbf{D}} \lhd (\top \wedge c_{\mathbf{D}}) \cong a_{\mathbf{D}} \lhd c_{\mathbf{D}} \quad \vdash \quad A \mid B$$

$$a_{\mathbf{D}} \lhd (b_{\mathbf{D}} \wedge \bot) \cong a_{\mathbf{D}} \lhd \bot \quad \vdash \quad A \mid C$$

Generic Type Systems

Captures the essence of dependency analysis

Can be *instantiated*:

- Write *semantic goals*
- Rules *parametrised* by *elementary rules*

Generic Type Systems

Captures the essence of dependency analysis
Can be *instantiated*:

- Write *semantic goals*
- Rules *parametrised* by *elementary rules*

Generic Type Systems

Captures the essence of dependency analysis
Can be *instantiated*:

- Write *semantic goals*
- Rules *parametrised* by *elementary rules*

Summary

- Behavioural Properties
- Type Inference Systems: Find properties automatically
- The π-calculus: A simple programming language
- Dependency Analysis: Reusable types for reusable code
- Generic Type Systems: Write an elementary rule, get a type system for free

Summary

- Behavioural Properties
- Type Inference Systems: Find properties automatically
- The π-calculus: A simple programming language
- Dependency Analysis: Reusable types for reusable code
- Generic Type Systems: Write an elementary rule, get a type system for free

Summary

- Behavioural Properties
- Type Inference Systems: Find properties automatically
- The π-calculus: A simple programming language
- Dependency Analysis: Reusable types for reusable code
- Generic Type Systems: Write an elementary rule, get a type system for free

Summary

- Behavioural Properties
- Type Inference Systems: Find properties automatically
- The π-calculus: A simple programming language
- Dependency Analysis: Reusable types for reusable code
- Generic Type Systems: Write an elementary rule, get a type system for free

Summary

- Behavioural Properties
- Type Inference Systems: Find properties automatically
- The π-calculus: A simple programming language
- Dependency Analysis: Reusable types for reusable code
- Generic Type Systems: Write an elementary rule, get a type system for free

Summary

- Behavioural Properties
- Type Inference Systems: Find properties automatically
- The π-calculus: A simple programming language
- Dependency Analysis: Reusable types for reusable code
- Generic Type Systems: Write an elementary rule, get a type system for free

Thank You

Answers to questions are non-isolated, non-deterministic, non-uniform, active, responsive, deadlock-free, and terminate.

Supplementary Material

Types & Multiplicities

Behavioural Statements $\Delta, \Xi, \ldots$

$$\Delta \quad ::= \quad \Delta \vee \Delta \;\mid\; \Delta + \Delta \;\mid\; \Delta \wedge \Delta \;\mid\; \Delta \lhd \Delta \;\mid\; p_k \;\mid\; \bot \;\mid\; \top \;\mid\; p^m$$

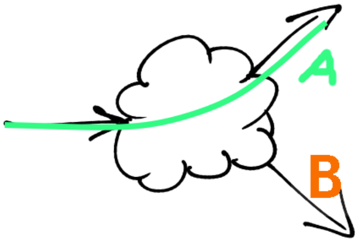
Multiplicities

$$m \quad ::= \quad 0 \;\mid\; 1 \;\mid\; \omega \;\mid\; \star$$

Choice

Definition (Selection $A \vee B$)

I will either behave like A or like B



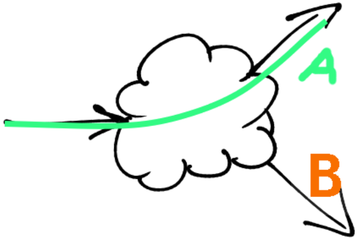
Definition (Branching $A + B$)

You can make me do A or B

Choice

Definition (Selection $A \vee B$)

I will either behave like A or like B



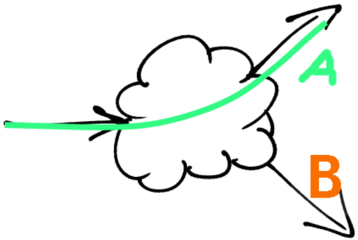
Definition (Branching $A + B$)

You can make me do A or B

Choice

Definition (Selection $A \vee B$)

I will either behave like A or like B



Definition (Branching $A + B$)

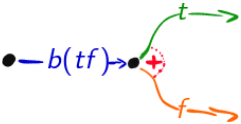
You can make me do A or B

Choice Examples (I)

- Data Encodings

$$b := \text{True} \stackrel{\text{def}}{=} !\, b(tf).\overline{t}$$

$$b := \text{False} \stackrel{\text{def}}{=} !\, b(tf).\overline{f}$$

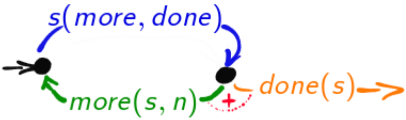


$$\text{If } b \text{ Then } P \text{ Else } Q \stackrel{\text{def}}{=} \overline{b}(\nu tf).(t.P + f.Q)$$

Choice Examples (II)

- Client-Server Conversations

$$\overline{prod}(\boldsymbol{\nu} s).s(more, done).\overline{more}(\boldsymbol{\nu} s, 2).$$
$$s(more, done).\overline{more}(\boldsymbol{\nu} s, 5).$$
$$s(more, done).\overline{done}(\boldsymbol{\nu} s).s(x).\overline{print}\langle x\rangle$$



$$!\, prod(s).\overline{p_0}\langle 1, s\rangle \quad | \quad !\, p_0(t, s).\overline{s}(\boldsymbol{\nu} more, done).$$
$$\big(more(s, n).\overline{p_0}\langle t \times n, s\rangle + done(s).\overline{s}\langle r\rangle\big)$$

Algebra

Spatial Operators

Parallel Composition $\Gamma_1 \odot \Gamma_2$, Restriction $(\boldsymbol{\nu} x)\,\Gamma$, ...

Logical Operators

Equivalence $\cong$, Weakening $\leq$, Reduction $\hookrightarrow$, ...

Dynamic Operator

Transition Operator $\Gamma \xrightarrow{\mu} (\Gamma \wr \mu)$.

Semantics (Universal)

Definition (Universal Semantics)

A $(\Gamma; P)$ typed process is *correct wrt. universal semantics* ("$\Gamma \models_{\mathcal{U}} P$") if, for all transition sequences $(\Gamma; P) \xrightarrow{\tilde{\mu}}\searrow (\Gamma'; P')$, the local component of Γ' being $\bigvee_{i \in I} p_{i\,k_i} \lhd \varepsilon_i$: for all $i \in I$ with $k_i \in \mathcal{U}$, $\text{good}_{k_i}(p_i \lhd \varepsilon_i, (\Gamma'; P'))$ holds.

Semantics (Existential)

(Abbreviated) Existential Semantics

A typed process $(\Gamma; P)$ is *correct* ("$\Gamma \models P$"), if $\exists$ a strategy f s.t. For any sequence $(\Gamma; P) = (\Gamma_0; P_0) \cdots \xrightarrow{\tilde{\mu}_i}\searrow (\Gamma'_i; P'_i) \xrightarrow{f} (\Gamma_{i+1}; P_{i+1}) \cdots$, let (for all i) μ_i be the label of $(\Gamma'_i; P'_i) \xrightarrow{f} (\Gamma_{i+1}; P_{i+1})$.
Then $\exists$ a resource p_k and $n \geqslant 0$ such that:

1. $\forall i : (p_k \lhd \mathrm{dep}_{\mathcal{K}}(\mu_i)) \preceq \Gamma'_i$
2. $\exists \varepsilon : (p_k \lhd \varepsilon) \preceq \Gamma_n$ and $\mathrm{good}_k(p \lhd \varepsilon, (\Gamma_n; P_n))$.

Type System (Universal)

$$\frac{\forall i : \Gamma_i \vdash_{\mathcal{K}} P_i}{\Gamma_1 \odot \Gamma_2 \vdash_{\mathcal{K}} P_1 \mid P_2}\ (\text{U-PAR}) \qquad \frac{\Gamma \vdash_{\mathcal{K}} P \qquad \Gamma(x) = \sigma}{(\boldsymbol{\nu} x)\,\Gamma \vdash_{\mathcal{K}} (\boldsymbol{\nu} x : \sigma)\,P}\ (\text{U-RES})$$

$$\frac{\begin{array}{c}\forall i : (\Sigma_i; \Xi_{\mathrm{L}i} \blacktriangleleft \Xi_{\mathrm{E}i}) \vdash_{\mathcal{K}} G_i.P_i \\ \Xi_{\mathrm{E}} \preceq \bigwedge_i \Xi_{\mathrm{E}i}\end{array}}{(\bigwedge_i \Sigma_i; \bigwedge_{k \in \mathcal{K}} \mathsf{sum}_k(\{p_i\}_i, \Xi_{\mathrm{E}}) \wedge \bigvee_i \Xi_{\mathrm{L}i} \blacktriangleleft \Xi_{\mathrm{E}}) \vdash_{\mathcal{K}} \sum_i G_i.P_i}\ (\text{U-SUM})$$

$$\frac{\Gamma \vdash_{\mathcal{K}} P \quad \mathsf{sub}(G) = p \quad \mathsf{obj}(G) = \tilde{x}}{\begin{array}{r}\left(p : \sigma; \blacktriangleleft p^m \wedge \bar{p}^{m'}\right) \quad \odot \\ \left(; p^{\#(G)} \blacktriangleleft\right) \quad \odot \\ !_{\text{if } \#(G) = \omega}\ (\boldsymbol{\nu}\mathsf{bn}(G)) \Big(\Gamma \quad \odot \\ \overline{\sigma}[\tilde{x}] \quad \odot \\ (; \bigwedge_{k \in \mathcal{K}} \mathsf{prop}_k(\sigma, G, m, m') \blacktriangleleft) \Big) \quad \vdash_{\mathcal{K}} G.P\end{array}}\ (\text{U-PRE})$$

Type System (Existential)

$$\frac{\forall i : \Gamma_i \vdash_{\mathcal{K}} P_i}{\Gamma_1 \odot \Gamma_2 \vdash_{\mathcal{K}} P_1 \mid P_2} \text{ (E-PAR)} \qquad \frac{\Gamma \vdash_{\mathcal{K}} P \qquad \Gamma(x) = \sigma}{(\boldsymbol{\nu} x)\,\Gamma \vdash_{\mathcal{K}} (\boldsymbol{\nu} x : \sigma)\, P} \text{ (E-RES)}$$

$$\frac{\begin{array}{c}\forall i : (\Sigma_i; \Xi_{\mathrm{L}i} \blacktriangleleft \Xi_{\mathrm{E}i}) \vdash_{\mathcal{K}} G_i.P_i \\ \Xi_{\mathrm{E}} \preceq \bigwedge_i \Xi_{\mathrm{E}i}\end{array}}{(\bigwedge_i \Sigma_i; \bigwedge_{k\in\mathcal{K}} \mathsf{sum}_k(\{p_i\}_i, \Xi_{\mathrm{E}}) \wedge \bigvee_i \Xi_{\mathrm{L}i} \blacktriangleleft \Xi_{\mathrm{E}}) \vdash_{\mathcal{K}} \sum_i G_i.P_i} \text{ (E-SUM)}$$

$$\frac{\Gamma \vdash_{\mathcal{K}} P \quad \mathsf{sub}(G) = p \quad \mathsf{obj}(G) = \tilde{x}}{\begin{array}{r} \left(p : \sigma; \blacktriangleleft p^{m} \wedge \bar{p}^{m'}\right) \odot \\ \left(; p^{\#(G)} \blacktriangleleft\right) \odot \\ !_{\text{if } \#(G) = \omega}\, (\boldsymbol{\nu}\mathsf{bn}(G)) \Big(\Gamma \lhd \color{red}{\mathsf{dep}_{\mathcal{K}}(G)} \odot \\ \bar{\sigma}[\tilde{x}] \lhd \color{red}{(\mathsf{dep}_{\mathcal{K}}(G) \wedge \bar{p}_{\mathbf{R}})} \odot \\ (; \bigwedge_{k\in\mathcal{K}} \mathsf{prop}_k(\sigma, G, m, m') \blacktriangleleft) \Big) \vdash_{\mathcal{K}} G.P \end{array}} \text{ (E-PRE)}$$

Properties

- **A** — Activeness

$$\mathsf{prop}_{\mathbf{A}}(G,\sigma,m,m') = \begin{cases} \mathsf{sub}(G)_{\mathbf{A}} & \text{if } \#(G) = \omega \text{ or } m' \neq \star \\ \top & \text{otherwise} \end{cases}$$

- **R** — Responsiveness
- **D** — Determinism (Functionality)
- **I** — Isolation
- **df** — Lock-Freedom
- **N** — Non-Reachability
- ϖ — Termination

Properties

- **A** — Activeness
- **R** — Responsiveness

$$\mathsf{prop}_{\mathbf{R}}(\sigma, G, m, m') = \mathsf{sub}(G)_{\mathbf{R}} \lhd \begin{cases} \sigma[\mathsf{obj}(G)] & \text{if } G \text{ is an input} \\ \overline{\sigma}[\mathsf{obj}(G)] & \text{if } G \text{ is an output} \end{cases}$$

- **D** — Determinism (Functionality)
- **I** — Isolation
- **df** — Lock-Freedom
- **N** — Non-Reachability
- ϖ — Termination

Properties

- **A** — Activeness
- **R** — Responsiveness
- **D** — Determinism (Functionality)

$$\varphi_{\mathbf{D}}(\sigma, G, m, m') \stackrel{\text{def}}{=} \begin{cases} \bot & \text{if } \star \in \{m, m'\} \text{ and } \omega \notin \{m, m'\} \\ \overline{\text{sub}(G)_{\mathbf{D}}} & \text{otherwise} \end{cases}$$

$$\varphi_{\mathbf{D}}(\{p_i\}_i, \Xi) \stackrel{\text{def}}{=} \begin{cases} \bot & \text{if } \Xi \text{ has concurrent environment } p_i \\ \top & \text{otherwise} \end{cases}$$

- **I** — Isolation
- **df** — Lock-Freedom
- **N** — Non-Reachability
- ϖ — Termination

Properties

- **A** — Activeness
- **R** — Responsiveness
- **D** — Determinism (Functionality)
- **I** — Isolation

$$\varphi_{\mathbf{I}}(\sigma, G, m, m') = \overline{\mathsf{sub}(G)}_{\mathbf{I}}$$

- **df** — Lock-Freedom
- **N** — Non-Reachability
- ϖ — Termination

Properties

- **A** — Activeness
- **R** — Responsiveness
- **D** — Determinism (Functionality)
- **I** — Isolation
- **df** — Lock-Freedom

$$\mathsf{prop}_{\mathbf{df}}(G, \sigma, m, m') = \mathsf{proc}_{\mathbf{df}} \lhd \overline{\mathsf{sub}(G)}_{\mathbf{A}}$$

- **N** — Non-Reachability
- ϖ — Termination

Properties

- **A** — Activeness
- **R** — Responsiveness
- **D** — Determinism (Functionality)
- **I** — Isolation
- **df** — Lock-Freedom
- **N** — Non-Reachability

$$\mathsf{prop}_{\mathbf{N}}(G, \sigma, m, m') \stackrel{\text{def}}{=} \mathsf{sub}(G)_{\mathbf{N}} \lhd \bot$$

- ϖ — Termination

Properties

- **A** — Activeness
- **R** — Responsiveness
- **D** — Determinism (Functionality)
- **I** — Isolation
- **df** — Lock-Freedom
- **N** — Non-Reachability
- ϖ — Termination

$$\mathsf{prop}_{\mathbf{N}}(G, \sigma, m, m') \stackrel{\text{def}}{=} \mathsf{sub}(G)_{\mathbf{N}} \lhd \perp \wedge \tau_{\mathbf{N}} \lhd \overline{\mathsf{sub}(G)}_{\mathbf{N}}$$

Universal Soundness

- Based on transition sequences?
  Semantic Predicates aren't transition based! (or are they?)
- Based on contextual semantics?
  "$\Delta_1 \lhd \Delta_2 \models P$ if $\forall Q$ s.t. $\Delta_2 \vdash Q$: $\Delta_1 \models P \mid Q$."
  The definition is circular!
- Implicit definition?
  "The set of correct typed processes is the largest that satisfies the above"
  There are many solutions!
- Stricter implicit definition?
  "The set of correct typed processes is the intersection of all those that satisfy the above"
  The intersection is empty!
- To be continued ...

Universal Soundness

- Based on transition sequences?
  Semantic Predicates aren't transition based! (or are they?)
- Based on contextual semantics?
  "$\Delta_1 \lhd \Delta_2 \models P$ if $\forall Q$ s.t. $\Delta_2 \vdash Q$: $\Delta_1 \models P \mid Q$."
  The definition is circular!
- Implicit definition?
  "The set of correct typed processes is the largest that satisfies the above"
  There are many solutions!
- Stricter implicit definition?
  "The set of correct typed processes is the intersection of all those that satisfy the above"
  The intersection is empty!
- To be continued ...

Universal Soundness

- Based on transition sequences?
  Semantic Predicates aren't transition based! (or are they?)
- Based on contextual semantics?
  "$\Delta_1 \lhd \Delta_2 \models P$ if $\forall Q$ s.t. $\Delta_2 \vdash Q$: $\Delta_1 \models P \mid Q$."
  The definition is circular!
- Implicit definition?
  "The set of correct typed processes is the largest that satisfies the above"
  There are many solutions!
- Stricter implicit definition?
  "The set of correct typed processes is the intersection of all those that satisfy the above"
  The intersection is empty!
- To be continued ...

Universal Soundness

- Based on transition sequences?
  Semantic Predicates aren't transition based! (or are they?)
- Based on contextual semantics?
  "$\Delta_1 \lhd \Delta_2 \models P$ if $\forall Q$ s.t. $\Delta_2 \vdash Q$: $\Delta_1 \models P \mid Q$."
  The definition is circular!
- Implicit definition?
  "The set of correct typed processes is the largest that satisfies the above"
  There are many solutions!
- Stricter implicit definition?
  "The set of correct typed processes is the intersection of all those that satisfy the above"
  The intersection is empty!
- To be continued ...

Universal Soundness

- Based on transition sequences?
  Semantic Predicates aren't transition based! (or are they?)
- Based on contextual semantics?
  "$\Delta_1 \lhd \Delta_2 \models P$ if $\forall Q$ s.t. $\Delta_2 \vdash Q$: $\Delta_1 \models P \mid Q$."
  The definition is circular!
- Implicit definition?
  "The set of correct typed processes is the largest that satisfies the above"
  There are many solutions!
- Stricter implicit definition?
  "The set of correct typed processes is the intersection of all those that satisfy the above"
  The intersection is empty!
- To be continued . . .

Universal Soundness

- Based on transition sequences?
  Semantic Predicates aren't transition based! (or are they?)
- Based on contextual semantics?
  "$\Delta_1 \lhd \Delta_2 \models P$ if $\forall Q$ s.t. $\Delta_2 \vdash Q$: $\Delta_1 \models P \mid Q$."
  The definition is circular!
- Implicit definition?
  "The set of correct typed processes is the largest that satisfies the above"
  There are many solutions!
- Stricter implicit definition?
  "The set of correct typed processes is the intersection of all those that satisfy the above"
  The intersection is empty!
- To be continued ...

Existential Soundness

Structural Liveness Strategies

$$
\begin{array}{rcl}
\rho & ::= & \pi\delta \quad | \quad \mathfrak{l} \quad | \quad \cdots \\
\delta & ::= & \div\ \rho \quad | \quad [\, s\,] \\
\pi & ::= & (\mathfrak{l}|\rho) \quad | \quad (\mathfrak{l}|\bullet) \quad | \quad (\bullet|\rho) \\
s & ::= & p_1 + p_2 + p_3 \ldots
\end{array}
$$

$\mathfrak{l}$: Guard reference

$\bullet$: Environment

$(\mathfrak{l}|\rho)$: Make $\mathfrak{l}$ and ρ communicate.

Future Work

- Generic Universal Soundness Proof
- Recursivity and **B**ounded Channels.
- Channel Type Reconstruction.
- Software Implementation.

Link to Appendices