# Undecidability of Asynchronous Session Subtyping:

## the Hunt for Significant Decidable Variants

**Mario Bravetti**

Department of Computer Science
University of Bologna

# Session Types

- Session types are types for controlling the communication behaviour of processes over channels.
  - they express the pattern of sends and receives that a process must perform
  - they can guarantee freedom from (some) communication errors, i.e. locking/data types
    - becoming popular with main stream language implementations, e.g., Haskell, GO, or RUST.

# Simple binary session types

**Definition** (Session types). *Given a set of labels $L$, ranged over by $l$, the syntax of binary session types is given by the following grammar:*

$$T \quad ::= \quad \oplus\{l_i : T_i\}_{i \in I} \quad | \quad \&\{l_i : T_i\}_{i \in I} \quad | \quad \mu \mathbf{t}.T \quad | \quad \mathbf{t} \quad | \quad \mathbf{end}$$

selection among outputs

branching among inputs

recursion

termination (success)

# Session Subtyping

- Traditional notion of subtyping in programming languages
  - a given program with type $T$ can be used in place of program with type $S$ whenever $T \leq S$ ($T$ is a subtype of $S$)

# Subtyping: output Covariance and input Contravariance

- Output Covariance
  - subtype may have a subset of outputs
  - example:

$$\oplus\{l_1 : T_1\} \leq \oplus\{l_1 : T_1, l_2 : T_2\}$$

- Input Contravariance
  - subtype may have a superset of inputs
  - example:

$$\&\{l_1 : T_1, l_2 : T_2\} \leq \&\{l_1 : T_1\}$$

# Synchronous Session Subtyping

**Definition** (Synchronous Subtyping, $\leq$)**.** *$\mathcal{R}$ is a synchronous subtyping relation whenever $(T, S) \in \mathcal{R}$ implies that:*

1. *if $T = \mathbf{end}$ then $\exists n \geq 0$ such that $\mathsf{unfold}^n(S) = \mathbf{end}$;*
2. *if $T = \oplus\{l_i : T_i\}_{i \in I}$ then $\exists n \geq 0$ such that $\mathsf{unfold}^n(S) = \oplus\{l_j : S_j\}_{j \in J}$, $I \subseteq J$ and $\forall i \in I. (T_i, S_i) \in \mathcal{R}$;*
3. *if $T = \&\{l_i : T_i\}_{i \in I}$ then $\exists n \geq 0$ such that $\mathsf{unfold}^n(S) = \&\{l_j : S_j\}_{j \in J}$, $J \subseteq I$ and $\forall j \in J. (T_j, S_j) \in \mathcal{R}$;*
4. *if $T = \mu \mathbf{t}.T'$ then $(T'\{T/\mathbf{t}\}, S) \in \mathcal{R}$.*

*$T$ is a synchronous subtype of $S$, written $T \leq S$, if there is a synchronous subtyping relation $\mathcal{R}$ such that $(T, S) \in \mathcal{R}$.*

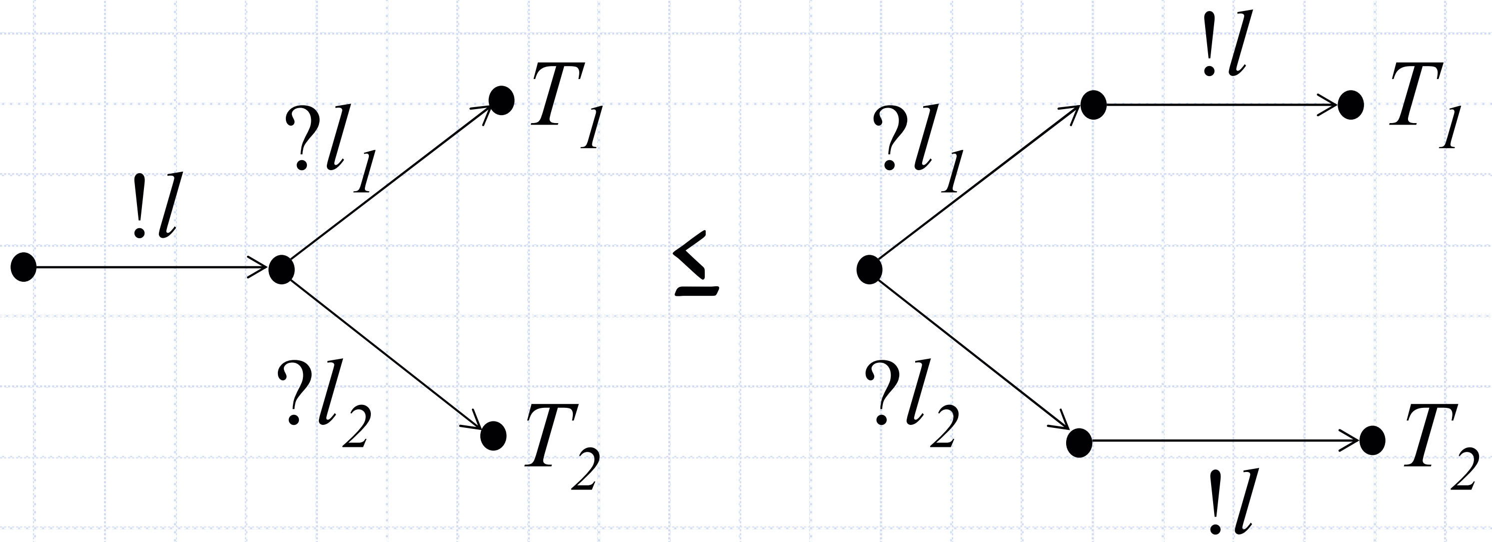
# Asynchronous Subtyping

- Bidirectional asynchronous channel with unbounded queues
  - messages sent by outputs are received in an ordered manner
- subtype may have outputs anticipated w.r.t. inputs (but order w.r.t. alike preserved)
  - example:

$$\oplus\{l : \&\{l_1 : T_1, l_2 : T_2\}\} \leq \&\{l_1 : \oplus\{l : T_1\}, l_2 : \oplus\{l : T_2\}\}$$

# Asynchronous Subtyping

$$\oplus\{l : \&\{l_1 : T_1, l_2 : T_2\}\} \leq \&\{l_1 : \oplus\{l : T_1\}, l_2 : \oplus\{l : T_2\}\}$$



# Input Context

**Definition** (Input Context)**.** *An input context $\mathcal{A}$ is a session type with multiple holes defined by the syntax:*

$$\mathcal{A} \quad ::= \quad [\,]^n \quad | \quad \&\{l_i : \mathcal{A}_i\}_{i \in I}$$

- Using input contexts:

$\mathcal{A}[T_k]^{k \in \{1,\ldots,m\}}$ *denotes the type obtained by filling each hole $k$ in $\mathcal{A}$ with $T_k$*

$$\&\{l_1 : \oplus\{l : T_1\}, l_2 : \oplus\{l : T_2\}\} \quad = \quad \&\{l_1 : [\,]^1, l_2 : [\,]^2\}$$

$$\oplus\{l : T_1\} \qquad \oplus\{l : T_2\}$$

# Asynchronous Session Subtyping

**Definition** (Asynchronous Subtyping, $\leq$). *$\mathcal{R}$ is an asynchronous subtyping relation whenever $(T, S) \in \mathcal{R}$ implies that:*

1. *if* $T = \mathbf{end}$ *then* $\exists n \geq 0$ *such that* $\mathsf{unfold}^n(S) = \mathbf{end}$;
2. *if* $T = \oplus\{l_i : T_i\}_{i \in I}$ *then* $\exists n \geq 0, \mathcal{A}$ *such that*
   - $\mathsf{unfold}^n(S) = \mathcal{A}[\oplus\{l_j : S_{kj}\}_{j \in J_k}]^{k \in \{1,\ldots,m\}}$,
   - $\forall k \in \{1, \ldots, m\}. I \subseteq J_k$ *and*
   - $\forall i \in I. (T_i, \mathcal{A}[S_{ki}]^{k \in \{1,\ldots,m\}}) \in \mathcal{R}$;
3. *if* $T = \&\{l_i : T_i\}_{i \in I}$ *then* $\exists n \geq 0$ *such that* $\mathsf{unfold}^n(S) = \&\{l_j : S_j\}_{j \in J}$, $J \subseteq I$ *and* $\forall j \in J. (T_j, S_j) \in \mathcal{R}$;
4. *if* $T = \mu\mathbf{t}.T'$ *then* $(T'\{T/\mathbf{t}\}, S) \in \mathcal{R}$.

*$T$ is an asynchronous subtype of $S$, written $T \leq S$, if there is an asynchronous subtyping relation $\mathcal{R}$ such that $(T, S) \in \mathcal{R}$.*

# Undecidability of Asynchronous Session Subtyping [BCZ InfoComp17]

- We prove undecidability of asynchronous subtyping by reduction from the termination problem for queue machines
- Queue machines are a formalism similar to pushdown automata, but with a queue instead of a stack.
- Queue machines are Turing equivalent

# Queue Machine

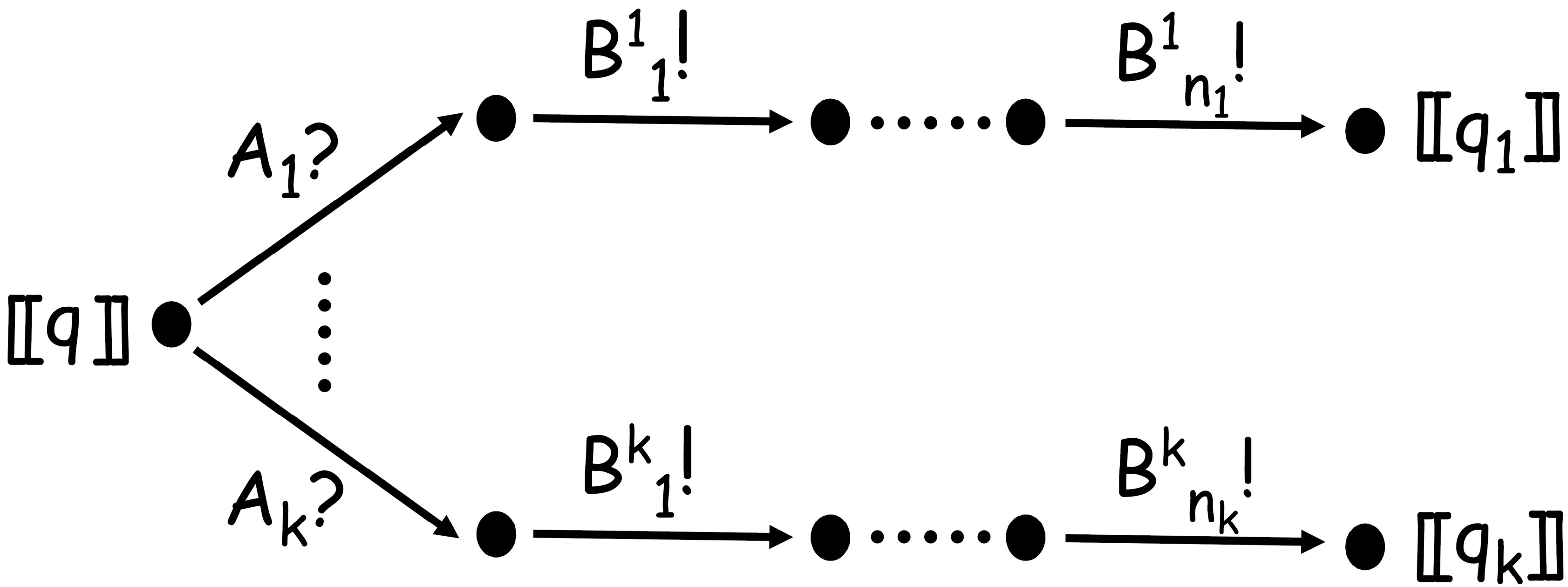
**Definition** (Queue machine). *A queue machine $M$ is defined by a six-tuple $(Q, \Sigma, \Gamma, \$, s, \delta)$ where:*

- $Q$ *is a finite set of states;*
- $\Sigma \subset \Gamma$ *is a finite set denoting the input alphabet;*
- $\Gamma$ *is a finite set denoting the queue alphabet;*
- $\$ \in \Gamma - \Sigma$ *is the initial queue symbol;*
- $s \in Q$ *is the start state;*
- $\delta : Q \times \Gamma \to Q \times \Gamma^*$ *is the transition function.*

# Queue Machine Execution

- *A* **configuration** *of a queue machine is an ordered pair* $(q, \gamma)$ *where* $q \in Q$ *is its current state and* $\gamma \in \Gamma^*$ *is the content of the queue.*
- *The starting configuration on an input string* $x$ *is* $(s, x\$)$.
- *The transition relation* $\rightarrow_M$ *from one configuration to the next one is defined as* $(p, A\alpha) \rightarrow_M (q, \alpha\gamma)$, *when* $\delta(p, A) = (q, \gamma)$.
- *A machine* $M$ **accepts** *an input* $x$ *if it blocks by emptying the queue.*
  - *Formally,* $x$ *is accepted by* $M$ *if* $(s, x\$) \rightarrow_M^* (q, \epsilon)$ *where* $\epsilon$ *is the empty string and* $\rightarrow_M^*$ *is the reflexive and transitive closure of* $\rightarrow_M$.

# Control encoding

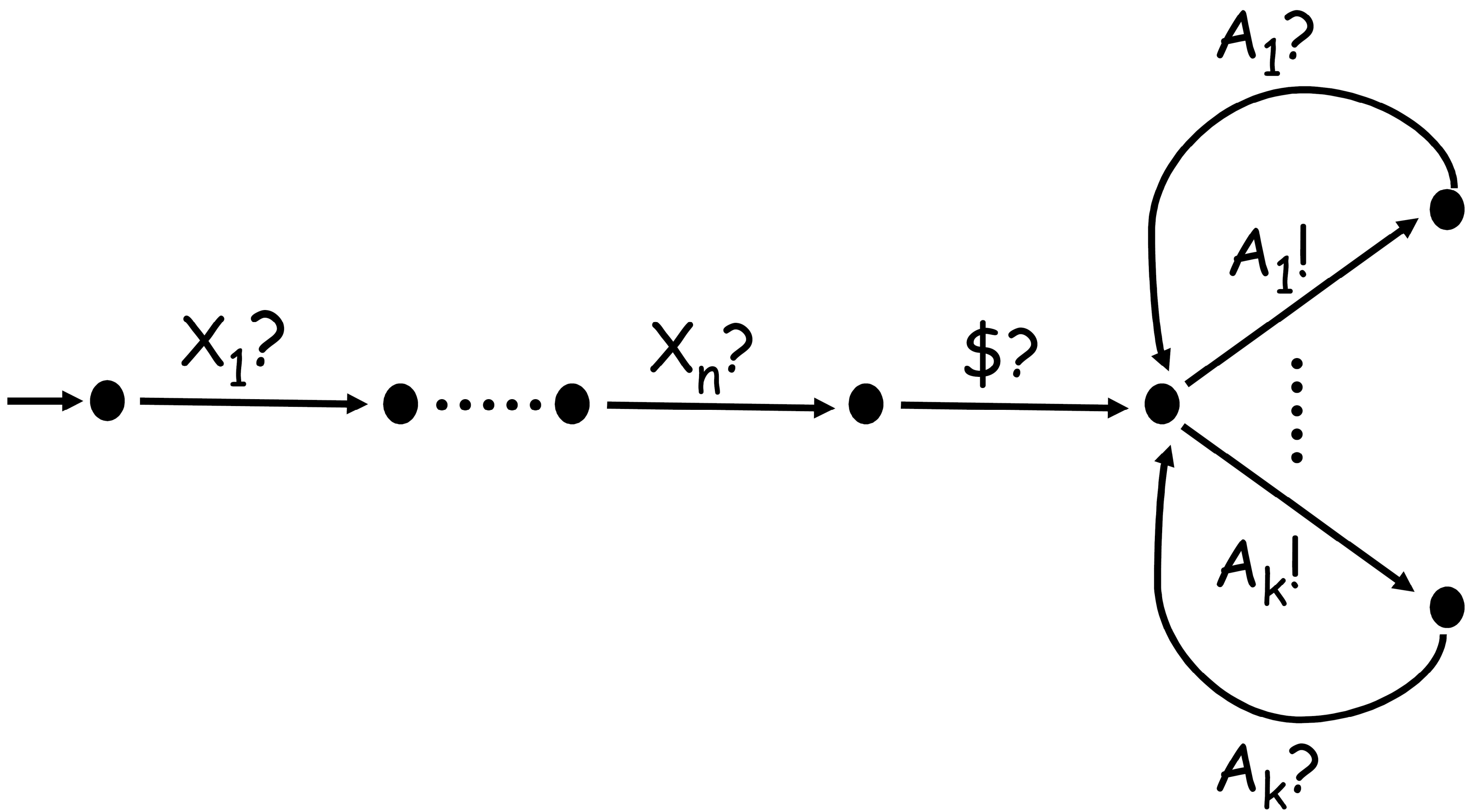



where:

$$\Gamma = \{A_i | i \leq k\}$$

$$\delta(q, A_i) = (q_i, B_1^i \cdots B_{n_i}^i) \text{ for every } i$$

# Queue encoding



# We have: subtyping corresponds to non-termination

- Our encoding yields an immediate correspondance between subtyping and (non) termination

**Theorem.** *Given a queue machine* $M = (Q, \Sigma, \Gamma, \$, s, \delta)$ *and an input string* $x \in \Sigma^*$, *we have* $[\![s]\!] \leq [\![x\$]\!]$ *if and only if* $M$ *does not terminate on* $x$.

**Corollary.** *Asynchronous subtyping* $\leq$ *is undecidable.*

# Output Covariance and Input Contravariance are not needed

- Undecidability of Asynchronous Subtyping can also be shown without resorting to
  - Output Covariance
    - possibility, in $T \leq S$, for T to have a subset of outputs
  - Input Contravariance
    - possibility, in $T \leq S$, for T to have a superset of inputs

# Some insight in the T ≤ S decidability problem

- Procedure just enacting the simulation game (S simulates moves of T) may not terminate in case T ≤ S holds
- Even adding a check that a pair T' ≤ S' has been already met [MYH ESOP 09] is not enough

**Example.** $T = \mu\mathbf{t}.\oplus\{l_1 : \&\{l_2 : \mathbf{t}\}\}$ *and* $S = \mu\mathbf{t}.\oplus\Big\{l_1 : \&\{l_2 : \&\{l_2 : \mathbf{t}\}\}\Big\}$

# Decidability of k-bounded Asynchronous Subtyping

- If we establish a bound k for the capability of anticipating outputs, we get termination

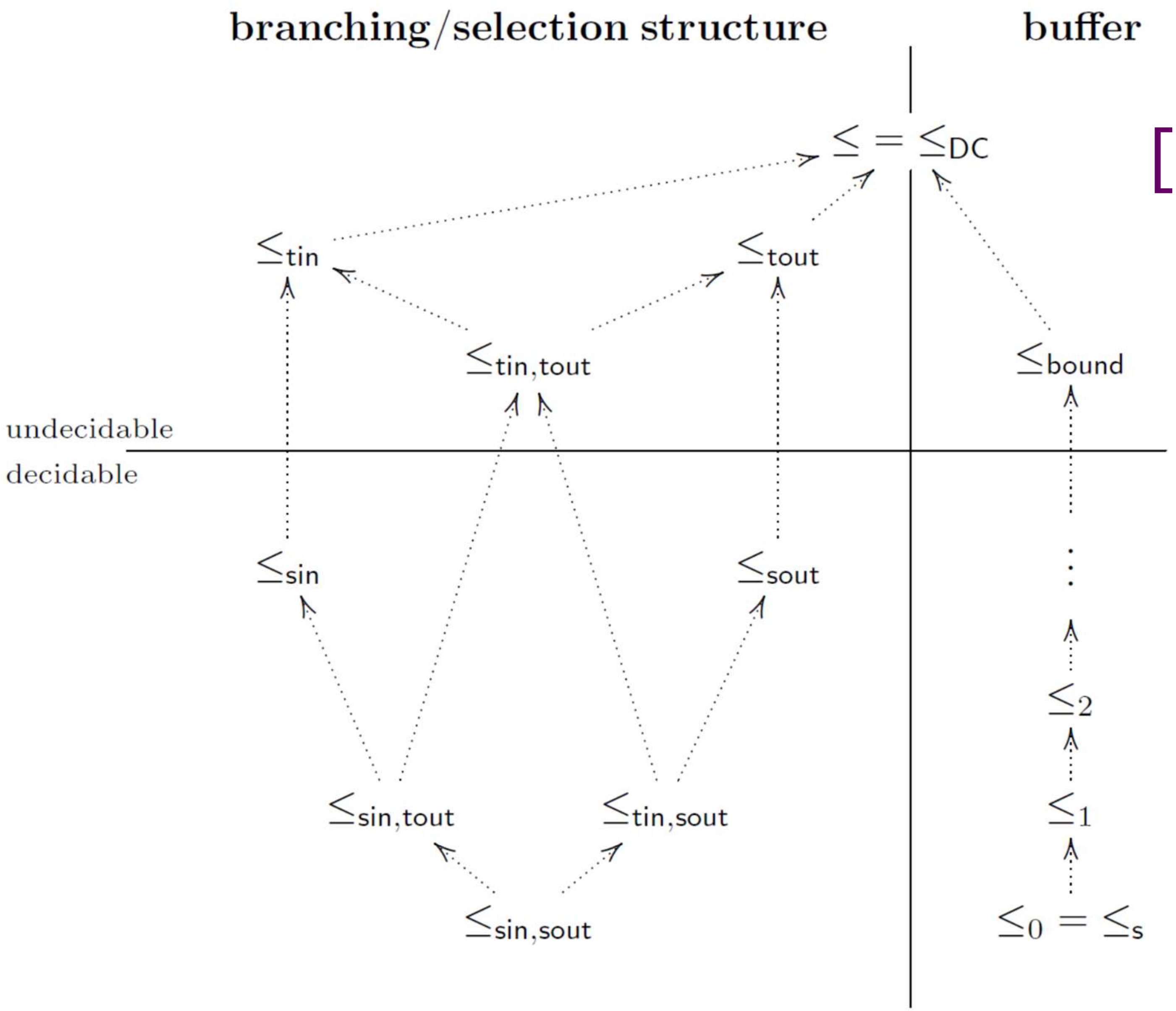
We say that an input context $\mathcal{A}$ is *$k$-bounded* if the maximal number of nested inputs in $\mathcal{A}$ is less or equal to $k$.

**Definition** ($k$-bounded Asynchronous Subtyping)**.** *The $k$-bounded asynchronous subtyping $\leq_k$ is defined as before, with the only difference that the input context $\mathcal{A}$ in item 2. is assumed to be $k$-bounded*

# Decidability of Subtyping for Single-Out and Single-In Types

- Algorithm that terminates if types are restricted to be single-out only or single-in only
  - Single-out session types are types where output selections are always singleton
    - common in web-services where a server accepts alternative clients requests but then it reacts deterministically
  - Single-in session types are types where input branches are always singleton
    - common in web-services where client code internally choses outputs and the corresponding inputs are always singletons
  - Our algorithm is thus usable in typing systems for client and server code.

[BCZ TCS18]




$\leq_s$ synchronous subtyping

$\leq$ orphan-message-free asynchronous subtyping

⋯⋯> set inclusion

# Orphan-message-free Subtyping

**Definition** (Orphan-Message-Free Subtyping, $\leq$). *$\mathcal{R}$ is an orphan-message-free subtyping relation whenever $(T, S) \in \mathcal{R}$ implies items 1., 3., and 4., plus an extended version of 2. that contains also the following requirement:*

- *if $\mathcal{A} \neq [\,]^1$ then $\forall i \in I. \& \in T_i$*

*$T$ is a orphan-message-free subtype of $S$, simply written $T \leq S$, if there is a orphan-message-free subtyping relation $\mathcal{R}$ such that $(T, S) \in \mathcal{R}$.*

# Effect of Additional Requirement

- It does not hold:

$$\mu\mathbf{t}.\oplus\{l:\mathbf{t}\}\leq\mu\mathbf{t}.\&\{l':\oplus\{l:\mathbf{t}\}\}$$

- That is, types must be related without "orphan" messages
  - messages sent by a communicating partner that remain forever in the queue

# Orphan-message-free Subtyping

- Our alternative equivalent formulation :

**Definition** (Asynchronous Subtyping, $\leq$). $\mathcal{R}$ *is an asynchronous subtyping relation whenever* *it is* **dual closed** *and* $(T, S) \in \mathcal{R}$ *implies that:*

1. *if* $T = \mathbf{end}$ *then* $\exists n \geq 0$ *such that* $\mathsf{unfold}^n(S) = \mathbf{end}$;
2. *if* $T = \oplus\{l_i : T_i\}_{i \in I}$ *then* $\exists n \geq 0, \mathcal{A}$ *such that*
   - $\mathsf{unfold}^n(S) = \mathcal{A}[\oplus\{l_j : S_{kj}\}_{j \in J_k}]^{k \in \{1,\ldots,m\}}$,
   - $\forall k \in \{1, \ldots, m\}. I \subseteq J_k$ *and*
   - $\forall i \in I. (T_i, \mathcal{A}[S_{ki}]^{k \in \{1,\ldots,m\}}) \in \mathcal{R}$;
3. *if* $T = \&\{l_i : T_i\}_{i \in I}$ *then* $\exists n \geq 0$ *such that* $\mathsf{unfold}^n(S) = \&\{l_j : S_j\}_{j \in J}$, $J \subseteq I$ *and* $\forall j \in J. (T_j, S_j) \in \mathcal{R}$;
4. *if* $T = \mu\mathbf{t}.T'$ *then* $(T'\{T/\mathbf{t}\}, S) \in \mathcal{R}$.

# Dual type and Dual closeness

Given a session type $T$, its dual $\overline{T}$ is defined as:

- $\overline{\oplus\{l_i : T_i\}_{i\in I}} = \&\{l_i : \overline{T}_i\}_{i\in I}$,
- $\overline{\&\{l_i : T_i\}_{i\in I}} = \oplus\{l_i : \overline{T}_i\}_{i\in I}$,
- $\overline{\mathbf{end}} = \mathbf{end}$, $\overline{\mathbf{t}} = \mathbf{t}$, and
- $\overline{\mu\mathbf{t}.T} = \mu\mathbf{t}.\overline{T}$.

- Dual closeness:

relation $\mathcal{R}$ on session types is *dual closed* if $(S, T) \in \mathcal{R}$ implies $(\overline{T}, \overline{S}) \in \mathcal{R}$

# Conclusion: Impact of undecidability (not only session types)

- Consequences of our results:
  - Orphan-message-free asynchronous Session subtyping is also undecidable
  - Asynchronous Session subtyping for standard session types (with communication with carried types besides branching/selection) is undecidable
  - Asynchronous Multiparty Session subtyping is undecidable
  - Refinement over Communicating Automata/Behavioural Contracts is undecidable [BZ SOSYM21]

# The Hunt for Decidable Variants Continues…

- Investigation of other forms of restriction that allow us to obtain decidability, while retaining:
  - general branching for both inputs and outputs
  - queue unboundedness
- **Sound** algorithmic approximations based on characterizing looping accumulation patterns, e.g. [BCLYZ LMCS21] and [BLZ FOSSACS21] for **fair** asynchronous subtyping
- Decidability for **specific forms** of asynchronous communication used in practice?