Asynchronous Subtyping by Trace Relaxation

Laura Bocchi and Andy King

University of Kent
Canterbury, UK

Maurizio Murgia

Gran Sasso Science Institute
L'Aquila, Italy

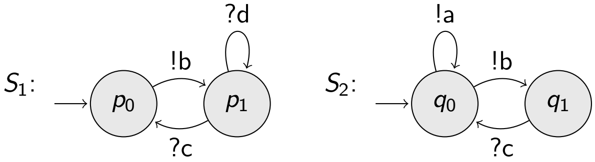
TACAS, 6-11 April 2024, Luxembourg City

Session subtyping

S_2 models a process, which in state p_0, either:

- sends a message a or
- sends a message b and then receives a message c

and does so repeatedly



S_1 can be safely substituted for S_2 because:

- S_1 has fewer sends (the absent $!a$) – co-variant
- S_1 has more receives (the additional $?d$) – contra-variant

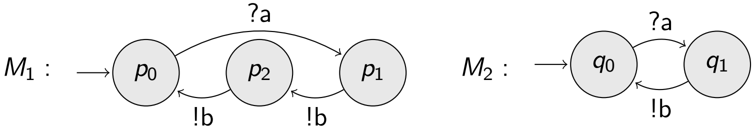
Write $S_1 \leq S_2$ iff a program with type S_1 can be safely substituted for a program with type S_2.

Why is subtyping useful?

- Check whether one component in a distributed system can be safety substituted with a patch
- Encourages a design methodology based on refinement
- Subtyping enables protocol optimisation in the order of sends and recieves are tweaked for improved performance

Why is subtyping tricky?

Consider M_2 which models a server producing a news feed (!b) on request from a client (?a):



After receiving on a, M_2 can mimic the first $!b$ of M_1, but it can only perform the second $!b$ after another $?a$ recieve

The input $?a$ is said to guard the output $!b$

One needs to reason about these dependencies to verify $M_1 \leq M_2$

Timeline for (very closely) related work

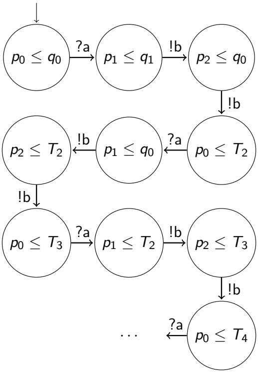
2005 S. Gay and M. Hole: Subtyping for Session Types in the Pi Calculus, Acta Informatica 42, 191–225 (2005).

2017 J. Lange and N. Yoshida: On the Undecidability of Asynchronous Session Subtyping, FoSSaCS, 441–457 (2017).

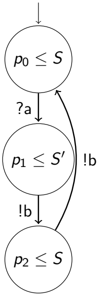
2021 M. Bravetti, M Carbone, J. Lange, N. Yoshida and G. Zavattaro: A Sound Algorithm for Asychronous Session Subtyping and its Implementation, LMCS 17(1), 1–35 (2021).

2021 S. Ghilezan, J. Pantovic, I. Prokic, A. Scalas and N. Yoshida: Precise Subtyping for Asynchronous Multiparty Sessions, POPL, 1–28 (2021).

From a simulation tree [LMCS'21] to a collecting simulation graph in a nutshell



$$T_2 = \langle a : q_0 \rangle$$
$$T_3 = \langle a : T_2 \rangle$$
$$T_4 = \langle a : T_3 \rangle$$
$$\vdots$$



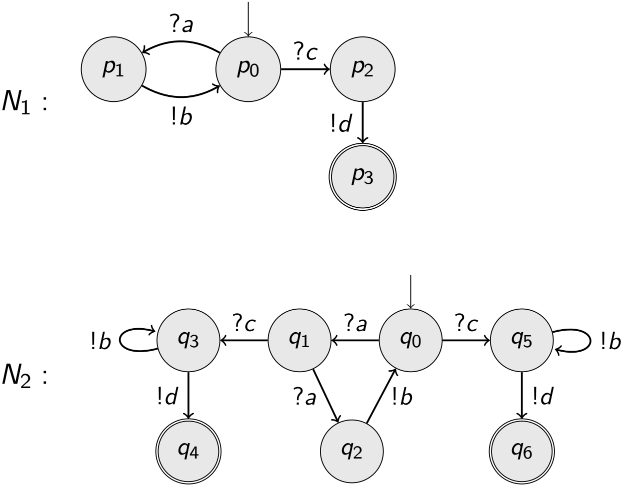
$$S = \bigcup_{i \geq 0} S_i$$
$$S' = S \cup \{q_1\}$$

where

$$S_0 = \{q_0\}$$
$$S_{i+1} = \{a \cdot \pi \mid \pi \in S_i\}$$

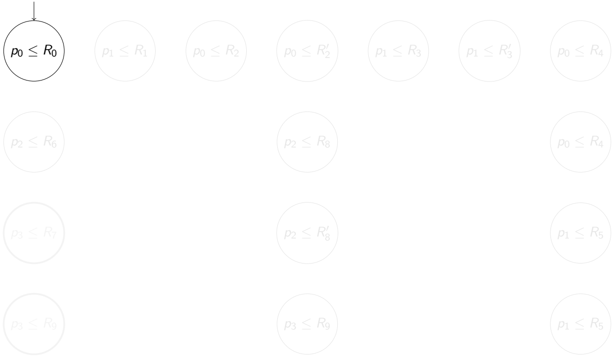
S and S' can be *finitely* represented by the regular strings $a^* q_0$ and $a^* q_0 + q_1$ respectively

Is $N_1 \leq N_2$?



Note that any cycle in N_1 passes through p_0; we put $wp = \{p_0\}$

step 0: A (collecting) simulation graph for proving $N_1 \leq N_2$



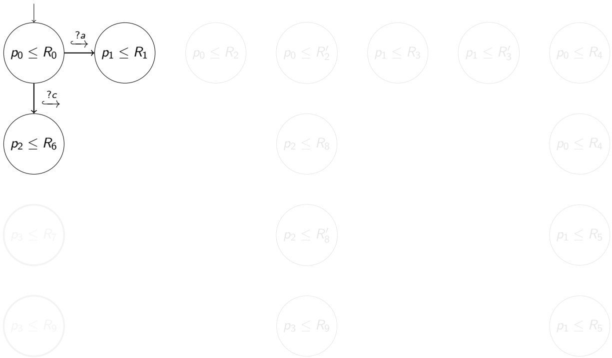
$$R_0 = \{q_0\}$$

Commentary on step 1

- N_1 receives at p_0 with $\text{in}_{N_1}(p_0) = \{a, c\}$
- Contra-variance of receive requires $\text{in}_{N_1}(p_0) \supseteq \text{in}_{N_2}(q_0)$
- But $\text{in}_{N_2}(q_0) = \{a, c\}$ so simulation proceeds with

$$p_0 \leq q_0 \xhookrightarrow{?a} p_1 \leq q_1 \text{ and } p_0 \leq q_0 \xhookrightarrow{?c} p_2 \leq q_5$$

step 1: A (collecting) simulation graph for proving $N_1 \leq N_2$



$$R_0 = \{q_0\}$$

$$R_1 = \{q_1\}$$

$$R_6 = \{q_5\}$$

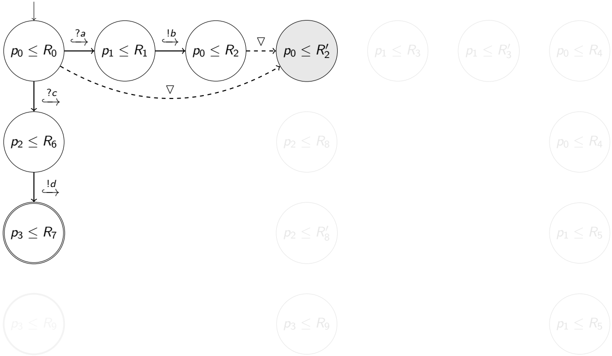
Commentary on step 2; just $p_1 \leq R_1$ where $R_1 = \{q_1\}$

- N_1 sends at p_1 with $\text{out}_{N_1}(p_1) = \{b\}$
- Co-variance of send requires $\text{out}_{N_1}(p_1) \subseteq \text{out}_{N_2}(q_i)$ for some q_i after q_1
- Two contendors for q_i are q_2 and q_3 because:
  - $q_1 \xrightarrow{?a} q_2$ and $\text{out}_{N_2}(q_2) = \{b\}$
  - $q_1 \xrightarrow{?c} q_3$ and $\text{out}_{N_2}(q_3) = \{b\}$
- But $q_2 \xrightarrow{!b} q_0$ and $q_3 \xrightarrow{!b} q_3$ so simulation proceeds with

$$p_1 \leq q_1 \xhookrightarrow{!b} p_0 \leq aq_0 \text{ and } p_1 \leq q_1 \xhookrightarrow{!b} p_0 \leq cq_3$$

- Can we simulate the send $!b$ and continue at q_0 (resp q_3) after a recieve $?a$ (resp. $?c$)

step 2: A (collecting) simulation graph for proving $N_1 \leq N_2$

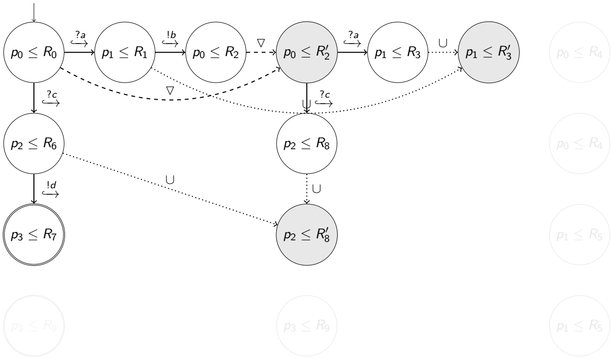


$R_0 = \{q_0\}$

$R_1 = \{q_1\}$
$R_6 = \{q_5\}$

$R_2 = \{aq_0, cq_3\}$
$R'_2 = \{a^* q_0, cq_3\}$
$R_7 = \{q_6\}$

step 3: A (collecting) simulation graph for proving $N_1 \leq N_2$



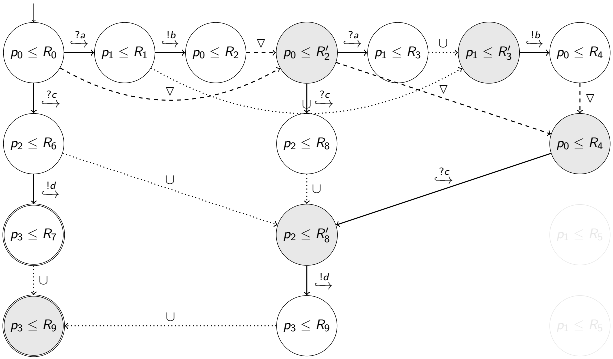
$R_0 = \{q_0\}$

$R_1 = \{q_1\}$
$R_6 = \{q_5\}$

$R_2 = \{aq_0, cq_3\}$
$R_2' = \{a^* q_0, cq_3\}$
$R_7 = \{q_6\}$

$R_3 = \{a^* q_0\}$
$R_3' = \{a^* q_0, q_1\}$
$R_8 = \{q_3\}$
$R_8' = \{q_3, q_5\}$

step 4: A (collecting) simulation graph for proving $N_1 \leq N_2$



$R_0 = \{q_0\}$

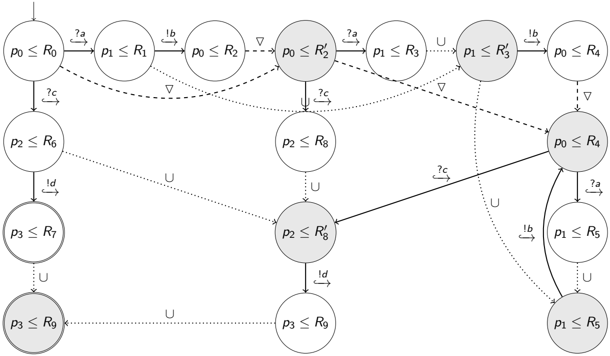
$R_1 = \{q_1\}$
$R_6 = \{q_5\}$

$R_2 = \{aq_0, cq_3\}$
$R_2' = \{a^* q_0, cq_3\}$
$R_7 = \{q_6\}$

$R_3 = \{a^* q_0\}$
$R_3' = \{a^* q_0, q_1\}$
$R_8 = \{q_3\}$
$R_8' = \{q_3, q_5\}$

$R_4 = \{a^* q_0, a^* cq_3, a^* cq_5\}$
$R_9 = \{q_4, q_6\}$

step 5: A (collecting) simulation graph for proving $N_1 \leq N_2$



$R_0 = \{q_0\}$

$R_1 = \{q_1\}$
$R_6 = \{q_5\}$

$R_2 = \{aq_0, cq_3\}$
$R'_2 = \{a^* q_0, cq_3\}$
$R_7 = \{q_6\}$

$R_3 = \{a^* q_0\}$
$R'_3 = \{a^* q_0, q_1\}$
$R_8 = \{q_3\}$
$R'_8 = \{q_3, q_5\}$

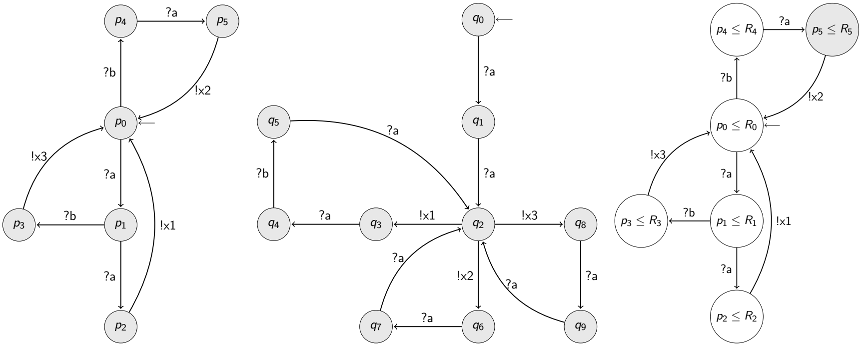
$R_4 = \{a^* q_0, a^* cq_3, a^* cq_5\}$
$R_9 = \{q_4, q_6\}$

$R_5 = \{a^* q_0, q_1, a^* cq_3, a^* cq_5\}$

Benchmarking

M_1	M_2	$\lvert M_1\rvert$	$\lvert M_2\rvert$	[LMCS'21]	regex	time
ctxta1	ctxta2	7	5	✗	✓	110
ctxtb1	ctxtb2	6	7	✗	✓	41
14may2 (N_1)	14may1 (N_2)	4	7	✗	✓	10
badseq1	badseq2	5	12	✗	✓	1127
march3testa1	march3testa2	6	7	✗	✓	222
aaaaaab1	aaaaaab2	5	3	✗	✓	43
ex1okloop	ex2okloop	10	8	✗	✓	1757
march3testa1	march3testb2	6	10	✗	✗	8

Post morten on march3testa1 $\leq$ march3testb2



Subtyping can be established by replacing

$$R_0 = \{q_0, \{a, b\}^* q_3, aq_8\} \text{ with } R_0 = \{q_0, Rq_3, Rq_6, Rq_8\}$$

where $R = (a^*(ba)^*a^*)^*$

Widening cannot infer strings with consecutive stared expressions

Complexity for $M_1 = (P, p_0, \delta_1)$ and $M_2 = (Q, q_0, \delta_2)$

function Subtype(M_1, M_2, Δ)
  forall ($p \in P$)
    if ($\Delta(p) \neq \emptyset \ \wedge \ p \leq \Delta(p) \not\rightarrow$) then return maybe
    else
      $R_p := \bigcup_{p' \in P} \{R \mid \exists \ell.\, p' \leq \Delta(p') \xhookrightarrow{\ell} p \leq R\}$
      $\Delta'(p) :=$ if ($p \in wp$) then $\Delta(p) \bigtriangledown R_p$ else $\Delta(p) \cup R_p$
    endif
  endfor
  if ($\Delta' \subseteq \Delta$) return Δ else return Subtype(M_1, M_2, Δ')
endfunction

The algorithm updates each state of P at most $(c|Q|)^{|wp|}$ times, updating Δ at most $|P|(c|Q|)^{|wp|}$ times, where c bounds the number of times a regular string can be relaxed

Conclusions

- We apply abstract interpretation to session subtyping to distil a more modular and more powerful checking algorithm
- Our approach is layered:
  - correctness is established with collecting sim trees;
  - collecting sim graphs accommodate trace relaxation;
  - traces are finitely represented by regular strings;
  - regular strings are finitely computed by widening
- This layering achieves modularity:
  - regular strings can be replaced with higher fidelity representations;
  - different widening techniques can be explored if required
- A certificate falls out of our subtyping algorithm