KIISE / SIGPL Summer School 2018

Aug. 20th (Day 1/3)

π-calculus, Intuitionistic Logic, and Compositional Verification

Ahn, Ki Yung

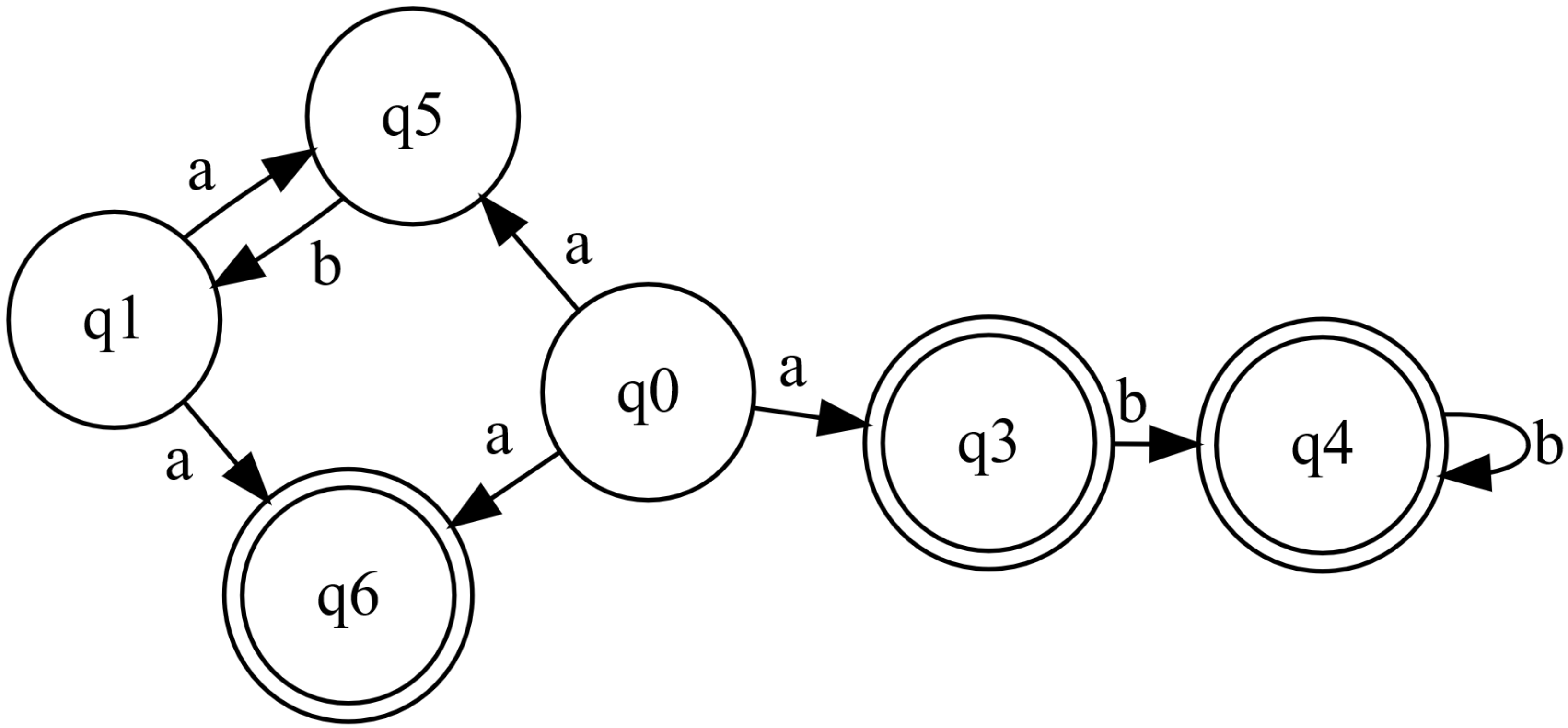
안기영(安基榮)

kya@hnu.kr

OUTLINE

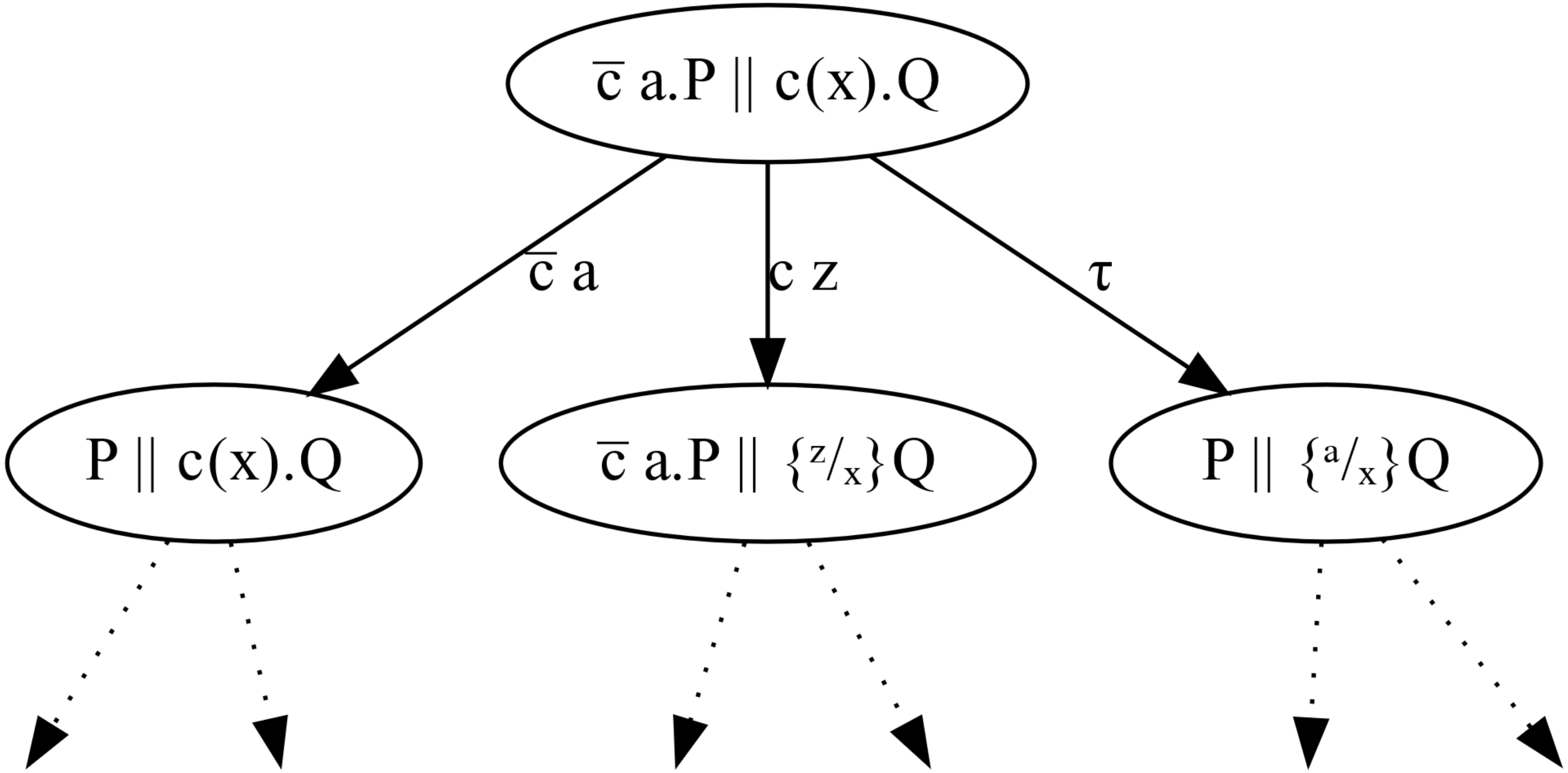
1. General Introduction
2. Specifying π-calculus Op. Sem. using λProlog
   - opam install elpi # assuming you have opam
   - https://bit.ly/2MDXjo4 (source files)
3. Formal Reasoning about π-calc. spec. in Abella
   - opam install abella
   - git clone https://github.com/abella-prover/PG
     cd PG && git checkout abella && make
     # assuming you have running emacs

INTRODUCTION



Labeled Transition System (LTS)

Nondeterministic Finite Automata



Labeled Transition System (LTS)

Nondeterministic

Infinite States

State = Process Term

$$
\begin{aligned}
P \ ::= \ & \bar{c}\, a\, .\, P \\
| \ & c(x).\, P \\
| \ & P \parallel P \\
| \ & \cdots
\end{aligned}
$$

π-calculus syntax

P ::=	0	stuck (no further action)
		$\bar{x}\ y\,.\,P$	output y on x then P
		$x(y).\,P$	bind y to input from x then P
		$\tau\,.\,P$	internal action then P
		$P_1 \parallel P_2$	parallel composition
		$P_1 + P_2$	nondeterministic choice
		$\nu z.\,P$	z is a fresh name
		$[x = y]\ P$	match (equality guard)
		$[x \neq y]\ P$	mismatch (inequality guard)
		$!\,P$	infinite parallel comp. of P

Sub-calculi of above

- Finite π-calculi
  - Finite π-calculus with Match only
  - Finite π-calc. with both Match and Mismatch

Syntactically Distinct but Equivalent π-calculus Processes

- $P \parallel 0 \;\sim\; 0 \parallel P \sim\; P \;\sim\; P + 0 \;\sim\; 0 + P$
- $P \;\sim\; [x = x]P$
- $0 \;\sim\; \nu z.[z = x]P$
- $\nu z.P \;\sim\; \nu z.[z \neq x]P$
- $\nu z.\tau.(P \parallel \{a/x\}Q) \;\sim\; \nu z.(\,\bar{z}a.P \parallel z(x).Q\,)$

Equivalent or Not?

- P
- $[x = y]P + [x \neq y]P$

Equivalences in π-calculus

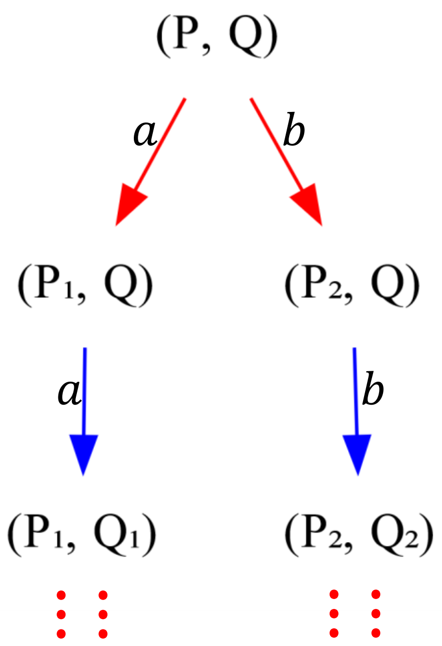
- Barbed Congruence/Equivalence
  - a natural obvservational equivalence
- Various Bisimulations
  - computatoinally effective

    (can write programs following the definitions)

Simulation and Bisimulation

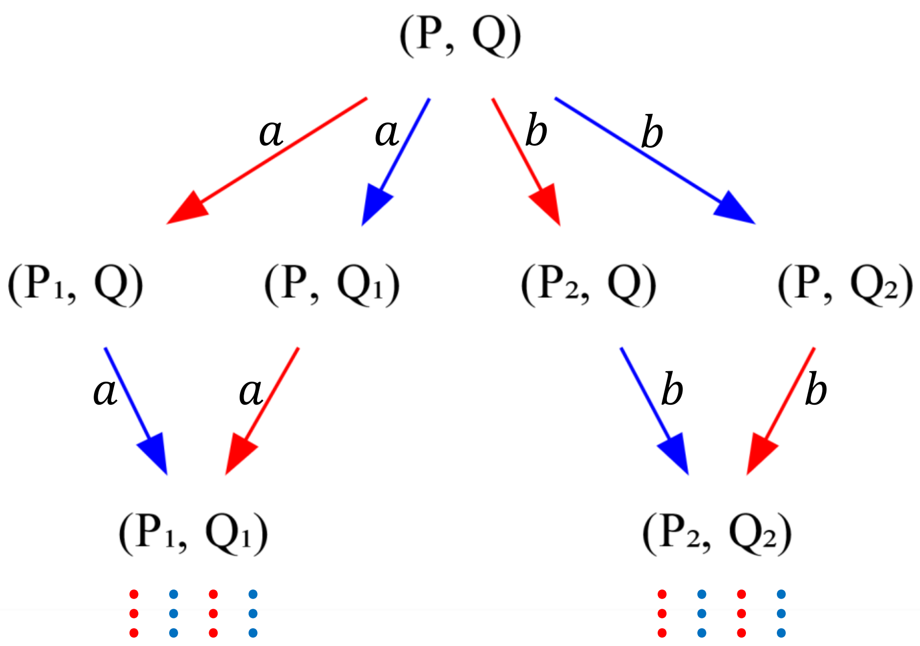
Q simulates P

- For every leading step from P there exists a following step from Q with the same label



P and Q are bisimilar

- For every leading step from any side there exists a following step from the other side with the same label



Equivalences in π-calculus

- Barbed Congruence/Equivalence
  - Equivalent processes have same barbs, i.e., $P \downarrow a$ iff $Q \downarrow a$ for any a
    - $P \downarrow a$ (P has barb a) when P can do input/output step on a
  - Equivalence relation is preserved under internal actions i.e., Let $\boldsymbol{R}$ be barbed eq. rel.; if $P\boldsymbol{R}Q$ then
    - for any $P \xrightarrow{\tau} P'$ there exists $Q \xrightarrow{\tau} Q'$ s.t. $P'\boldsymbol{R}Q'$
    - for any $Q \xrightarrow{\tau} Q'$ there exists $P \xrightarrow{\tau} P'$ s.t. $P'\boldsymbol{R}Q'$
  - Closed version patches up $\boldsymbol{R}$ afterwards to make it congruent
  - Open version additionally requires the definition $\boldsymbol{R}$ to be Contextual (close under all process contexts) at every step

Equivalences in π-calculus

Closed World / Classical Logic

NOT preserved under substitutions

NOT good for modular verification

- Barbed Congruence
- Bisimulation relations (variations on bindings of input variables)
  - **Early** Bisimilarity $\mathcal{FM}$
    - coincides with Barbed Cong.
  - **Late** Bisimilarity $\mathcal{LM}$
    - sub-relation of Early Bisimilarity

Open World / Intuitionistic Logic

Preserved under (respectful) **substitutions**

Good for modular verification

- Barbed Equivalence (open ver.)
- Bisimulation relations (variations on bindings of input variables)
  - $i\mathcal{FM}$ **Quasi-Open** Bisimilarity
    - coincides with Barbed Equiv.
  - $\mathcal{OM}$ **Open** Bisimilarity
    - sub-relation of Quasi-Open Bisim.

Modal Logics characterizing Bisimulations

HISTORY

- (Milner 1980) *A Calculus of Communicating Systems*
- (Hennessy and Milner 1980) *On Observing Nondeterminism and Concurrency*
  - Hennessy—Milner Logic
- (Milner, Parrow, and Walker 1992) *A Calculus of Mobile Processes (Part I, II)*
  - Early and Late bisimulations for the π-calculus with match only
  - Modal Logics categorizing finite π-calculus with match only
- (Sangiorgi 1996) *A Theory of Bisimulation for the π-Calculus*
  - Open bisimulation (for the π-calculus with match only)
- (Sangiorgi and Walker 2001) *On Barbed Equivalences in π-Calculus*
  - Quasi-Open bisimulation (with match only)
- (Ahn, Horne, and Tui 2017) *A Characterisation of Open Bisimilarity using an Intuitionistic Modal Logic*
- (Horne, Ahn, Lin and Tiu 2018) *Quasi-Open Bisimilarity with Mismatch is Intuitionistic*

Applied π-calculus

$$
\begin{aligned}
P \ ::= \ & 0 \\
| \ & \overline{M}\ N\,.\,P \\
| \ & M(x).\,P \\
| \ & \tau\,.\,P \\
| \ & P_1 \parallel P_2 \\
| \ & P_1 + P_2 \\
| \ & \nu z.\,P \\
| \ & [M = N]\ P \\
| \ & [M \neq N]\ P \\
| \ & !\,P
\end{aligned}
$$

- Richer term structure (M, N) not just names (x, y, z)
- E.g., symbolic crypto.

$$
\nu k.\nu z.\left(\begin{array}{l} \overline{z}(\mathsf{enc}(N,k)).P \parallel \\ z(x).[\,\mathsf{dec}(x,k) = N].Q \end{array}\right)
$$

Specifying π-calculus Operational Semantics using λProlog

Prolog vs. λProlog

Prolog

- Classical
- Predicates defined by First-order Horn clauses
- First-order Unification over untyped terms

λProlog

- Intuitionistic
- Predicates defined by Higher-order Hereditary Harrop formulae
- Higher-order Unification over simply-typed terms
  - i.e., unification over simply-typed HOAS (aka. λ-tree syntax)
  - modulo $\alpha\beta\eta$-equivalence

λProlog term syntax

- `x\y\x` means $\lambda x.\lambda y.x$
- `x\M x` means $\lambda x.M\ x$
  - `x` cannot appear free in `M` (including any instantiation of `M`) because the scope of `M` is larger than `x`.

π-calculus syntax

$$
\begin{aligned}
P ::= {} & 0 \\
| \ & \bar{x}\, y\,.\,P \\
| \ & x(y).\,P \\
| \ & \tau\,.\,P \\
| \ & P_1 \parallel P_2 \\
| \ & P_1 + P_2 \\
| \ & \nu z.\,P \\
| \ & [x = y]\,P \\
| \ & [x \neq y]\,P
\end{aligned}
$$

```
sig pic. %% file “pic.sig”

kind n   type. % name
kind p   type. % process
kind a   type. % label

type null p.
type out  n → n → p → p.
type inp  n → (n → p) → p.
type taup p → p.
type par  p → p → p.
type plus p → p → p.
type nu   (n → p) → p.
type mat  n → n → p.
type mis  n → n → p.
```

pic.sig (continued)

```
% constants for labels (actions)

type dn, up   n → n → a. % input, output
type tau              a. % internal action

% type sig for labelled transition relations

type one   p →       a   →       p   → o.
type oneb  p → (n → a) → (n → p) → o.
```

Transition Rules

```
module pic. %% file “pic.mod”
one (out X Y P) (up X Y) P.
one (inp X P)   (dn X Y) (P Y).   % P : n → p
one (taup P)    tau      P.
one (par P Q) A (par P1 Q) :- one P A P1.
one (par P Q) A (par P Q1) :- one Q A Q1.
one (par P Q) tau (par P1 Q1) :- one P (up X Y) P1,
                                 one Q (dn X Y) Q1.
one (par P Q) tau (par P1 Q1) :- one P (dn X Y) P1,
                                 one Q (up X Y) Q1.
one (plus P Q) A P1 :- one P A P1.
one (plus P Q) A Q1 :- one Q A Q1.
one (nu P) A (nu Q) :- pi x\ one (P x) A (Q x). % P,Q : n → p
one (mat X X P) A Q :- one P A Q.
```

Formal Reasoning about π-calculus Op.Sem. spec. in Abella

Related Tools (and we need more ...)

- Teyjus
  - De facto standard implementation of lambda-Prolog
  - Development frozen, annoying to install. (Github src fails to build with recent Ocaml)
  - Difficult to encode constructive inequality judgment for open terms
    (common problem in majority of logic programming languages including Prolog)
  - Difficult to encode bisimuation for open terms
- Bedwyr
  - Freshness, Nominal conditions natively supported via $\triangledown$ (nabla) quantifier
  - Easy to encode bisimulation for open terms
  - Automatic proof searching theorem prover (or model checker)
    Supporting sublanguage of Abella proof assistant
  - Still difficult to encode constructive inequality
- Abella
  - Not difficult to encode bisimulation of open terms with constructive inequality
  - Not an automatic solver