Type-Based Verification of Correspondence Assertions for Communication Protocols

Daisuke Kikuchi, Naoki Kobayashi
(Tohoku University)

Outline

- Introduction
- π^{CA}: π-Calculus with Correspondence Assertions
- Type and Effect System
- Type Checking Algorithm
- Related Work and Conclusion

Correspondence Assertions

- Formal notation for stating expected authenticity properties [Woo and Lam, '93]
  - Authenticity: Guarantee there are no falsification of messages and pretender of protocol users

Type and Effect system for checking correspondence assertions [Gordon and Jeffrey, '01] (GJ's type system)

- Advantage (over other verification methods)
  - Efficiency
- Disadvantage
  - Complicated type annotations

This Work

- Extension of GJ's type system with fractional effects
  - Polynomial-time type inference
  - More expressive power
- Proof of NP-hardness of GJ's type system (without type annotations)

Outline

- Introduction
- π^{CA}: π-Calculus with Correspondence Assertions
- Type and Effect System
- Type Checking Algorithm
- Related Work and Conclusion

Syntax of π^{CA}

- P ::= 0 — Inaction
  - | $x![\tilde{y}]$ — Output
  - | $x?[\tilde{y}].P$ — Input
  - | $(P_1|P_2)$ — Parallel composition
  - | *P — Replication
  - | $(\nu x)P$ — Name generation
  - | if x=y then P else Q — Conditional
  - | begin L.P — Begin-assertion
  - | end L.P — End-assertion

Event Labels: $<x_1,\ldots,x_n>$

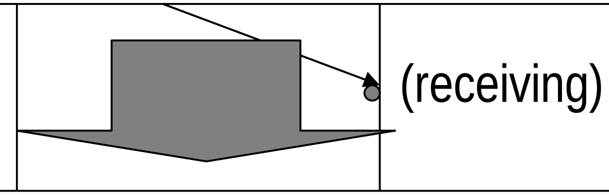
Example: Transmit-Acknowledge-Handshake protocol

(ν c)(Sender(a,b,c) |
Receiver(a,b,c))

Sender(a,b,c) $\triangleq$
(ν msg)(ν ack)
(c![msg,ack] |
ack?[])

Receiver(a,b,c) $\triangleq$
c?[m,r]. r![]

Whenever Sender *a* receives an acknowledgement for a message, Receiver *b* has already received it



On execution flow of the protocol, begin <a,b,msg> precedes the corresponding end <a,b,msg>

Correspondence Assertions

Safety

- Definition: A process *P* is *safe* if whenever an end-event occurs in *P*, the corresponding begin-event must have occurred before

- Example:
  - begin <x>.end <x> : *safe*
  - begin <x>.end <x>.end <x> : *unsafe*
  - begin <x>.begin <x>.end <x> : *safe*

Outline

- Introduction
- π^{CA}: π-Calculus with Correspondence Assertions
- Type and Effect System
- Type Checking Algorithm
- Related Work and Conclusion

Key Idea of Type System

- Extend channel types with information about capabilities for raising end-events

  - Example:
    - Ch()[<x>↦1]
      a channel used for passing a unit value and the capability for raising one *end* <*x*> event
    - Ch(Name)[<1>↦2]
      a channel used for passing a name value and the capability for raising two end-events on the name

Syntax of Types and Effects

- $T ::= \text{Name}$ Name Type
  $\mid Ch(T_1,\ldots,T_n)e$ Channel Type
  (e: capabilities passed through the channel)
- $e ::= [L_1 \mapsto q_1,\ldots,L_n \mapsto q_n]$ Fractional Effects
  ($q_1, \ldots, q_n$: non-negative rational numbers)
- $L ::= <\boldsymbol{\alpha}_1,\ldots,\boldsymbol{\alpha}_k>$ Extended Event Labels
- $\boldsymbol{\alpha} ::= x \mid \boldsymbol{\iota}$ Extended Names
- $\boldsymbol{\iota} ::= \uparrow \boldsymbol{\iota} \mid 1 \mid 2 \mid \ldots$ Indices

Type Judgment

Assumptions about how the names may be used

$$\Gamma \vdash P : e$$

Capabilities of end-events that P may raise

- Example:
  - $x{:}\mathrm{Name} \vdash \mathrm{begin}\ <x>.\mathrm{end}\ <x>:[]$
  - $x{:}\mathrm{Ch}(\mathrm{Name})[<1> \mapsto 1] \nvdash x?[y].\mathrm{end}\ <y>.\mathrm{end}\ <y>:[]$
  - $x{:}\mathrm{Ch}()[<y> \mapsto 0.5] \vdash x?[].x?[].\mathrm{end}\ <y>:[]$

Difference from GJ's type system

	GJ's type system	Our type system
Effects	Mapping from event labels to natural numbers	Mapping from event labels to rational numbers
Type annotations	Explicit	Implicit
Channel Type Representation	Name Based Ch(x:Name)[<x>]	Index Based Ch(Name)[<1>↦1]

Typing Rules

$$\frac{\Gamma \vdash P : e + [L \mapsto 1] \quad N(L) \subseteq \mathrm{dom}(\Gamma)}{\Gamma \vdash \mathrm{begin}\ L.P : e} \,(\mathrm{T-Begin})$$

$$\frac{\Gamma \vdash P : e \quad N(L) \subseteq \mathrm{dom}(\Gamma)}{\Gamma \vdash \mathrm{end}\ L.P : e + [L \mapsto 1]} \,(\mathrm{T-End})$$

Typing Rules

$$\frac{\Gamma \vdash x : Ch(T)e \quad \Gamma \vdash y : [y/\uparrow 1]T}{\Gamma \vdash x![y] : [y/1]e} \;(T-Out)$$

$$\frac{\begin{array}{c}\Gamma \vdash x : Ch(T)e_1 \quad \Gamma, y : T' \vdash P : e_2 \\ T' = [y/\uparrow 1]T \quad e + [y/1]e_1 \geq e_2\end{array}}{\Gamma \vdash x?[y].P : e} \;(T-In)$$

Type Soundness

- Theorem:
  If $\Gamma \vdash P:[]$, then P is *safe*

- Proof:
  Essentially the same as the proof of the type soundness theorem of GJ's type system

Comparison with GJ's Type System

- Our type system is *strictly* more expressive than GJ's type system

Example:

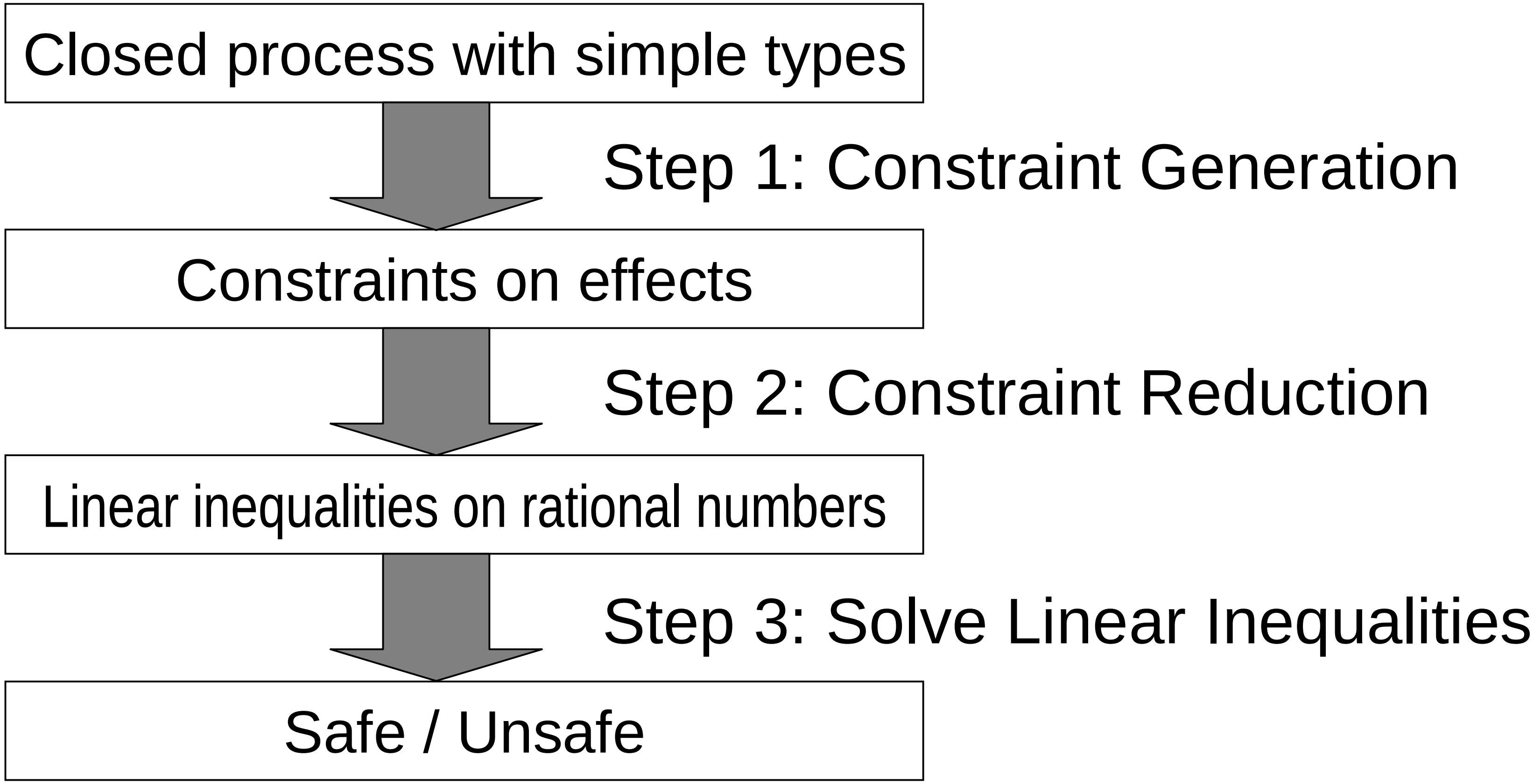
$P \triangleq$ (begin <a>.(c![] | c![])) | (c?[].c?[].end <a>)

- Typable in our type system
  - $c{:}Ch()[\langle a\rangle \mapsto 0.5] \vdash P{:}[]$
- Untypable in GJ's type system

Outline

- Introduction
- π^{CA}: π-Calculus with Correspondence Assertions
- Type and Effect System
- Type Checking Algorithm
- Related Work and Conclusion

Type Checking Algorithm



Example: Transmit-Acknowledge-Handshake protocol

$(\nu$ c:Ch(Name,Ch() $\rho_0) \rho_c)$
(Sender(a:Name,b:Name,
c:Ch(Name,Ch() $\rho_0) \rho_c)$ | ...) : ρ_{sys}

Sender(a:Name,b:Name,
c:Ch(Name,Ch() $\rho_0) \rho_c)$: $\rho_s \triangleq$
$(\nu$ msg:Name)(ν ack:Ch() $\rho_{ack})$
(c![msg:Name,ack:Ch() ρ_{ack}] |
ack?[].end <a,b,msg>)

……

Step 1: Constraint Generation

- Sender(a:Name,b:Name, c:Ch(Name,Ch() ρ_0) ρ_c): $\rho_s \triangleq$
  (ν msg:Name)
  (ν ack:Ch() ρ_{ack})
  (c![msg:Name,ack:Ch() ρ_{ack}]
  |
  ack?[].end $<a,b,msg>$)

$\rho_s \geqq \rho_{c!} + \rho_{ack?}$

$\rho_{ack} = [msg/\uparrow 1]\,\rho_0$

$\rho_{c!} \geqq [msg/1, ack/2]\,\rho_c$

$\rho_{ack?} + \rho_{ack} \geqq [<a,b,msg> \mapsto 1]$

Step 2: Constraint Reduction

- Relevant events:
  $L = \{\langle x_1, x_2, x_3\rangle \mid x_1, x_2, x_3 \in \{a, b, c, msg, ack, 1, 2, \uparrow 1, \ldots\}\}$
  $= \{L_1, \ldots, L_m\}$

- Conversion of inequalities:
  $\rho_1 \geqq \rho_2 + \rho_3$

  $\zeta_{1,\langle a,b,msg\rangle} \geqq \zeta_{2,\langle a,b,msg\rangle} + \zeta_{3,\langle a,b,msg\rangle},$
  $\zeta_{1,\langle a,c,ack\rangle} \geqq \zeta_{2,\langle a,c,ack\rangle} + \zeta_{3,\langle a,c,ack\rangle}, \cdots\cdots$
  $(\rho_i = \{L_1 \mapsto \zeta_{i,1}, \ldots, L_m \mapsto \zeta_{i,m}\})$

Step 3: Solve Linear Inequalities

- Inequalities:

  $\zeta_{c!,<a,b,msg>} \geqq \zeta_{c,<a,b,\uparrow 1>}$,
  $\zeta_{ack?,<a,b,msg>} + \zeta_{ack,<a,b,msg>} \geqq 1$,
  $\zeta_{s,<a,b,msg>} \geqq \zeta_{c!,<a,b,msg>} + \zeta_{ack?,<a,b,msg>}$,
  $\zeta_{ack,<a,b,msg>} = \zeta_{0,<a,b,\uparrow 1>}$, ……

- Solutions:

  $\zeta_{0,<a,b,\uparrow 1>} = \ldots = \zeta_{ack,<a,b,msg>} = 1$,
  $\zeta_{c!,<a,b,msg>} = \zeta_{ack?,<a,b,msg>} = \ldots = \zeta_{s,<a,b,msg>} = 0$,
  ……

ack:Ch()[<a,b,msg>↦1]

Sender(a,b,c):[]

Efficiency of the Algorithm

- Assumption: Both the size of simple types and that of event labels are bounded by a constant

Step 1: polynomial in the size of input *P*

Step 2: polynomial in the size of constraints

Step 3: polynomial in the size of linear inequalities

⇓

The whole procedure runs in time polynomial in |*P*|

Complexity of GJ's Type System

- The typability in GJ's type system is NP-hard without type annotations
- Proof:
  Reduction of 3-SAT problem into the type-checking problem in GJ's type system

Outline

- Introduction
- π^{CA}: π-Calculus with Correspondence Assertions
- Type and Effect System
- Type Checking Algorithm
- Related Work and Conclusion

Related Work

- Extended type and effect system [Gordon and Jeffrey, ’01-’03]
  - Verify authenticity of cryptographic protocols in spi-calculus
- Fractional effects [Boyland, ‘03][Terauchi and Aiken, ‘06]
  - Prevent interference of read/write operations on reference cells or channels

Conclusion

- Extended Gordon and Jeffrey’s type system for checking correspondence assertions
  - Fractional effects for polynomial-time type inference and more expressive power
- Proved NP-hardness of GJ’s type system (without type annotations)

Future work

- Extension of the type system to deal with cryptographic primitives

- Implementation of a protocol verification tool

Fin.

Thank you for listening to my presentation

Index Constructors

- Example:
  - c:Ch(Name,Ch()[<↑1>↦1])[<1>↦1]

    corresponds to
  - c’:Ch(x:Name,y:Ch()[<x>])[<x>]

    in GJ’s type system