SE547: Lecture 5

Overview

Types for protocols

Types for secrecy

Cryptyc

Types for authenticity

Types for protocols

What is a type system? Why?

What do the types for the lambda-calculus look like?

What might types for the pi-calculus look like?

What might types for the spi-calculus look like?

Types for protocols

History:

Pi-calculus sorting (Milner)

Pi-calculus typing (Pierce and Sangiorgi, ...)

Spi-calculus types for secrecy (Abadi and Blanchet)

Spi-calculus types for authenticity (Gordon and Jeffrey)

Types for protocols

Take the pi-calculus :

P, Q, R ::=
- $\mathbf{0}$
- out $x\ y; P$
- in $x\ (y); P$
- $P \mid Q$
- $!P$
- new $(x); P$

Types for protocols

Add lists of messags as primitives:

P, Q, R ::=
- **0**
- out $M\ N; P$
- in $M\ (y); P$
- $P \mid Q$
- $!P$
- new $(x); P$
- split L is $(x, y); P$

L, M, N ::=
- x
- $()$
- (M,N)

Types for protocols

Allow bad states:

P, Q, R ::=
...
FAIL

with dynamic semantics:

split (L) is (x,y); $P \rightarrow$ FAIL (L not of the form (M,N))
out L N; $P \rightarrow$ FAIL (L not of the form c)
in L (x); $P \rightarrow$ FAIL (L not of the form c)

A process is *safe* if there is no Q such that $P \rightarrow^* Q \mid$ FAIL.

Types for protocols

Are the following safe?

1. out c d; **0**
2. out c (); **0**
3. out () d; **0**
4. out () (); **0**
5. in c (x); out x (); **0**
6. in c (x); out x (); **0** | out c (d); **0**
7. in c (x); out x (); **0** | out c (); **0**

Types for protocols

Add types:

S, T, U ::=
chan (T)
()
(T,U)

(Recall Γ is a lookup table of types for variables x:T.)

Rules for messages saying when $\Gamma \vdash M : T$ is true:

1. If x:$T \in \Gamma$ then $\Gamma \vdash x : T$
2. $\Gamma \vdash () : ()$
3. If $\Gamma \vdash M : T$ and $\Gamma \vdash N : U$ then $\Gamma \vdash (M,N) : (T,U)$

What are the matching type rules for processes saying when $\Gamma \vdash P$: ok is true?

Types for protocols

Can we find types for c and d which make the following ok?

1. out c d; **0**
2. out c (); **0**
3. out () d; **0**
4. out () (); **0**
5. in c (x); out x (); **0**
6. in c (x); out x (); **0** | out c (d); **0**
7. in c (x); out x (); **0** | out c (); **0**

Types for protocols

Results:

1. Type safety: if $\Gamma \vdash P : \text{ok}$ then $\text{FAIL} \notin P$.
2. Subject reduction: if $\Gamma \vdash P : \text{ok}$ and $P \to Q$ then $\Gamma \vdash Q : \text{ok}$

Plug these together and we get that well-typed processes are safe!

Types for protocols

Example RPC call:

!in *rpc* (*return*, *arg*); ... out *return result*; **0**
new (*return*); out *rpc* (*return*, *arg*); in *return* (*result*); ...

What types do we give *rpc*, *return* to get this to typecheck?

Assume *arg*:*Arg* and *result*:*Result*.

Types for protocols

Add message tags:

P, Q, R ::=
...
case L is inl(x) P is inr(y) Q

L, M, N ::=
...
inl (M)
inr (M)

Dynamic semantics:

case L is inl(x) P is inr(y) $Q \rightarrow$ FAIL
(L not of the form inr(M) or inr(N))

Types for protocols

Add types for message tags:

$$
\begin{array}{l}
S, T, U ::= \\
\quad \ldots \\
\quad T + U
\end{array}
$$

1. If $\Gamma \vdash M : T$ then $\Gamma \vdash \text{inl}\,(M) : T+U$
2. If $\Gamma \vdash N : U$ then $\Gamma \vdash \text{inr}\,(N) : T+U$

What is the matching rule for case?

Types for protocols

What types can we give to c, d, e, x, y and z here:

Example | out c (inl(*fred*)); out c (inr(*wilma*)); **0**

where:

Example =
  !in c (x);
  case x is inl(y)
    out d (y); **0**
  is inr(z)
    out e (z); **0**

(Assume *fred*:*Boy* and *wilma*:*Girl*.)

Types for secrecy

Types for spi are different from types for pi, because:

1. We care about dishonest agents (not just honest agents).
2. We care about *robust* safety, not just safety.
3. Dishonest agents don't play by the type rules.

The type system for pi does *not* give us robust safety, because of attackers who don't play by the type rules.

Example =
  !in c (x);
  case x is inl(y)
    out d (y); **0**
  is inr(z)
    out e (z); **0**

This process is safe, but not robustly safe. (What is the attacker?)

Types for secrecy

Introduce a new untrusted type for the dishonest agents:

S, T, U ::=
...
Un

satisfying *opponent typability*:

1. If x:Un $\in \Gamma$ for every $x \in M$ then $\Gamma \vdash M$: Un.
2. If x:Un $\in \Gamma$ for every $x \in P$ then $\Gamma \vdash P$: ok

We switch off the type system when we hit type Un.

In particular, $\vdash$ FAIL : ok, unsurprising, since attackers don't play ball with types!

Types for secrecy

Introduce a new trusted type of private secrets:

S, T, U ::=
...
Private

In particular, we have:

net : Un, s : Private $\not\vdash$ out net (s); **0** : ok

Types for secrecy

We use crypto to keep secrets!

P, Q, R ::=
  ...
  decrypt M is $\{ x \}_N$; P

L, M, N ::=
  ...
  $\{ M \}_N$

Is this example robustly safe for secrecy of s?

Example_1 = new (key); !Sender | !Receiver
Sender = out net $\{ s \}_{key}$; **0**
Receiver = in net $\{ x \}_{key}$; **0**

Types for secrecy

We use crypto to keep secrets!

P, Q, R ::=
  ...
  decrypt M is $\{ x \}_N$; P

L, M, N ::=
  ...
  $\{ M \}_N$

Is this example robustly safe for secrecy of s?

Example_2 = new (key); !Sender | !Receiver | !Oracle
Sender = out net $\{ s \}_{key}$; **0**
Receiver = in net $\{ x \}_{key}$; **0**
Oracle = in net $\{ x \}_{key}$; out net (x); **0**

Types for secrecy

Types for crypto:

S, T, U ::=
...
Key (T)

Typing encryption:

1. If $\Gamma \vdash M : T$ and $\Gamma \vdash N : \text{Key}\ (T)$ then $\Gamma \vdash \{\ M\ \}_N : \text{Un}$.

What is the matching rule for decrypt?

Types for secrecy

What types do we give for s, x and key here:

Example = new (key); !Sender | !Receiver | !Oracle
Sender = out net { s }$_{key}$; **0**
Receiver = in net { x }$_{key}$; **0**

Why doens't this typecheck:

Oracle = in net { x }$_{key}$; out net (x); **0**

Types for secrecy

Result:

> If net : Un, s : Private $\vdash P$: ok
> then P is robustly safe for secrecy of s.

Cryptyc

Cryptyc (Gordon and Jeffrey) is a spi-calculus based protocol verification tool, based on type-checking.

To run cryptyc:

java -jar cryptyc.jar *filename*

for example:

java -jar cryptyc.jar secrecy.cry

produces output:

Type checked OK!

but if you uncomment the oracle, you get:

Type error!

Cryptyc

Cryptyc syntax is like the pi-calculus, but it uses a client-server model.

This is just syntax sugar:

server *S* at *A* is (*socket*:*Socket*) { *P* }
client *C* at *B* is { establish *S* at *A* is (*socket* : *Socket*); *Q* }

translates into pi as:

!input *S*@*A* (*socket*:*Socket*); *P* |
new (*socket* : *Socket*); output *S*@*A* (*socket*); *Q*

The action is in the types, for example:

type MyKey = Key (Private);

says a lot about this protocol!

Cryptyc also supports Struct and Union types.

Types for authenticity

Authenticity is declared using *correspondences*:

> A begins (Sender sent M)
> (1) $A \rightarrow B$: $\{ M \} Kab$
> B ends (Sender sent M)

Recall, this protocol is *not* robustly safe. (Why not?)

Example is coded in Cryptpyc as fail-auth.cry.

Solution: nonce challenges!

Types for authenticity

We need to track the *effect* of a process: these are the ends without matching begins.

What are the effect of the following:

1. begin *hello*; end *hello*; **0**
2. begin *hello*; end *world*; **0**
3. end *hello*; end *world*; **0**
4. end *hello*; begin *hello*; **0**

Types for authenticity

Calculating the effect for straight-line code is easy:

es, fs, gs ::=
  end L_1, ..., end L_n

1. If $\Gamma \vdash P$: [es] then $\Gamma \vdash$ end L; P : [es + end L]
2. If $\Gamma \vdash P$: [es] then $\Gamma \vdash$ begin L; P : [es - end L]
3. $\Gamma \vdash \mathbf{0}$: [ ]

Which of these have empty effect?

1. begin *hello*; end *hello*; **0**
2. begin *hello*; end *world*; **0**
3. end *hello*; end *world*; **0**
4. end *hello*; begin *hello*; **0**

Revisit fail-auth.cry.

Types for authenticity

Types for nonces (simplified: the real thing makes sure each nonce is only used once):

S, T, U ::=
...
Nonce (es)

with:

1. If $\Gamma \vdash M$: Un and $\Gamma \vdash N$: Nonce (es) and $\Gamma \vdash P$: [fs]
   then $\Gamma \vdash$ check M is N; P : [fs - es]

Still doesn't quite work: fail-simple2.cry.

Types for authenticity

Need to allow nonces to start at type Un, then later on be converted to Nonce type:

P, Q, R ::=
...
cast N is (x : Nonce (es)); P

at run-time this is a no-op:

cast N is (x : Nonce (es)); $P \rightarrow P[N/x]$

but when we type-check, we make sure we can justify the effect!

1. If $\Gamma \vdash N$: Un and Γ, x:Nonce(es) $\vdash P$: [fs]
   then $\Gamma \vdash$ cast N is (x:Nonce(es)); P : [fs + es]

Hooray, now it works: simple.cry.

Types for authenticity

Result:

> If $net : \mathrm{Un} \vdash P : [\,]$
> then P is robustly safe for authenticity.

Usual collection of examples is bundled with the Cryptyc tool...

Next week

Midterm.