Proyecto Final Verificación Formal

Session Type Systems - Mecanización

Ciro Iván García López

Febrero 2021

Contenidos

- Introducción.
- Estructura del proyecto.
- Proposiciones.
- Procesos.
- Tipos.

Introducción

En el presente documento se exponen los avances del primer proyecto del curso de verificación formal. El objetivo de este proyecto es mecanizar el articulo de Bas van den Heuvel y Jorge Pérez Session Type Systems based on Linear Logic en Coq.

Estructura del proyecto

La mecanización consta de tres partes fundamentales:

- Proposiciones.
- Procesos.
- Tipos.

Cada una de estas partes, a su vez, se divide en definiciones y proposiciones.

Proposiciones

Definición

Las proposiciones de ULL son generados por la siguiente gramática:

$$A, B := 1 \mid \bot \mid A \otimes B \mid A \parr B \mid A \multimap B \mid !A \mid ?A \tag{1}$$

```
Inductive Proposition : Type :=
  | ONE : Proposition
  | ABS : Proposition
  | TEN (A : Proposition) (B : Proposition) : Proposition
  | PAR (A : Proposition) (B : Proposition) : Proposition
  | EXP (A : Proposition) : Proposition
  | MOD (A : Proposition) : Proposition.
Hint Constructors Proposition : core.
```

Proposiciones

Se define el operador *x* a partir de los otros.

```
Definition ULLT_IMP (A : Proposition) (B : Proposition) : Proposition := (A^⊥) ⅋ B.
```

También se define la noción de dual.

```
Fixpoint Dual_Prop ( T : Proposition ) : Proposition
```

Procesos

Definición

Un proceso está generado por la siguiente gramática:

$$P := \mathbf{o} \mid [x \leftrightarrow y] \mid (\nu y)P \mid P|Q \mid x\langle y\rangle.P \mid x(y).P \mid !x(y).P \mid x\langle\rangle.\mathbf{o} \mid x().P \tag{2}$$

Se diseña una representación libre de nombres para los procesos siguiendo las ideas expuestas por Charguéraud [Cha12].

$$N := FName(x) \mid BName(i)$$
$$P := \mathbf{o} \mid [N \leftrightarrow N] \mid \nu P \mid P|Q \mid N\langle N\rangle.P \mid N.P \mid !N.P \mid N\langle\rangle.\mathbf{o} \mid N().P$$

Procesos

No todo es un proceso, por ejemplo $[BVar(0) \leftrightarrow BVar(1)]$.

Definición

Un *Process* es un término que se construye bajo las 10 reglas definidas.

$$\frac{Process(FVar(x)) \qquad Process(FVar(y))}{Process([FVar(x) \leftrightarrow FVar(y)])}$$

$$\frac{\exists L\ \forall\, x \notin L \qquad FVar(x) \qquad Process(P^x)}{Process(FVar(x).P)}$$

$$\frac{Process(FVar(x))}{Process(FVar(x)\langle\rangle, 0)}$$

$$\frac{\exists L\ \forall\, x \notin L \qquad FVar(x) \qquad Process(P^x)}{Process(!FVar(x).P)}$$

Procesos - Body

```
Inductive Process_Name : Name → Prop

Inductive Process : Prepro → Prop
```

Una de las definiciones *nuevas* más importantes de la mecanización.

```
Inductive Body : Prepro → Prop :=
  is_body : forall (P : Prepro), (forall (x : Name)(L : list Name),
  Process_Name x → ~ (In x L) → Process ( ({ 0 ~> x }P) )) → Body(P).
```

Procesos - Apertura y cerradura

```
1  Definition Open_Name ( k : nat )(z N : Name ) : Name :=
2  match N with
3    | FName x ⇒ FName x
4    | BName i ⇒ if ( k =? i ) then z else (BName i)
5  end.
6
7  Fixpoint Open_Rec (k : nat)( z : Name )( T : Prepro ) {struct T} : Prepro
8
9  Definition Close_Name ( k : nat )( z N : Name ) : Name :=
10
11 Fixpoint Close_Rec (k : nat)( z : Name )( T : Prepro ) {struct T} : Prepro
```

Procesos - Congruencia y reducción

```
Reserved Notation "R '= = =' S" (at level 60).
Inductive Congruence : Prepro → Prepro → Prop :=
...
| Con_abs_restriction : forall (P Q : Prepro),
      Process P → Body Q → (P ↓ (ν Q)) = = = ν (P ↓ Q)
...

Reserved Notation "R '-- >' S" (at level 60).
Inductive Reduction : Prepro → Prepro → Prop :=
...
| Red_parallel_fuse : forall (x y : Name) (P : Prepro),
   Process P → ((P ↓ [x ↔ y]) -→ (Subst x y P))
...
```

Congruencias y Reducciones

Se debe verificar que las congruencias y reducciones están bien definidas.

```
Theorem Congruence_WD :
forall P Q : Prepro,
(P = = = Q) → Process(P) → Process(Q).

Theorem ProcessReduction_WD :
forall P Q : Prepro,
(P -→ Q) → Process(P) → Process(Q).
```

Procesos

Es necesario suponer los siguientes axiomas.

```
Axiom Ax_Alpha :
forall (x y: Name)( P : Prepro),
({ y \ x} P ) = P.

Axiom Ax_Process_Name :
forall ( x : Name),
Process_Name x.
```

Tipos

Se definen algunas nociones para llegar a los tipos.

```
1  Inductive Assignment : Type
2
3  Inductive Assig : Assignment → Prop
4
5  Inductive Collect : list Assignment → Prop
```

Tipos

Se definen las reglas de inferencia para los tipos.

```
Reserved Notation "D ';;;' F '!-' P ':::' G" (at level 60).
Inductive Inference : Prepro → list Assignment → list Assignment → list Assign
Prop
...
  | reccr : forall ( D F G: list Assignment ) ( y x : Name)( A B : Proposition)(P : Prepro ),
    Collect D → Collect F → Collect G → Process_Name x → Process_Name y → Proc
    ( D ;;;  F !- P :::  ( (cons (x:B) (cons (y:A) nil) ) ++ G ) ) →
    ( D ;;;  F !- (x . (Close y P)) ::: (( cons (x:(A⅋B) ) nil) ++ G ) )
...
where "D ';;;' F '!-' P ':::' G" := (Inference P D F G).
```

Soundness

La propiedad clave del artículo.

```
Theorem Soundness :
forall (P : Prepro)(D F G : list Assignment),
  ( D ;;;  F !- P ::: G ) →
  (forall (Q: Prepro), (P -→ Q) → ( D ;;; F !- Q ::: G )).
Proof.
...
Qed.
```

Se divide la prueba en dos casos, reglas de corte y de no corte.

Regla de no corte

Para las reglas de no corte se prueba la imposibilidad en la reducción.

```
Lemma No_Red_AX4 :
forall (x : Name)(P Q: Prepro),
~( (x . P) --> Q ).
Proof.
  unfold not.
  intros.
  inversion H.
Qed.
```

Reglas de corte

Se analiza en búsqueda de las posibles reducciones y se aplica la regla de corte.

$$\frac{\Gamma; . \vdash P :: (x : A) \qquad \Gamma, (u : A); \Delta \vdash Q :: \Lambda}{\Gamma; \Delta \vdash (\nu u)(!u(x).P|Q) :: \Lambda}$$

```
Lemma Char_Red_Chanres2 :
forall (P Q : Prepro)(x : Name),
Process P → (ν (Close x P) -→  Q ) →
exists (Q0 : Prepro), ( Q = (ν (Close x Q0)) ∧ P -→  Q0).
Proof.
...
Qed.
```

Conclusiones

- El nivel de formalidad con el que se expresan los resultados del cálculo π en la mayoría de artículos y textos [SW03, vdHP20, Hon93] hace de la mecanización una labor extensa, ya que es necesario capturar la esencia de esta baja formalidad y en algunos casos conlleva plantear marcos de trabajo lo suficientemente amplios.
- La representación libre de nombres brinda beneficios a la hora de trabajar con variables ligadas, es sencillo expresar los términos bajo la representación y sus resultados son naturales. No obstante en la práctica se evidencia que no suprime la necesidad de las α equivalencias, lo cual implica asumir algunos resultados no deseados.

David Castro, Francisco Ferreira, and Nobuko Yoshida.
Emtst: Engineering the meta-theory of session types.
In Armin Biere and David Parker, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 278–285. Springer International Publishing, 2020.

Arthur Charguéraud.
The locally nameless representation.
Journal of Automated Reasoning - JAR, 49:1–46, 10 2012.

Luís Caires, Frank Pfenning, and Bernardo Toninho.
Towards concurrent type theory.
Conference Record of the Annual ACM Symposium on Principles of Programming Languages, 01 2012.

Jean-Yves Girard.
Linear logic: A survey.
In Friedrich L. Bauer, Wilfried Brauer, and Helmut Schwichtenberg, editors, *Logic and Algebra of Specification*, pages 63–112, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.


Kohei Honda.
Types for dyadic interaction.
In Eike Best, editor, *CONCUR'93*, pages 509–523, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.


D. Sangiorgi and D. Walker.
The Pi-Calculus: A Theory of Mobile Processes.
Cambridge University Press, 2003.


Bas van den Heuvel and Jorge A. Pérez.
Session type systems based on linear logic: Classical versus intuitionistic.
Electronic Proceedings in Theoretical Computer Science, 314:1–11, Apr 2020.


Uma Zalakain.
Type-checking session-typed π-calculus with coq.
University of Glasgow, 2019.