

Finales PLP

Finales	2
03/08/2020 - Melgratti/Garbervetsky - Virtual	2
13/07/2020 - Melgratti/Garbervetsky - Virtual	4
18/09/2019 - Melgratti	6
19/12/2018 - Melgratti	8
19/09/2017 - Melgratti	12
08/08/2017 - Garbervetsky	14
01/08/2017 - Garbervetsky	15
Teóricos de parcial	16
Verano 2019, primer parcial	16
Verano 2019, recuperatorio primer parcial	16
Verano 2019, segundo parcial	17
2do Cuat 2018, primer parcial	18
2do cuat, 2018. Segundo Parcial	19

Finales

03/08/2020 - Melgratti/Garbervetsky - Virtual

1. Diferencia entre la regla de resolución de lógica proposicional y de lógica de primer orden

Hay dos grandes diferencias, una es que en la lógica proposicional buscamos un literal y su negación en dos cláusulas pero la variable proposicional es sintácticamente igual, mientras que en la lógica de primer orden podemos encontrar algo más general en lugar de alguna de estas dos apariciones. Por ejemplo $\{\{P(a)\}, \{\sim P(X)\}\}$ no es sintácticamente igual pero hay una refutación con la sustitución $\{x/a\}$. Esto no ocurre en la log proposicional porque no tenemos cuantificadores ni variables ligadas, solo variables proposicionales.

En segundo lugar, la regla tal cual de lógica proposicional toma un solo literal de cada lado. Hacer esto en la lógica de primer orden es incompleto. Por ejemplo, no podemos refutar $\{\{P(X), P(Y)\}, \{\sim P(V), \sim P(W)\}\}$ con el análogo de la lógica proposicional (llamado regla de resolución binaria). Lo que sí se puede hacer es cambiarla para que tome posiblemente más de un literal a la vez de cada lado (que deberán unificar todos a lo mismo) o agregar una regla de factorización para unificar literales en una misma cláusula.

2. Arbolito SLD y que ramas no se recorrían con un cut (es el arbolito que está como ejemplo de cut de SLD)

La respuesta es exacta la que está en ese ejemplo

3. que pasa con not(not(p(X)))

Not es distinto a la negacion logica. Uno esperaría que la negacion logica de $P(X)$ me devuelva instancias de X para las que no vale P . En su lugar not devuelve True si no puede probar $P(X)$, es decir, no existe instancia de X tal que $P(X)$ es verdadero.

Si tenés sólo un hecho que es $p(juan)$, $\text{not}(p(X))$ falla, porque $p(X)$ es exitoso con $X=juan$. $\text{not}(\text{not}(p(X)))$ es exitoso, pero no instancia $X=juan$, sólamente retorna true. Es decir, $\text{not}(\text{not}(p(X))) \neq p(X)$. Cuando se refiere a que no se liga X , es lo siguiente:

```
not(G) :- call(G), !, fail  
not(G).
```

Cuando se llama a G , es un llamado de G normal, es decir ahí sí vale que se ligan las variables (por eso $\text{not}(p(X))$ falla, porque $p(X)$ es exitoso con $X=juan$)

Pero si G falla, entonces se descartan las sustituciones, se va a la segunda línea y el not es exitoso, sin ligar ninguna variable de G . Por eso $\text{not}(\text{not}(p(X)))$ daría true sin ninguna instancia.

Mandale un mail a Hernán porque me retó en el final por decirle que no lo había entendido y no mandarle consulta.

4. si definieras elem :: a -> [a] -> bool con fix, como un fix M, que tipo tiene M ?

- $M:\sigma\rightarrow\sigma$
 $\text{fix } M: \lambda\sigma$
 $M::(a\rightarrow[a]\rightarrow\text{bool})\rightarrow a\rightarrow[a]\rightarrow\text{bool}$

La respuesta era $M :: (a\rightarrow[a]\rightarrow\text{bool})\rightarrow a\rightarrow[a]\rightarrow\text{bool}$

Cuando le dije ese tipo, me dijo "y si después tenés que hacer que elem tenga el mismo tipo de fix M?" Me confundió un poco eso, era simplemente decir que fix M si M es de tipo sigma en sigma, es sigma. (como escribiste arriba)

$\text{fix } M :: (a\rightarrow[a]\rightarrow\text{bool})\rightarrow a\rightarrow[a]\rightarrow\text{bool}$

- b. M es de tipo: función recursiva sobre sus parámetros, y luego los parámetros.
- 5. - Un seguimiento con clases C1,C2,C3,C4 donde le mandas un mensaje a C4:**
- a. C4 no tiene definido el mensaje, lo tiene C3. C3 tiene un super (dispatch estatico), entonces vas a buscar el mensaje a C2. C2 tiene un self de ese mensaje, self (al igual que super) se liga al evaluar el método al objeto receptor del mensaje que siempre fue de la clase C4, en el caso de self el method lookup comienza nuevamente desde C4 siguiendo por la cadean de ancestros en la herencia.,
- 6. - Que es que con subtipado algo sea covariante**
- Básicamente que el orden de los tipos “abajo” de la regla es el mismo que el de “arriba”. No me salía un ejemplo pero le dije que sería algo así
- ```
sigma <: tau

algo con sigma <: algo con tau
```
- 7. - Ref que es? covariante? contravariante?**
- Solo compara tipos equivalentes (es decir covariante y contravariante al mismo tiempo)
- 8. - Que pierdes al agregar referencias a un cálculo lambda normalito (con bool y nat ponele)**

Vamos a tener que ajustar las nociones de progreso y preservación, extendiéndose principalmente con la información que ahora hay de tipos en el caso de las referencias y que eso tenga sentido con lo que se guarda en las evaluaciones. Podemos verlo en las slides de la teórica. Por otro lado, la ejecución puede colgarse, con solo bools y nat hay terminación asegurada.

- 9. Pregunta compuesta:**
- a. Si tenés cálculo lambda de bool y naturales, sin el fix, podés tener recursión?
    - i. No
  - b. Si agregas referencias pero no agregas el fix, podés tener recursión?
    - i. Podemos guardar una función en una referencia, luego actualizarla para que desreferencie la misma referencia dentro. Y así definir por ejemplo el factorial.
- ```
Ref F = (x:Nat. 0);
F := (x:Nat. If isZero(x) Then 1 else ((!F) (x-1)) * x);
F 10
```

Incluso podemos colgarnos definiendo algo del estilos $f(x) = f(x)$

- 10. Qué es método dispatch estatico vs. dinamico.**
- a. cuando le dije me dijo: que hay que es parecido a eso en smalltalk?
 - b. La respuesta era super. Ponele que tenés en una clase C2 que hereda de C1 y tiene un método que llama a super, siempre se va a ir a buscar esa definición a C1. En eso se “parece” a dispatch estatico
- 11. Para que se quiere subtipado**
- a. Dije que, por ejemplo, para no tener que redefinir operaciones para todos los tipos específicos. Es decir poder escribir algo general que aplique a muchas cosas. El CLT que vemos en la materia, acepta términos bien tipados y cerrados, esto lo hacemos para asegurar que no admitimos programas que se cuelgan o que no se puedan ejecutar pero deja afuera muchos programas que a priori no son malos, como por ejemplo, la identidad definida sobre Floats aplicada sobre un Nat. Bajo esta perspectiva nos sirve para aceptar más programas.

1. De cálculo lambda: Definir con fix la función 'esPar'

```
a. let esPar =
  \f: Nat -> Bool. \x:Nat.
    if isZero(x) then true else
    if isZero(pred(x)) then false else
      f pred(pred(x))
in fix esPar
```

$M' := \lambda x: \text{Nat}. \text{if isZero}(x) \text{ then true else if isZero}(\text{pred}(x)) \text{ then false else } f \text{ pred}(\text{pred}(x))$
 $M := (\lambda f: \text{Nat} \rightarrow \text{Bool}. M')$

let esPar = ($\lambda f: \text{Nat} \rightarrow \text{Bool}$) M' in fix esPar

2. De cálculo lambda: Que problema hay con definir la regla de semántica de "fix f -> f fix f" (lease f como un lambda)

- a. En Haskell anda porque hay evaluación Lazy, en Cálculo Lambda no (loop infinito). La regla de aplicación de Cálculo Lambda espera a que del lado derecho haya un valor para aplicar la función. Pero fix f con esta definición nunca llega a un valor.

3. Qué relación hay entre el juicio de tipado de un término M, y el juicio de tipado producto de hacer $W(Erase(M))$

- a. El juicio de tipado producto de hacer $W(Erase(M))$ es un tipo principal y el juicio de M es una instancia de este.. En otras palabras, el juicio de tipado de $W(Erase(M))$ puede tener variables de tipo, mientras que el de M no.

4. De regla de inferencia: Encontrar un término M que al hacer $W(Erase(M))$, te queda un término distinto sintácticamente a M, pero del mismo tipo

- a. $M = (\lambda f: \text{Nat} \rightarrow \text{Nat}. \text{true}) (\lambda x: \text{Nat}. x)$
- b. M' te queda $(\lambda f: t \rightarrow t. \text{true}) (\lambda x: t. x)$
- c. $M' \neq M$, pero ambos son de tipo Bool (evaluan a true)
- d.

5. De lógica: Un arbolito SLD con cuts y me preguntó qué ramas visitaba y que no.

a.

6. De lógica: $\text{not}(P(X))$, cuando falla, cuando no. Si el árbol de resolución es infinito, ¿qué pasa?

- a. $\text{not}(g) :- \text{call}(g), !, \text{fail}$
 $\text{not}(g).$

Importante entender como funciona:

Si encuentra una solución para g falla, pero por el corte NO BUSCA OTRAS SOLUCIONES. Entonces devuelve **false**.

Si no encuentra una solución para g también falla, pero entra en la segunda regla. Por eso devuelve **true**. Importante, **not** nunca instancia variables.

Por eso no es igual al not lógico, devuelve true si no puede probar que g sea consecuencia lógica del programa (porque no pudo refutar la negación) y falso cuando puede probarlo, osea cuando “miPrograma => g” es valido.

not es verdadero, cuando el árbol de P falla finitamente, es decir pudo recorrer todas las ramas de árbol y no encontró ninguna solución. Si encuentra una sol (aunque sea infinito) da false, y si no encuentra sol, pero es infinito se cuelga el programa y por ende el not. (Se puede separar en infinito por izq y por derecha, si es por izq se cuelga, si es por derecha capaz encuentra algo antes de perderse)

7. De cálculo de objetos: ¿Puede haber recursión infinita? Dar un ejemplo.

18/09/2019 - Melgratti

1. V o F. Sea M derivable en cálculo lambda.

1. $\forall M \exists V \text{ tal que } f i x \ M \Rightarrow V ? \Rightarrow \text{FALSO}$

a. Sea $M = \lambda x:\sigma.x$

$\Rightarrow fix \ M \rightarrow fix (\lambda x:\sigma.x) \rightarrow x \{x < \text{fix} (\lambda x.x)\} \rightarrow fix (\lambda x.x)$

Lo cual es igual a $fix \ M$

Tomar como M la función identidad, para cualquier valor, no termina

2. $\forall M \not\vdash V \text{ tal que } f i x \ M \Rightarrow V ? \Rightarrow \text{FALSO}$

a. $M = \lambda x:\sigma.0$. Fix $M \Rightarrow 0$.

b. Otro ejemplo sería fixFactorial (como está definido en la teórica) para cualquier número

2. Dos términos de $\lambda^{\{b,n\}}$

1. Decir si existen $M : \sigma$, $M' : \sigma'$ tales que $M \neq M'$ y $\sigma \neq \sigma'$ pero $W(Erase(M)) = W(Erase(M'))$

Podemos usar la función identidad para Floats como M y para Bools como M', no son sintácticamente iguales porque tienen anotaciones de tipos que difieren, y también sus tipos difieren. Pero $W(Erase(X))$ en este caso reemplaza el tipo y las anotaciones por variables de tipos, entonces son el mismo juicio principal salvo renombres de variables.

2. Decir si existen $M : \sigma$, $M' : \sigma$ tales que $M \neq M'$ y $W(Erase(M)) = W(Erase(M'))$

$M1 = (\lambda f: \text{Nat} \rightarrow \text{Nat}. \text{true}) (\lambda x: \text{Nat}. x)$

$M2 = (\lambda f: \text{Bool} \rightarrow \text{Bool}. \text{true}) (\lambda x: \text{Bool}. x)$

$M1 \neq M2$ pero $erase(M1) = erase(M2) = (\lambda f.\text{true}) (\lambda x.x)$

Y además Vacio :- M1 : Bool y Vacio :- M2 : Bool.

3. Dar la regla de subtipado para el If-Then-Else

$$\frac{\Gamma \triangleright M : \text{Bool} \quad \Gamma \triangleright P : \tau' \quad \Gamma \triangleright Q : \tau'' \quad \begin{array}{c} \tau' <: \sigma \\ \tau'' <: \sigma \end{array}}{\Gamma \triangleright \text{if } M \text{ then } P \text{ else } Q : \sigma}$$

Nada evita en realidad que usemos subtipado en la condición $\Gamma \triangleright M : \tau'$, $\tau' <: \text{Bool}$ Por ejemplo podemos definir un nuevo tipo Trues, que solo tiene el valor True, y siempre que espero un valor de tipo Bool me pueden devolver un valor de tipo Trues.

4. Mostrar que pasaba si cambiabas el orden de las reglas de semántica operacional de E-Assign1 y E-Assign2. (Proponen primero reducir la parte derecha de la asignación y luego la izquierda)

Como ahora tenes referencias a memoria, podes cambiar algún valor del lado derecho que cambie el resultado del lado izquierdo y viceversa, ej:

let x = ref 3 in (x := !x+2; x) := (!x+8)

- Derivando lado **derecho** primero
 - (x := !x+2; x) := (x!+8) | u+(x->3)
 - (x := !x+2; x) := 11 | u+(x->3)
 - x := 11 | u+(x->5)
 - Unit | u+(x->11)
- Derivando lado **izquierdo** primero
 - (x := !x+2; x) := (x!+8) | u+(x->3)
 - x := (x!+8) | u+(x->5)
 - x := 13 | u+(x->5)
 - Unit | u+(x->13)

5. Prolog. Había un programa simple y una variante con un cut y preguntaba si eran distintas las soluciones en los dos programas con consultas ground.

?-p(X,Y).
?-p(a,b). <- consulta ground Hmmm a corroborar

6. Pregunta de resolución tenias algo asi como {P,P},{-P}. Preguntaba si había resolvente y si podía resolverse en un sólo paso.

Si, hay una resolvente y es C3={}.

7. Ejercicios de seguimiento clásicos de Smalltalk.

19/12/2018 - Melgratti

Resolución posta

https://www.cubawiki.com.ar/images/c/ca/PLP_resolucion-final_19-12-18.pdf

Ejercicio 1

1. Sería correcto definir la semántica del operador `fix` con la siguiente regla? Justifique

$$\text{fix } V \rightarrow V(\text{fix } V) \quad (\text{NEW-FIX})$$

No, se hace infinito. Va metiendo valores a la izquierda cada vez que quiere reducir el fix

Ejercicio 2

2. Mostrar con un ejemplo que al modificar el algoritmo de inferencia de tipos para el caso `if-then-else` tomando S como $S = MGU\{\sigma \doteq \tau, \rho \doteq \text{Bool}\}$, se obtiene un algoritmo que no es correcto.

If M then N else L

Con N: o , L:r y M:p = Bool

If x then x else 2

En este ejemplo, tiparía, pues el tipo de N unifica con el tipo de L que es Nat, pero se infiere que M es de tipo bool y por ende x : Nat y x: Bool al mismo tiempo. Esto está mal, pero el algoritmo no fallaría.

Qué pasa al intentar unir dos contextos que contengan la misma variable (y distinto tipo sin unificar)? Asumimos que la implementación de la unión devuelve un contexto inconsistente.

Una más rebuscada

M = If true then succ(x y) else succ(y x)

Los tipos de x e y son diferentes y la modificación no unifica los tipos de las variables en los contextos, y este M pasaría por más que esté mal

Ejercicio 3

3. Mostrar a través de un ejemplo por qué la siguiente definición de la regla de subtipado para los tipos función no es correcta.

$$\frac{\sigma' <: \sigma \quad \tau' <: \tau}{\sigma \rightarrow \tau <: \sigma' \rightarrow \tau'} \text{(S-FUNC)}$$

Para subtipado la imagen de la función tiene que ser covariante,

Esta bien reemplazar la función por otra que tome mas cosas, pero esta mal devolver más cosas de lo esperado.

Ej

Función original: nat -> bool

Función nueva: float -> nat

A la nueva le van a pasar nats, lo cual está bien, nats es un caso particular de float

Pero la nueva en vez de devolver booleanos va a devolver nats, si la función vieja se usaba en la guarda de un if, al poner esta nueva función no tiparía

Otro: **función original Nat -> Nat.**

Reemplazo por otra de tipo **Float -> Bool.** ($\text{Float} >: \text{Nat}$. $\text{Bool} <: \text{Nat}$).

A la función le llega un Nat, se coerciona a Float. Luego aplico la función nueva, obtengo un Bool, y lo coerciono a Nat.

Si reemplazara por: **Float->Float** (con la nueva regla)

A la función le llega un Nat, se coerciona a Float. Luego aplica la función nueva, obtengo un Float, y no puedo coercionarlo a Nat.

Ejercicio 4

4. Dado el siguiente programa Prolog.

```
a(1,2).  
a(1,3).  
b(3).  
  
p(X,Y):-a(X,Y),b(Y).
```

Y la siguiente modificación para la primera cláusula del programa

```
a(1,2):-!.
```

Indicar si las siguientes afirmaciones son verdaderas o falsas. Justificar mostrando los árboles SLD.

- a) Para toda consulta ground, el conjunto de soluciones no se altera con la modificación del programa.
- b) La consulta `?-not(p(X,Y))` produce distintos resultados si se cambia el programa.

Ejercicio 5

5. Decir si las siguientes afirmaciones son verdaderas o falsas. Justificar:

- a) Sean F y Q dos cláusulas para las cuales no existe resolvente. Luego $\neg F \vee \neg Q$ es una tautología.
- b) Dado el goal $\neg p(X, Y), \neg p(Y, Z)$ y la cláusula de programa $p(X, Y) : -q(Y)$, en el árbol de evaluación que recorre PROLOG se encuentra el goal $\neg q(Y), \neg q(Z)$.

a) Supongo $F = \{ p(X) \}$ y $Q = \{ q(X) \}$

Busco forma clausal de $\neg F \vee \neg Q$.

$\neg F = \neg \forall x p(x) = \exists x \neg p(x) \rightarrow (\text{skolem}) \neg p(c)$

Análogamente para $\neg Q$ llego a $\neg q(d)$

Luego la forma clausal de $\neg F \vee \neg Q$ es $\{ \neg p(c), \neg q(d) \}$

Si fuera una tautología, no debería encontrar una refutación.

Busco una refutación:

$\{ \neg p(c), \neg q(d) \}$ con F queda $\{ \neg q(d) \}$

$\{ \neg q(d) \}$ con Q queda $[]$ (cláusula vacía). Encuentre una refutación.

Luego, **no es una tautología**. REVISAR

Ejercicio 6 (dice 7)

7 **Sólo si cursaste Verano o 2do cuatrimestre 2018** Explicar al menos 4 diferencias significativas entre el modelo de objetos del *cálculo de objetos* y el de *JavaScript*.

7 Si no cursaste Verano ni 2do cuatrimestre 2018 Considere la siguiente definición de clase

```
Object subclass: #Miclase
instanceVariableNames: ''
classVariableNames: ''
poolDictionaries: ''
category: 'Unclassified'

m:aBlock
| a |
a:=2.
^(aBlock value + a).
```

a) Decir cual es el resultado de evaluar el siguiente código.

```
|a|
a :=1.
b := [a].
^(Miclase new) m:b.
```

Objeto	Mensaje	Colaboradores	Resultado
Miclase	new		aMiclaseObject
aMiclaseObject	m	b -> [a]; a->1; [a]->[1]	
[1]	value		1
1	+	2	3

b) Suponer que se agregan las siguientes definiciones:

```
MiClase subclass: #MiClase1
instanceVariableNames: ''
classVariableNames: ''
poolDictionaries: ''
category: 'Unclassified'

n
^super m:[0]

MiClase1 subclass: #MiClase2
instanceVariableNames: ''
classVariableNames: ''
poolDictionaries: ''
category: 'Unclassified'

m:aBlock
^self n.
```

Indicar cuál es el resultado que se obtiene al evaluar

`^(MiClase2 new) m:[0].`

Justificar mostrando las colaboraciones.

Objeto	Mensaje	Colaborador es	MethodLook up	Resultado
Miclase2	new		Miclase2	aMiclase2Object
aMiclase2Object	m	[0]	Miclase2	2
aMiclase2Object	n		Miclase1	2
aMiclase2Object	m	[0]	Miclase	2
[0]	value		BlockClass	0
0	+	2	Int	2

c) Qué sucede al evaluar la misma expresión si la definición del método n se modifica como:

```
n
^self m:[0]
```

Objeto	Mensaje	Colaborador es	MethodLook up	Resultado
Miclase2	new		Miclase2	aMiclase2Object
aMiclase2Object	m	[0]	Miclase2	
aMiclase2Object	n		Miclase1	
aMiclase2Object	m	[0]	Miclase2	
aMiclase2Object	n		Miclase1	
aMiclase2Object	m	[0]	Miclase2	loop infinito

19/09/2017 - Melgratti

1. Dado $M = \lambda x : \sigma \rightarrow \sigma . \lambda y : \sigma . x \ y$

1. Existe V tal que $\text{fix } M \Rightarrow V$?

Aplicando la definición de fix ya te queda un valor

$\text{fix} (\lambda x : \sigma \rightarrow \sigma . \lambda y : \sigma . x \ y)$

$\lambda y : \sigma . \text{fix} (\lambda x : \sigma \rightarrow \sigma . \lambda y : \sigma . x \ y) \ y$

2. Existe V tal que $\text{fix } (M(\lambda x : \sigma . x)) \Rightarrow V$?

$\text{fix } (M(\lambda x : \sigma . x))$

$\text{fix } ((\lambda x : \sigma \rightarrow \sigma . \lambda y : \sigma . x \ y) (\lambda x : \sigma . x))$

Aplico $(\lambda x : \sigma . x)$ a M

$\text{fix } \lambda y : \sigma . ((\lambda x : \sigma . x) y)$

Ya es un valor, aplico fix

$\lambda x : \sigma . (x (\text{fix } \lambda y : \sigma . ((\lambda x : \sigma . x) y)))$

Ya queda un valor

2. Pregunta de subtipado que no recuerdo bien. Tenía dos clases (una era subclase de la otra), y pregunta si puedo usar un supertipo de uno de los parámetros de la función de la superclase en una función sobreescrita por la subclase. Pregunta similar pero con ref (siendo este un atributo de la subclase)

3. Prolog.

p(1).

p(2).

q(X) :- p(X), not(q(X)).

q(X) :- p(X).

a) Que devuelve q(W)?

Se cuelga

b) Que sucede si intercambio las ultimas dos lineas del programa?

p(1).

p(2).

q(X) :- p(X).

q(X) :- p(X), not(q(X)).

Devuelve x=1, x=2 y despues el false

c) Que sucede si cambio la tercer linea por q(X) :- not(p(X)), q(X). ?

Devuelve x=1, x=2

4. Pregunta de resolución cuya fórmula no recuerdo. Preguntaba si podía resolverse en un sólo paso.

5. Ejercicios de seguimiento clásicos de Smalltalk.

08/08/2017 - [Garbervetsky](#)

1. Dado $f \ a \ b = b : (a \ (-b))$ decir que representa la expresión fix f 1

1. fix f 1
2. fix ($\lambda a. \lambda b. Nat. b : (a(-b))$) 1
3. ($\lambda b. Nat. b : (a(-b))$) { $a < fix (\lambda a. \lambda b. Nat. b : (a(-b)))$ } 1
4. $\lambda b. Nat. b : ((fix \lambda a. \lambda b. Nat. b : (a(-b))) (-b))$ 1
5. $1 : ((fix \lambda a. \lambda b. Nat. b : (a(-b))) (-1))$
6. $1 : (\lambda b. Nat. b : (fix \lambda a. \lambda b. Nat. b : (a(-b))) (-1))$
7. $1 : -1 : ...$

2. Dado los términos:

$$U_1 = (\lambda x. \lambda y. x y) (\lambda z. z) (\lambda w. w)$$

$$U_2 = (\lambda x. \lambda w. z w) (\lambda y. y)$$

Ver y justificar. ¿El algoritmo de inferencia de tipos infiere el mismo resultado para ambos?.

Asumiendo un tipo en la z de U2, tomándola como una x (sino los contextos no serían los mismos y los tipos resultados tampoco).

$$W(U_1) = \text{vacío} :> (\lambda x : (t_1 \rightarrow t_1) \rightarrow (t_1 \rightarrow t_1). \lambda y : t_1 \rightarrow t_1. x y) (\lambda z : t_1 \rightarrow t_1. z) (\lambda w : t_1 \rightarrow t_1. w) : t_1 \rightarrow t_1$$

$$W(U_2) = \text{vacío} :> (\lambda x : (t_1 \rightarrow t_1). \lambda w : t_1. x w) (\lambda y : t_1. y) : t_1 \rightarrow t_1$$

El algoritmo devuelve un juicio de tipado, como los términos son distintos ninguno es instancia del otro. Si es verdad que los tipos de los juicios son iguales salvo renombre y que el contexto en ambos es vacío.

3. Dado el programa en Prolog:

1. P(X) :- ! , Q(X)
2. P(X) :- R(X)

1. Indicar verdadero o falso para cualquier Q(X), R(X):

1. El cut de (i) se puede eliminar sin alterar el conjunto de soluciones.

Falso, como el cut viene primero y siempre tiene éxito, no se exploran la segunda cláusula que resuelve con $\sim P(x)$, si quitamos el cut entonces pueden haber más soluciones dependiendo de R(X). Ejemplo si agregamos “R(1).” al programa.

2. El conjunto de soluciones no se modifica si la segunda regla se cambia por P(X) :- ! , R(X)

Verdadero, nunca se llega a la segunda cláusula de P(X)

4. Dado : C 1 = { $\neg P(x)$, $\neg P(a)$ } y C 2 { $P(w)$ } , indicar verdadero o falso:

1. C 3 = { } es resolvente de C1 y C2.

Verdadero.

Utilizando resolucion normal (se puede usar más de átomo de la misma cláusula en el paso de resolución)

$$S = Mgu = \{x <- w, a <- w\},$$

Luego S(C1)

Entonces el resolvente entre S(C1) y C2 es {}

Falso si nos obligan a usar resolución binaria (no está claro en el enunciado)

2. C3 puede obtenerse en un único paso de resolución SLD a partir de C1 y C2. =>

Falso, porque SLD resuelve una sola cláusula del goal, utiliza resolución binaria y en este caso hay que factorizar o hacer dos pasos de resolución.

$$\{\neg P(x), \neg P(a)\}, \{P(w)\} \{\neg P(z)\} \text{ Fact}$$

$\{\neg P(x), \neg P(a)\}, \{P(w)\} \{\neg P(z)\} \{\}$ C2 y factorizada MGU ($w <- z$)

$$\{\neg P(x), \neg P(a)\}, \{P(w)\} \{\neg P(z)\} \quad \text{C1 y C2 con MGU } (w <- a)$$

$\{\neg P(x), \neg P(a)\}, \{P(w)\} \{\neg P(z)\} \{\}$ resolvente y C2 MGU ($w <- z$)

5. Explicar qué pasaría si redefinimos (S-func) de la siguiente manera:

$$\sigma <: \sigma' \tau <: \tau'$$

$$\sigma \rightarrow \tau <: \sigma' \rightarrow \tau'$$

6. Objetos

1. Seguimiento. Clase A, m:^1, n:^self m. Clase B n: super n. Clase C, n:^2, Clase D ...

1. Indicar objetos y mensajes de: (ClaseD new) n

2. Realizar lo mismo, pero agregando a ClaseC new: ^ClaseB new.

01/08/2017 - Garbervetsky

1. Dado $f x = x (f x)$ en haskell

1. ¿Cuál es el resultado de hacer $f (\lambda x \rightarrow 1 : x)$?

$f (\lambda x \rightarrow 1 : x) \rightarrow (\lambda x \rightarrow 1 : x) (f (\lambda x \rightarrow 1 : x)) \rightarrow 1 : (f (\lambda x \rightarrow 1 : x))$

Lista infinita de 1s (por ejecución lazy de haskell) => No vale en cálculo lambda

- $f (\lambda x \rightarrow 1 : x)$
- $(\lambda x \rightarrow 1 : x) (f (\lambda x \rightarrow 1 : x))$
- $(\lambda x \rightarrow 1 : x) (\lambda x \rightarrow 1 : x) (f (\lambda x \rightarrow 1 : x))$

2. ¿Qué representa la función f?

- Aplicar una función infinitas veces
- Representa la función fix

2. Dados los términos:

$U_1 = (\lambda x . \lambda y . x y) (\lambda z . z) (\lambda w . w)$

$U_2 = (\lambda x . x x) (\lambda h . h)$

¿El algoritmo de inferencia da el mismo resultado para ambos?.

3. Se tiene el siguiente programa en Prolog:

$P(X) :- Q(X), R(X), !, S(X).$

$P2(X) :- Q(X), !, R(X), S(X).$

Para cualquier $Q(X), R(X), S(X)$ (es posible definir los que se requieran).

1. ¿Los resultados de $P(X)$ están contenidos en los de $P2(X)$?

Falso. Podemos definir Q,R y S de la siguiente forma y P tiene soluciones mientras P2 ninguna.

$Q(1).$

$Q(2).$

$R(2).$

$S(2).$

2. ¿Los resultados de $P2(X)$ están contenidos en los de $P(X)$?

Falso. Podemos definir Q para que sea valida sin instanciar X, luego varias instancias de R S que satisfagan un mismo valor de X, y P encontrara solo una dado que R(X) fija X y no vuelve backtrackear una vez del cut.

$Q(X).$

$R(1).$

$R(2).$

$S(1).$

$S(2).$

3. ¿Qué pasa ahora si tenemos P3(X) :- Q(X), !, R(X), !, S(X)? (Cómo se comporta en comparación a P y P2)

Va por casos. Si Q(X) fija el valor de X, entonces P2 y P3 pueden conseguir en el mejor caso una solución y quizás ninguna, mientras que P puede conseguir 0, 1 o más.

Si R(X) fija el valor de X, sucede algo parecido pero P3 y P tienen 0 o 1 solución mientras P2 0, 1 o más.

Si S(X) fija el valor de X, entonces todas consiguen la misma cantidad de soluciones.

4. Verdadero o falso:

1. $\{P(x, y)\} \cup \{\neg P(y, f(y))\}$ no unifican
 - i. $\{P(x, y)\} \cup \{\neg P(z, f(z))\}$
 1. $x <- z, y <- f(z)$
2. La Forma Normal de Skolem de $\forall x \forall y \exists z (P(x, z) \wedge \neg P(y, z))$ es $\forall x \forall y \exists z (P(x, f(x)) \wedge \neg P(y, f(y)))$
 - i. No, es $\forall x \forall y (P(x, f(x, y)) \wedge \neg P(y, f(x, y)))$
3. La Forma Normal de Skolem de $\forall x \forall y ((\exists z P(x, z)) \wedge (\exists z \neg P(y, z)))$ es $\forall x \forall y (P(x, f(x)) \wedge \neg P(y, f(y)))$
 - i. Falso, deberían usarse dos funciones distintas (no f y f), la skolemización queda: $\forall x \forall y (P(x, f(x)) \wedge \neg P(y, g(y)))$

Otra forma pasando a prenexa: (sacado del apunte de Ivan A.)

$$\forall x \forall y \exists z \exists w (P(x, z) \wedge \neg P(y, w))$$

$$\forall x \forall y \exists w (P(x, f(x, y)) \wedge \neg P(y, w))$$

$$\forall x \forall y (P(x, f(x, y)) \wedge \neg P(y, g(x, y)))$$

5. Explicar qué pasaría si redefinimos (S-func) de la siguiente manera:

$$\begin{aligned}\sigma' &<: \sigma \quad \tau' <: \tau \\ \sigma \rightarrow \tau &<: \sigma' \rightarrow \tau'\end{aligned}$$

Resuelto en [19/12/2018 - Melgratti](#)

6. Objetos

1. ¿Cuál es la diferencia entre self y super?

Self se liga en tiempo de ejecución y super es estático

2. Seguimiento.

Teóricos de parcial

Verano 2019, primer parcial

Preguntas Teóricas

- Dicir si las siguientes afirmaciones son válidas. Justificar su respuesta.
 - Existe $M \in \lambda^{bn}$ tal que $\emptyset \vdash M : \sigma$ es derivable, $M \rightarrow V$ y $\emptyset \vdash V : \tau$ es derivable para algún $\tau \neq \sigma$.
 - Para todo $M \in \lambda^{bn}$, si $\emptyset \vdash \text{fix } M : \sigma$ es derivable, entonces existe V tal que $\text{fix } M \rightarrow V$.
- Mostrar con un ejemplo que al modificar el algoritmo de inferencia de tipos para el caso **if-then-else** tomando S como $S = MGU\{\sigma \doteq \tau, \rho \doteq \text{Bool}\}$, entonces el algoritmo que se obtiene no es correcto.
- Mostrar, si es posible, un término M tal que se cumpla:
 - $\emptyset \vdash M : \sigma$ y
 - $\mathbb{W}(\text{Erase}(M)) = \Gamma \vdash M' : \rho$ y
 - $\sigma = \rho$ y
 - $M \neq M'$.

Soluciones (corregidas): https://www.cubawiki.com.ar/images/d/d0/PLP_1parcial_2019_v.pdf

A.1) No existe. Ya que rompería la preservación de tipos. Preservación:

Si $\Gamma > M : \sigma$ y $M \rightarrow N$, entonces $\Gamma > N : \sigma$

A.2) Falso. Fix $M:\sigma$, con $M: \sigma \rightarrow \sigma$. Sin embargo, no reduce a un valor para fix ($t: \text{Nat}$. t)

b) Hecho acá

c) $M = ((\lambda x:\text{bool}.0) y:\text{bool}) : \text{Nat}$, cuando se borren los tipos y se haga la inferencia, x va a quedar tipada con una variable de tipo s y el $M':\text{Nat}$, y van a ser distintos.

Verano 2019, recuperatorio primer parcial

Preguntas teóricas

- Definir la función **map** sobre listas en términos del operador **fix**.
- Considerar el cálculo lambda con referencias. Ilustrar a través de un ejemplo que la semántica para algunos programas cambia al modificar las siguientes reglas :

$$\frac{M_2 \mid \mu \rightarrow M'_2 \mid \mu'}{M_1 := M_2 \mid \mu \rightarrow M_1 := M'_2 \mid \mu'} \text{ (E-ASSIGN1)} \quad \frac{M_1 \mid \mu \rightarrow M'_1 \mid \mu'}{M_1 := V \mid \mu \rightarrow M'_1 := V \mid \mu'} \text{ (E-ASSIGN2)}$$

- Dar un ejemplo para ilustrar que la siguiente definición del algoritmo de inferencia es incorrecta:

$$\mathbb{W}(\lambda x.U) = \Gamma \triangleright \lambda x : s.M : s \rightarrow \sigma$$

donde s fresca y $\mathbb{W}(U) = \Gamma \triangleright M : \sigma$.

a:

```
fix :: (a -> a) -> a
fix f = let {x = f x} in x
```

Reescribimos map con abstracciones sin pattern matching

```
mapf :: (a->b) -> [a] -> [b]
mapf = \f -> \as -> if (vacia as) then [] else f (cabeza as) : mapf f (cola as)
```

```
fixmap :: (a->b) -> [a] -> [b]
```

```
fixmap = fix (\mapfunc -> \f -> \as -> if (vacia as) then [] else f (cabeza as) : mapfunc f (cola as))
```

b:

```
let x = ref 3 in (x := !x+2; x) := (!x+8)
```

- Derivando lado **derecho** primero
 - $(x := x!+2; x) := (x!+8) | u+(x>3)$
 - $(x := x!+2; x) := 11 | u+(x>3)$
 - $x := 11 | u+(x>5)$
 - $Unit | u+(x>11)$
- Derivando lado **izquierdo** primero
 - $(x := x!+2; x) := (x!+8) | u+(x>3)$
 - $x := (x!+8) | u+(x>5)$
 - $x := 13 | u+(x>5)$
 - $Unit | u+(x>13)$

C:

$$\begin{array}{c}
 \text{Wr} \left(\underbrace{(\lambda x. \text{suc}(x))}_{u} / \underbrace{(\text{if } x \text{ then } 0 \text{ else } \text{suc}(0))}_{v} \right) = ?
 \\[10pt]
 w(u) \stackrel{\text{CAMBIO}}{=} \{x : \text{Nat}\} \triangleright \lambda x : \text{Nat}. \text{suc}(x) : \text{Nat} \rightarrow \text{Nat}
 \\[10pt]
 w(v) = \{x : \text{Bool}\} \triangleright \text{if } x \text{ then } 0 \text{ else } \text{suc}(0) : \text{Nat}
 \\[10pt]
 \cancel{\text{MGU} = \dots \{ \text{Nat} = \text{Bool} \mid \begin{array}{l} x : \text{Nat} \in F_1 \\ x : \text{Bool} \in F_2 \end{array} \}}
 \\[10pt]
 \underline{\text{FALLA}}
 \end{array}$$

Verano 2019, segundo parcial

Preguntas Teóricas

a. Indicar si las siguientes afirmaciones son verdaderas o falsas. Justificar:

- I) $\{\{P(x, g(x))\}, \{P(f(g(x)), g(x))\}$ es una forma clausal de $\forall x. \exists y. (\exists z. P(z, y) \wedge P(x, y))$.
- II) Considere la siguiente fórmula en forma clausal:

$$\{\{P(X, g(a), Y)\}, \{\neg P(Z, g(X), Z)\}, \{P(W, g(W), g(W))\}\}$$

- 1) Existen al menos dos refutaciones SLD distintas.
- 2) Prolog computa la solución $\{X \leftarrow a, Z \leftarrow a\}$.

b. Definir, si es posible, un predicado $p(X)$ tal que el árbol SLD para el goal $\neg p(W)$ es infinito y

- I) Prolog responde **false** a la consulta $\text{not}(p(W))$.
- II) Prolog responde **true** a la consulta $\text{not}(p(W))$.

c. Considere un conjunto de traits t_1, \dots, t_n definidos de manera tal que para todo i

$$t_i = [x = \lambda(z)b_i^x, y = \lambda(z)b_i^y]$$

- I) Definir una clase C que provee un constructor new que recibe como parámetro a uno de estos traits y crea instancias que responden a los mensajes x e y de acuerdo con el trait usado al momento de la creación. Por ejemplo, si se definen $t_1 = [x = \lambda(z)0, y = \lambda(z)\text{true}]$ y $t_2 = [x = \lambda(z)1, y = \lambda(z)[]]$, entonces $C.\text{new}(t_1).x$ evalúa 0 y $C.\text{new}(t_2).x$ evalúa 1.
- II) Cómo se modifica la definición de C si las instancias generadas deben permitir cambiar dinámicamente al trait que utilizan. Por ejemplo, $C.\text{new}(t_1).\text{cambiar}(t_2).x$ debe evaluar 1.

A.1

No. Skolemizando,
primero por la y :

$$\forall x (\exists z P(z, g(x)) \wedge P(x, g(x)))$$

Luego, la z :

$$\forall x (P(f(x), g(x)) \wedge P(x, g(x))).$$

Si elegimos primero la z :

$$\forall x (\exists z P(z, g(x)) \wedge P(x, g(x)))$$

Es distinto a lo que dice el enunciado.

Creo que eligiendo primero la z se llega a lo que dice el enunciado:

$$\forall x \exists y (\exists z P(z, y) \wedge P(x, y)) \rightarrow \exists z \text{ solo abarca el primer } P \text{ porque no tiene parentesis.}$$

$$\forall x \exists y (P(f(y), y) \wedge P(x, y))$$

$$\forall x \exists y (P(f(y), y) \wedge P(x, y))$$

$$\forall x (P(f(g(x)), y) \wedge P(x, g(x)))$$

$\{P(f(g(x)), y), P(x, g(x))\} \rightarrow$ es lo mismo que el enunciado.

Otra ?

$\forall x \exists y (\exists z P(z, y) \wedge P(x, y)) \rightarrow$ paso a Prenexa

$\forall x \exists y \exists z (P(z, y) \wedge P(x, y))$

$\forall x \exists y (P(f(x, y), y) \wedge P(x, y))$

$\forall x \exists y (P(f(x, y), y) \wedge P(x, y))$

$\forall x (P(f(x, g(x)), y) \wedge P(x, g(x)))$

A.2

1.

Resolucion SLD:

Goal: $\neg P(Z, g(X), Z)$

1) $P(X, g(a), Y)$

2) $P(W, g(W), g(W))$

Goal con 1) :

Renombro 1) a $P(X_1, g(a), Y_1)$

{}. Con S = { $Y_1 \leftarrow X_1, Z \leftarrow X_1, X \leftarrow a$ }

Goal con 2): no unifican.

Existe solo 1 resolucion SLD.

Con 2)

Elegí un archivo con cláusulas para empezar:

Seleccionar archivo final.txt

- final.txt (text/plain) - 39 bytes, modificado: Sat Sep 26 2020 21:14:17 GMT-0300 (hora estándar de Argentina)

Luego hacé clic en las cláusulas para seleccionarlas, y elegí el o los literales a unificar para cada cláusula.

1: { $p(X_1, g(a), Y_1)$ }

2: { $p(W_2, g(W_2), g(W_2))$ }

3: { $\neg p(Z_3, g(X_3), Z_3)$ }

{ $\neg p(Z_3, g(X_3), Z_3)$ } { $p(W_2, g(W_2), g(W_2))$ }

Falla: occurs check para el par $Z_3 \doteq g(Z_3)$.

Calcular resolvente

Agregar a la lista

Con 1)

Elegí un archivo con cláusulas para empezar:

Seleccionar archivo final.txt

- final.txt (text/plain) - 39 bytes, modificado: Sat Sep 26 2020 21:14:17 GMT-0300 (hora estándar de Argentina)

Luego hacé clic en las cláusulas para seleccionarlas, y elegí el o los literales a unificar para cada cláusula.

1: { $p(X_1, g(a), Y_1)$ }

2: { $p(W_2, g(W_2), g(W_2))$ }

3: { $\neg p(Z_3, g(X_3), Z_3)$ }

Calcular resolvente Agregar a la lista

De 1 y 3 con { $X_1 \leftarrow Z_3, Y_1 \leftarrow Z_3, X_3 \leftarrow a$ }:

4: \square

2. Que hace Prolog? En base a la primer resolución deberá devolver X <- a, con Z cualquier valor.

Lo probe en prolog y devuelve:

?- p(Z, g(X), Z).

X = a ; (primera sol)

Z = X, X = g(X). -> no entiendo de donde saca esta otra solución.

B.1)

p(1).

p(X) :- p(X), p(X).

B.2) No se puede, Si el árbol es infinito el not nunca puede evaluarlo todo para encontrar que no falla.

2do Cuat 2019, primer parcial

Ejercicios teóricos

- a. Definir a la función **filter** sobre listas en términos del operador **fix**.
- b. Mostrar, si existen, dos términos $M_1, M_2 \in \lambda^{b,n,\text{fix}}$ tal que:
 - i) $\Gamma_1 \triangleright M_1 : \sigma \rightarrow \tau, \Gamma_2 \triangleright M_2 : \sigma$ y $M_1 M_2$ no es tipable. En caso contrario, justificar.
 - ii) $\exists V_1, V_2$ tal que $M_1 \rightsquigarrow V_1, M_2 \rightsquigarrow V_2$, pero no existe V_3 tal que $M_1 M_2 \rightsquigarrow V_3$. En caso contrario, justificar.
- c. Considerar al algoritmo de unificación. Mostrar con un ejemplo que si se eliminan la condición $s \notin FV(\sigma)$ de la regla **Eliminación de variable** y la regla **Occur check** el algoritmo que se obtiene no es correcto.

a)

filter = fix \f: (sigma -> Bool) -> [sigma] -> [sigma].

\p: sigma -> Bool. \l: [sigma]. case l of

[] -> []sigma

h :: t -> if (p h) then h :: (f p t) else (f p t)

b)1)

M1: {x:Bool} > (\y: Nat) if x then succ(y) else 0

M2: {x:Nat} > succ(x).

No tipa, los contextos no unifican

b)2) No entendí bien el enunciado, asumo que no sabemos nada de los tipos de M1 y M2. En ese caso, podemos definir M1 = 0, M2= succ(0). M1M2 no devuelve un valor, ya que no tipa.

Asumiendo que los tipos de M1 y M2 sean como en el punto anterior: M1: sigma -> tau y M2: sigma, una sol podría ser:

$M_1 = \lambda x: \text{Nat} \rightarrow \text{Nat}. \text{fix } x$

$M_2 = \lambda y: \text{Nat} . y$

Ambos M_1 y M_2 son abstracciones (valores) pero $M_1 M_2$ no evalua a un valor (fix id se cuelga)

c)

No sé si está bien:

Se modifica la regla 4 y se elimina la condición como dice el enunciado

Se elimina la regla 6 (occur check) como dice el enunciado

El siguiente ejemplo rompe

{ $s = \text{Nat} \rightarrow s$, $u = s \rightarrow \text{Bool}$ }

Usas regla 4: ($s = \text{Nat} \rightarrow s$) { $u = \text{Nat} \rightarrow s \rightarrow \text{Bool}$ }

Esto no va a terminar nunca porque se puede seguir reemplazando s infinitamente

2do Cuat 2018, primer parcial

Preguntas Teóricas

Este ejercicio será corregido si el examen se encuentra en condición de promocionar. Entregar en una hoja separada al resto de los ejercicios.

a. Dada la siguiente función Haskell:

$f a b = b : (a \ (-b))$

Decir que representa la expresión $\text{fix } f \ 1$.

b. Mostrar un término $M \in \lambda^{bn}$ tal que $\emptyset \triangleright M : \text{Nat} \rightarrow \text{Nat}$ y

i) $\text{fix } M \Rightarrow V$ con V valor.

ii) no existe V tal que $\text{fix } M \Rightarrow V$ y V valor.

c. Mostrar, si es posible, dos términos $M_1 \neq M_2 \in \lambda^{bn}$ tal que:

▪ $\emptyset \triangleright M_1 : \sigma$, $\emptyset \triangleright M_2 : \sigma'$, y $\text{Erase}(M_1) = \text{Erase}(M_2)$.

▪ $\emptyset \triangleright M_1 : \sigma$, $\emptyset \triangleright M_2 : \sigma$, y $\text{Erase}(M_1) = \text{Erase}(M_2)$.

Soluciones (corregidas):

https://www.cubawiki.com.ar/images/a/ac/PLP_1parcial_2018_2c_solucion.pdf

a)

La lista infinita: [1,-1,1,-1...]

b)

1)
 $(\lambda f: \text{Nat} \rightarrow \text{Nat}, \lambda n: \text{Nat} . \text{if } \text{isZero}(n) \text{ then succ}(0) \text{ else } n * f(\text{pred}(x))) \rightarrow \text{E-fixBeta}$
 $\lambda n: \text{Nat} . \text{if } \text{isZero}(n) \text{ then succ}(0) \text{ else } n * (\text{fix } (\lambda f: \text{Nat} \rightarrow \text{Nat}, \lambda n: \text{Nat} . \text{if } \text{isZero}(n) \text{ then succ}(0) \text{ else } n * f(\text{pred}(x)))) (\text{pred}(x))$

Otra opción:

$M = \lambda x: \text{Nat}. 0 \rightarrow 0$

2) $M = (\lambda x: \text{Nat}. x)$

c)

1) $M_1 = (\lambda x: \text{Nat}. x) M_2 = (\lambda x: \text{Bool}. x)$

2) $M_1 = \lambda x : \text{Nat}. \text{suc}(x) \text{ y } M_2 = \lambda x : \text{Nat}. \text{suc}(\text{suc}(x))$

O $M_1 = (\lambda f: \text{Nat} \rightarrow \text{Nat}. \text{true}) (\lambda x: \text{Nat}. x) \text{ y } M_2 = (\lambda f: \text{Bool} \rightarrow \text{Bool}. \text{true}) (\lambda x: \text{Bool}. x)$

2do cuat, 2018. Segundo Parcial

Preguntas Teóricas

a. Decidir si por verdadero o falso cada una de las siguientes afirmaciones, justificando cada una de sus respuestas:

- I) No existe resolvente para $\{P(g(a, x), x, f(a))\} \cup \{\neg(P(x, y, y)\}$ porque los literales no unifican;
- II) $\forall x. \forall y. (P(x, f(x)) \wedge \neg P(y, f(y)))$ es una forma normal de Skolem de $\forall x. \forall y. \exists z. (P(x, z) \wedge \neg P(y, z))$

b. Definir en el cálculo de objetos, la clase **Conjunto** cuyas instancias responden los mensajes descriptos en el ítem 2.b.

c. Sea un programa en Prolog que redefine el **not** como:

```
not(G) :- call(G), fail, !.  
not(G).
```

Comparar el resultado de evaluar `not(P)` con la definición original de `not`.

a.1)

Primero, re-nombro las variables:

$P(g(a,x), x, f(a)) \rightarrow P(g(a,z), z, f(a))$ [x por z]

Luego:

$S = \{g(a,z) \leftarrow x, z \leftarrow y, f(a) \leftarrow y\}$

Entonces $P(g(a,z), z, f(a)) \rightarrow P(x, y, y)$

Por tanto unifican y el resolvente es {} (la cláusula vacía)

a.2)

Skolemizamos: $\forall x \forall y \exists z (P(x, z) \wedge \neg P(y, z))$ y nos queda

$\forall x \forall y P(x, f(x, y)) \wedge P(y, f(x, y))$

Lo cual es distinto que $\forall x \forall y P(x, f(x)) \wedge P(y, f(y))$ (como dice el enunciado)

c) not original

`not(G) :- call(G), !, fail`

`not(G).`

Si G tiene una solución, `not(G)` tiene que dar falso.

En la versión modificada, si G tiene una solución, falla antes de ejecutar el ! y entra en la segunda cláusula, por lo que el `not` devuelve true.

En la versión original hace el `call`, encuentra solución, no busca otra y falla