```
(defrule progenitor
(padre-de ?padre ?hijo)
(marido-de ?padre ?madre)
=>
(assert (progenitor-de ?madre ?hijo))
)
(defrule esposa
(marido-de ?hombre ?mujer)
=>
(assert (esposa-de ?mujer ?hombre))
(printout t ?mujer " es la esposa de " ?hombre crlf)
)
(defrule padre
(progenitor-de ?padre ?hijo)
(hombre ?padre)
=>
(assert (padre-de ?padre ?hijo))
(printout t ?padre " es padre de " ?hijo crlf)
)
(defrule madre
(progenitor-de ?madre ?hijo)
(mujer ?madre)
=>
(assert (madre-de ?madre ?hijo))
(printout t ?madre " es madre de" ?hijo crlf)
)
; ----- Abuelos
(defrule abuelos
(progenitor-de ?padre ?hijo)
(progenitor-de ?hijo ?nieto)
=>
```

```
(assert (abuelos-de ?padre ?nieto))
)
(defrule abuelo
(abuelos-de ?padre ?nieto)
(hombre ?padre)
=>
(assert (abuelo-de ?padre ?nieto))
(printout t ?padre " es el abuelo de " ?nieto crlf)
)
(defrule abuela
(abuelos-de ?madre ?nieto)
(mujer ?madre)
=>
(assert (abuelo-de ?madre ?nieto))
(printout t ?madre " es el abuela de " ?nieto crlf)
)
;----- Hermanos ---
(defrule hermanos-padre
(padre-de ?padre ?hijo1)
(padre-de ?padre ?hijo2)
(test (neq ?hijo1 ?hijo2))
=>
(assert (hermanos ?hijo1 ?hijo2))
)
(defrule hermanos-madre
(madre-de ?madre ?hijo1)
(madre-de ?madre ?hijo2)
(test (neq ?hijo1 ?hijo2))
=>
(assert (hermanos ?hijo1 ?hijo2))
)
```

```
(defrule hermano
(hermanos ?hijo1 ?hijo2)
(hombre ?hijo1)
=>
(assert (hermano-de ?hijo1 ?hijo2))
(printout t ?hijo1 " es hermano de " ?hijo2 crlf)
)
(defrule hermana
(hermanos ?hijo1 ?hijo2)
(mujer ?hijo1)
=>
(assert (hermana-de ?hijo1 ?hijo2))
(printout t ?hijo1 " es hermana de " ?hijo2 crlf)
)
; ----- Tios -----
(defrule tios
(progenitor-de ?padre ?hijo)
(hermanos ?padre ?hermano)
=>
(assert (tios ?hermano ?hijo))
)
(defrule tio
(tios ?tio ?sobrino)
(hombre ?tio)
=>
(assert (tio ?tio ?sobrino))
(printout t ?tio " es tio de " ?sobrino crlf)
)
(defrule tia
(tios ?tia ?sobrino)
(mujer ?tia)
```

```
=>
(assert (tia-de ?tia ?sobrino))
(printout t ?tia " es tia de " ?sobrino crlf)
)
(defrule sobrino
(tios ?tios ?sobrino)
(hombre ?sobrino)
=>
(assert (sobrino-de ?sobrino ?tios))
(printout t ?sobrino " es sobrino de " ?tios crlf)
)
(defrule sobrina
(tios ?tios ?sobrina)
(mujer ?sobrina)
=>
(assert (sobrina-de ?sobrina ?tios))
(printout t ?sobrina " es sobrina de " ?tios crlf )
)
(defrule bisabuelos
(progenitor-de ?abuelo ?padre)
(progenitor-de ?padre ?hijo)
(progenitor-de ?hijo ?nieto)
=>
(assert (bisabuelos-de ?abuelo ?nieto))
)
(defrule bisabuelo
(progenitor-de ?abuelo ?nieto)
(hombre ?abuelo)
=>
(assert (bisabuelo-de ?abuelo ?nieto))
(printout t ?abuelo " es bisabuelo de " ?nieto crlf)
```

```
)
(defrule bisabuela
(bisabuela-de ?abuela ?nieto)
(mujer ?abuela)
=>
(assert (bisabuela-de ?abuela ?nieto))
(printout t ?abuela " es bisabuela de " ?nieto crlf)
)
(defrule primos
(primos ?primo1 ?primo2)
(sobrino ?tio ?primo1)
(sobrino ?tio2 ?primo2)
(test (neq ?primo1 ?primo2))
=>
(assert (primos ?primo1 ?primo2))
)
(defrule primo
(primos ?primo1 ?primo2)
(hombre ?primo1)
=>
(assert (primo-de ?primo1 ?primo2))
(printout t ?primo1 " primo de " ?primo2 crlf)
)
(defrule prima
(primos ?primo1 ?primo2)
(mujer ?primo1)
=>
(assert (primo-de ?primo1 ?primo2))
(printout t ?primo1 " prima de " ?primo2 crlf)
)
; --- Hechos ----
```

(deffacts inicio

(hombre Ricardo)

(mujer Fabiola)

(hombre Andres)

(mujer Rosa)

(mujer Maria)

(hombre Mario)

(mujer Irene)

(hombre David)

(hombre Alejandro)

(hombre Luis)

(hombre Enrique)

(progenitor-de Ricardo David)

(progenitor-de Ricardo Alejandro)

(progenitor-de Ricardo Mario)

(marido-de Ricardo Fabiola)

(progenitor-de David Luis)

(marido-de David Rosa)

(bisAbuelo-de David)

)