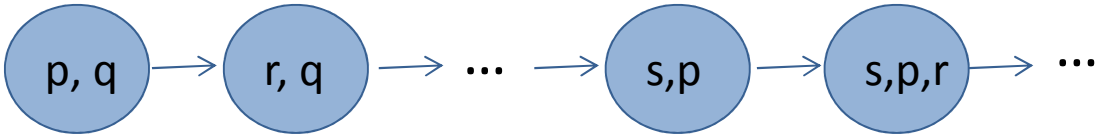
TeDiLog
Model Checking

Trabajo práctico de
Programación Funcional

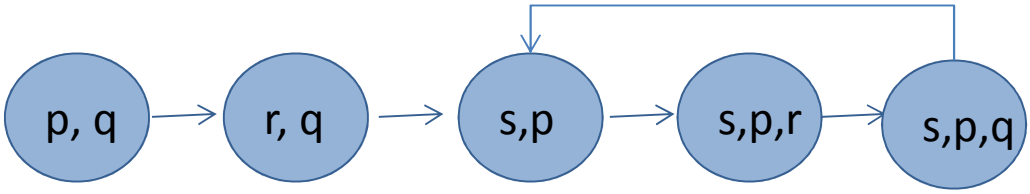
Curso 2012-13

Representación de estructuras (modelos)

Un modelo es una estructura lineal infinita de estados dónde cada estado contiene las proposiciones que son ciertas en él.



Para tratar con TeDiLog, los modelos que nos interesan acaban en un sucesión de estados que se repite infinitamente.



Por tanto, en Haskell un modelo se puede representar como un par de listas donde la segunda lista tiene el ciclo que se repite y la primera la parte inicial que no se repite. En el ejemplo de arriba la primera lista tendría dos estados: [p,q] y [r,q], mientras que la segunda lista tendría tres estados: [s,p], [s,p,r] y [s,p,q]. A efectos de las formulas que satisface una de estas estructuras con un ciclo al final, es una representación finita de la correspondiente estructura infinita (la que repite infinitamente la secuencia de estados del final). Por ejemplo, la estructura finita de arriba representa a la estructura infinita que tiene como estados:

[p,q], [r,q], [s,p], [s,p,r], [s,p,q], [s,p], [s,p,r], [s,p,q], [s,p], [s,p,r], [s,p,q], ...

Definir el tipo Estado como una lista de proposiciones (Strings). Para ganar eficiencia considerar los estados como listas ordenadas (recúerdese que el orden lexicográfico entre strings es predefinido en Haskell).

Definir el tipo Modelo como un par de listas cuyas elementos son estados, como se ha explicado arriba. El ejemplo de arriba sería representado por el par de listas:

([[p,q], [q,r]], [[p,s], [p,r,s], [p,q,s]])

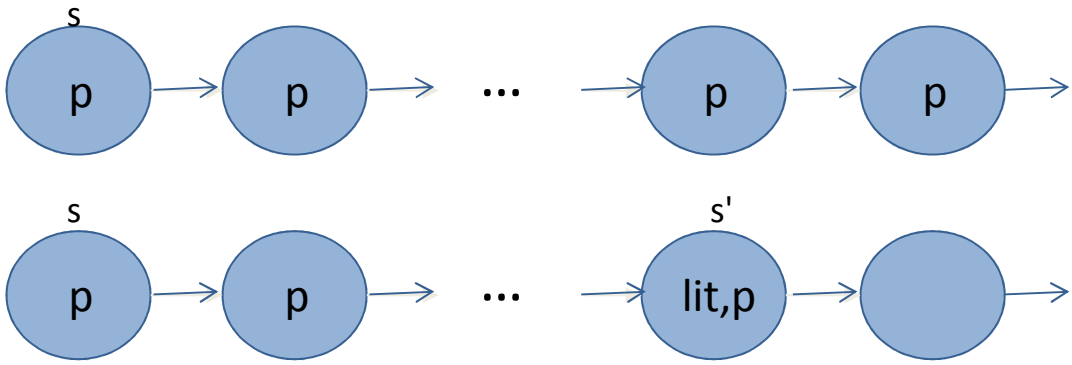
Se puede dar como caso particular que la primera lista sea vacía, o que un estado sea vacío, pero la segunda lista debe contener al menos un estado, aunque sea el estado vacío.

Satisfacción de cláusulas

Un estado s (dentro de una estructura o modelo) satisface

- una proposición p si p pertenece a s.
- un literal positivo (PL p) si p pertenece a s
- un literal negativo (NL p) si p no pertenece a s
- un átomo temporal (U lit p) si hay algún estado s' sucesor de s (puede ser s mismo) que satisface p y todos los estados desde s (incluido) hasta s' (excluido) satisfacen lit.

- un átomo temporal (R lit p) si o bien p se satisface desde s en adelante para siempre, o bien hay algún estado s' a partir de s (incluido) que satisface lit y hasta s' (incluido) se satisface p.



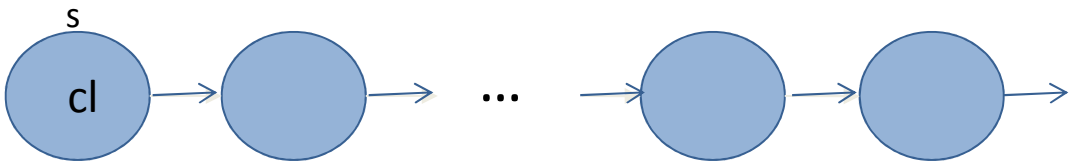
- un átomo temporal (F p) si hay algún estado s' sucesor de s (puede ser s mismo) que satisface p.

- un átomo temporal (G p) si todos los estados a partir de s (incluido) satisfacen p.

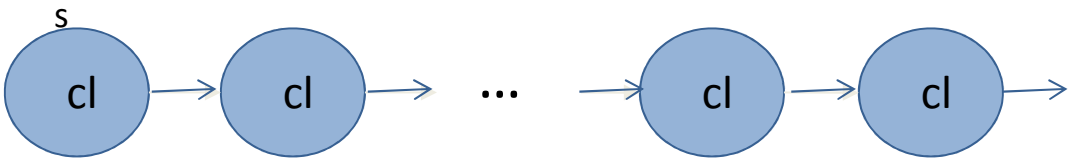
- un átomo (A p) si p pertenece a s.
- un átomo (TA t) si s satisface el átomo temporal t.
- un átomo (X a) si el sucesor (siguiente estado) de s satisface a que es un átomo.

- una now-cláusula $[a_1,...,a_n]$:- $[b_1,...,b_m]$ si o bien s no satisface alguno de los átomos $b_1,...,b_m$, o bien satisface alguno de los $a_1,...,a_n$. Es decir, una now-cláusula se comporta como la implicación lógica $(b_1 \wedge ... \wedge b_m) \rightarrow (a_1 \vee ... \vee a_n)$.

- una cláusula (Now cl) si satisface la now-cláusula cl.



- una cláusula (Alw cl) si todos los estados desde s satisfacen cl.



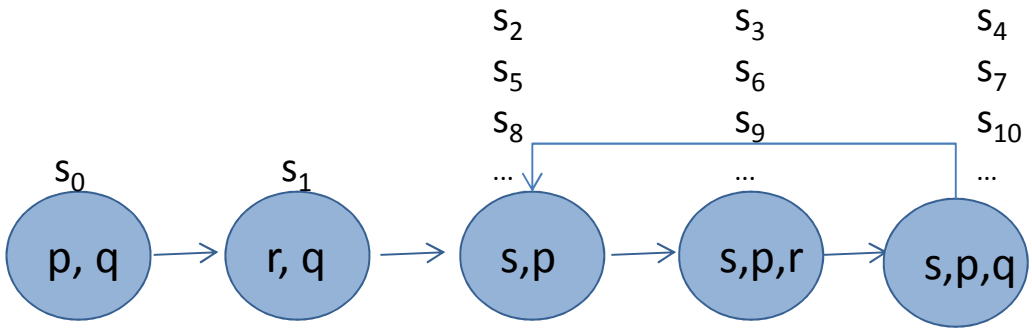
MODELO DE UN PROGRAMA

Una estructura que en su mundo inicial (el primero) satisface todas las cláusulas de un programa se dice que es modelo de dicho programa.

En la fase 2 se trata de definir en Haskell una función que decida si una estructura dada es modelo de un programa dado. Nótese que primero debe cumplir el ser una estructura válida, es decir que su segunda lista no sea vacía. Deberás idear tu propio juego de pruebas y adjuntarlo como documentación.

Ejemplos

En la estructura



que estaría representada por

```
([[p,q], [q,r]], [[p,s], [p,r,s], [p,q,s]])
```

1. p se satisface en todos los estados menos en s_1.
2. s se satisface en todos los estados a partir de s_2, por tanto
   - s_2 cumple el átomo temporal (G s).
   - s_1 cumple X (G s) .
   - s_0 cumple X (X (G s)).
3. (Not s) se satisface en s_0 y s_1.
4. s_9 satisface (X(X(X r))).
5. s_0 cumple el átomo temporal (U q s) y también (F s) .
6. s_0 cumple el átomo temporal (U (Not s) q) y también (F q).
7. s_2 satisface (R (Not q) p) .
8. s_0 satisface (R r q) .
9. s_{10} satisface (F r) y también (U p r).
10. s_0 no satisface (G p).
11. s_2 no satisface (U (Not r) q) .
12. s_1 no satisface (U p s) .
13. s_2 no satisface (R (Not p) r) .
14. s_{10} no satisface (G q) .
15. s_9 no satisface (X(X r)).