Guía para yahc

En el segundo bloque de la materia vamos a trabajar con un programa que comprueba si los pasos de nuestras demostraciones son correctos. Este programa se llama yahc y ha sido desarrollado por Renato Cherini y Miguel Pagano. Pueden ver más información sobre el mismo en http://cs.famaf.unc.edu.ar/~mpagano/yahc/ .

El programa ya se encuentra instalado en las máquinas de los laboratorios. Para instarlo en su casa, sigan las instrucciones de la guía de instalación de yahc en debian.

Para usar yahc, deberán seguir los siguientes pasos:

1. ejecutar yahc. Para empezar, vamos a dar un ejemplo de ejecución en modo interactivo, con el comando `yahc -i`
   ```
   user@dijkstra:~$ yahc -i
   Para ver los posibles comandos y lo que hacen podes hacer "help"
   ```
   > yahc tiene algunos parecidos con la línea de comandos y al intérprete de haskell, por ejemplo, el símbolo ">" que aparece es el prompt, que está esperando que le demos alguna instrucción.
2. dentro de yahc, podemos ejecutar varias instrucciones:
   - ver qué comandos tenemos disponibles, con el comando `help`. Qué nos dice el comando `help`?
   - ver qué teoremas tenemos disponibles, con el comando `show rules`. Qué teoremas tenemos en yahc y cómo los puedo usar?
   - empezar una derivación, con el comando `start`. Esto es lo que vamos a ver ahora.
3. **Para empezar una derivación** usamos el comando `start` y el teorema que queremos demostrar, dividido en dos partes separadas por una coma: de dónde queremos partir y a dónde queremos llegar. Por ejemplo, si queremos demostrar que p ⇒ q ≡ p ∧ q ≡ p , podemos partir de p ⇒ q y llegar a p ∧ q ≡ p . En ese caso, escribiremos: `start p => q , p /\ q == p` , y yahc nos dirá:
   ```
   > start p => q , p /\ q == p
   p => q
   >
   ```
   Pero también podemos demostrar la misma expresión p ⇒ q ≡ p ∧ q ≡ p partiendo de p ⇒ q ≡ p ∧ q y tratando de llegar a p . En ese caso, escribiremos `start p => q == p /\ q , p` , y yahc nos dirá:
   ```
   > start p => q == p /\ q , p
   p => q == p /\ q
   >
   ```
   Ven que los operadores lógicos no se escriben igual en yahc que en el papel? Vean la tabla de equivalencias entre operadores en papel y en yahc. Si no les quedó claro, vean más ejemplos sobre cómo iniciar una derivación.
4. Después de decirle a yahc lo que queremos hacer, nos está mostrando nuestro punto de partida, en este caso, `p => q` .
5. Para hacer una derivación, debemos explicarle a yahc en cada paso qué regla queremos aplicar y qué resultado esperamos que produzca. **Para aplicar una regla** usamos el comando `next`, el nombre de la regla que queremos aplicar y el resultado que esperamos obtener, separados por una coma. Por ejemplo, si a la fórmula inicial le queremos aplicar la propiedad "`Definición de la Implicación`" para que nos quede p ∧ q ≡ q, diremos: `next Definicion de implicacion , p \/ q == q` . Y obtendremos el siguiente resultado: `> next Definicion de implicacion , p \/ q == q`

   ```
   p => q
   ```

```
≡ { Definición de la implicación }

   p v q = q
```

6. Una vez que hemos aplicado esta regla, podemos aplicar otra regla para seguir con la derivación. En nuestro ejemplo, quizás querremos aplicar la Regla Dorada. Para hacerlo, debemos usar el comando `next` junto con el nombre de la regla que queremos aplicar y el resultado que esperamos que produzca. En el ejemplo, lo haremos así: `next Regla Dorada , p /\ q == p` , con lo cual obtendremos `p ∧ q ≡ p`.
7. Terminamos cuando hayamos obtenido la expresión que habíamos marcado como objetivo al empezar la derivación, es decir, el segundo argumento que le hemos dado al comando `start`. En este caso, el segundo argumento era justamente `p /\ q == p`, que es lo que hemos alcanzado al aplicar la regla dorada. En ese momento, yahc nos dice: `Ha concluido la derivación.`, y nos muestra toda la derivación siguiendo a la aplicación de la regla que lleva al último paso de la derivación:

```
> next Regla Dorada , p /\ q == p
   p => q
≡ { Definición de la implicación }
   p v q = q
≡ { Regla dorada }
   p ^ q = p
Ha concluido la derivación.
```

En cualquier momento de la derivación podemos consultar las reglas, usando el comando `show rules`.

En cualquier momento de la derivación podemos consultar la derivación hasta el momento, usando el comando `show derivation`.

Si nos equivocamos al describir el resultado de la aplicación de una regla usando el comando `next`, yahc no la aplicará, y la derivación seguirá estando en el mismo estado en el que estaba antes de escribir el comando `next`.

Si queremos deshacer la última regla que hemos aplicado, podemos usar el comando `undo`. Este comando nos devuelve a la expresión en que estábamos antes de aplicar el paso de derivación inmediatamente anterior.

Cuando ya manejen yahc, pueden usar dos opciones avanzadas: agregar nuevas reglas y agregar nuevas constantes.

Para agregar nuevas reglas, usamos el comando `add`. Este comando añade la última derivación que hemos concluido correctamente como un teorema. Por ejemplo, una vez que terminamos de derivar p ⇒ q ≡ p ∧ q ≡ p, podemos añadir este teorema para usarlo más adelante para otras pruebas. El comando `add` requiere que demos un nombre y abreviatura para añadir la derivación.

Qué nos dice el comando `help`?
Si ponemos "`help`" en el prompt, nos devuelve:

```
> help
```

```
Los comandos existentes son (entre paréntesis las versiones cortas)
  show  (sh)
  help  (h)
  start (s)
  next  (n)
  quit  (q)
  add   (a)
Puedes obtener más información sobre cada comando haciendo "help comando"
>
```

Veamos qué nos dice `help` sobre los diferentes comandos que podemos usar en yahc:

start

```
> help start
start formula formula'
comienza una nueva derivación y descarta
la derivación actual.
```

Es decir, el comando "`start`" sirve para empezar una derivación, o sea, una demostración. Si ya teníamos una demostración empezada, la descarta y empieza una nueva. Para empezar una derivación, debemos darle dos fórmulas, que son, por ejemplo, las dos partes de una equivalencia, si estamos tratando de demostrar una equivalencia. l primer argumento de "`start`" (la primera fórmula de la izquierda) es la fórmula desde la que parte nuestra demostración, y el segundo argumento es la fórmula a la que queremos llegar. Por ejemplo, si queremos demostrar `p v q = q = p => q`, escribiremos `start p \/ q == q , p => q.`

Como habrán notado, los operadores lógicos se escriben de una forma particular en yahc. Vean más abajo la tabla de correspondencia entre operadores lógicos y su forma de escribirlos en yahc.

next

```
> help next
next regla resultado
trata de aplicar la regla
a la conclusión de la derivación actual
si tiene éxito agrega este paso a la
derivación; si no tiene éxito no hace nada.
```

Este comando se usa para aplicar una regla en un paso de derivación. Requiere dos argumentos, separados entre ellos por una coma. El primer argumento es el nombre de la regla a aplicar, que puede ser el nombre completo o cualquiera de las abreviaturas que aparecen en el listado de reglas (que podemos ver con el comando `show rules`). El segundo argumento es el resultado que esperamos de la aplicación de la regla, es decir, cómo esperamos que quede la expresión después de que se aplique la regla. Si nos equivocamos al describir el resultado de la aplicación de una regla usando el comando `next`, yahc no la aplicará, y la derivación seguirá estando en el mismo estado en el que estaba antes de escribir el comando `next` . Un ejemplo de aplicación sería:

```
p => q
> next Definicion de implicacion , p \/ q == q
   p => q
≡ { Definición de la implicación }
   p v q = q
```

show

```
> help show
show [rules | goal | derivation]
```

```
muestran las reglas existentes,
         el objetivo,
         y la derivación hasta ahora, respectivamente.
```

El comando show nos muestra información. Puede mostrarnos diferentes tipos de información:

- **`show rules`** : muestra el listado con las reglas que tenemos disponibles para aplicar, junto con sus formas abreviadas. Podemos usar cualquiera de los nombres de una regla, completo o abreviado, para usarla en el comando `next`.
- **`show goal`** : muestra la expresión a la que pretendemos llegar con la derivación, es decir, la expresión que hemos dado como segundo argumento del comando `start`.
- **`show derivation`** : muestra el punto de partida de la derivación y todos los 2 pasos que se han aplicado hasta el momento.

Cualquiera de estos comandos `show` se puede usar en cualquier momento de la derivación, sin que eso afecte para nada al estado de la derivación. Por ejemplo, podemos estar haciendo una derivación, hacer un `show rules` porque no recordamos el nombre de alguna regla, ver las reglas, y luego, para recordar bien cómo iba nuestra derivación, hacer `show derivation` y recién ahí aplicar la regla que queremos con `next NOMBRE-REGLA , EXPRESIÓN.`

Qué teoremas tenemos en yahc y cómo los puedo usar?

Los teoremas que tenemos en yahc son:

```
Asociatividad Equivalencia [A =,A Eq]
Conmutatividad Equivalencia [C =,C Eq]
Neutro equivalencia [N =,N Eq]
Definicion de negacion [D -,D Neg]
Definicion de false [D F,D False]
Definicion de discrepancia [D =\=,D Disc]
Idempotencia disyuncion [I \/,I Disy]
Asociatividad Disyuncion [A \/,A Disy]
Conmutatividad Disyuncion [C \/,C Disy]
Distributividad de disyuncion con la equivalencia [D \/ =,D Disy Eq]
Tercero excluido [TE]
Regla dorada [RD]
Definicion de implicacion [D =>,D Impl]
Definicion de consecuencia [D <=,D Cons]
Doble negacion [--,Neg Neg]
Equivalencia y negacion [= -,Eq Neg]
Absorbente de la disyuncion [EA \/,EA Disy]
Neutro de la disyuncion [N \/,N Disy]
Teorema (*) [T *]
```

Y se aplican usando el comando `next`, como su primer argumento. Se puede usar la forma extendida o cualquiera de sus abreviaciones.

Por ejemplo, si quiero aplicar el teorema estrella a (r ≡ s) ∨ (p ⇒ (q ∨ r)) para obtener (r ≡ s) ∨ ¬(p ⇒ (q ∨ r)) ≡ (r ≡ s), puedo hacerlo así:

```
next Teorema (*) , ( r == s ) \/ -(p => (q \/ r)) == (r == s)
```

o bien así:
```
next T * , ( r == s ) \/ -(p => (q \/ r)) == (r == s)
```

O por ejemplo, si quiero aplicar la regla dorada para convertir p v q en la expresión p ∧ q ≡ p ≡ q, puedo hacerlo así:
```
next Regla dorada , p /\ q == p == q
```
o bien así:
```
next RD , p /\ q == p == q
```

Ejemplos sobre cómo empezar una derivación

p v (q v r) ≡ (p v q) v (p v r)
start p \/ (q \/ r) , (p \/ q) \/ (p \/ r)

(p ≢ (q ≢ r)) ≡ ((p ≢ q) ≢ r)
start p =/= (q =/= r) , (p =/= q) =/= r

p v true ≡ true
start p \/ True , True

¬(p v q) ≡ ¬p ∧ ¬q
start -(p \/ q) , -p /\ -q

Cómo se escriben los distintos operadores lógicos en yahc?

≡ ... **==**
≢ ... **=/=**
⇒ ... **=>**
v ... \/
∧ ... /\
¬ ... -
true ... True
false ... False