**Modelos de Computación (2015-2016)**
Grado en Ingeniería Informática
Universidad de Granada

# Memoria Proyecto de Lex

Laura Tirado López

November 29, 2015

# Contents

# 1 Descripción del problema

El problema a resolver consiste en dada una expresión de lógica proposicional reducirla de forma que las únicas conectivas que intervengan en la expresión sean $\vee$ y $\wedge$, además de la negación ($\neg$). Para poder reducirla, hay que aplicar las transformaciones de las posibles conectivas que intervenga en la fórmula, como la equivalencia ($\leftrightarrow$) o la implicación ($\rightarrow$). Un ejemplo de reducción sería dada la fórmula $(p \leftrightarrow q) \wedge (\neg p \rightarrow q)$ su forma reducida sería $((\neg p \vee q) \wedge (\neg q \vee p)) \wedge (p \vee q)$.

# 2 Solución planteada

Para reducir una expresión lógica, hay que realizar la siguientes transformaciones a la fórmula:

- Si tenemos una subfórmula de la forma $\alpha \leftrightarrow \beta$, la sustituimos por $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$.
- Si tenemos una subfórmula de la forma $\alpha \rightarrow \beta$, la sustituimos por $\neg\alpha \vee \beta$.
- Cualquier subfórmula de la forma $\neg\neg\alpha$ la sustituimos por $\alpha$.
- Introducir la conectiva $\neg$ dentro de los paréntesis utilizando las equivalencias lógicas:

  - $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
  - $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$

- Si al hacer este último paso nos apareciera una nueva subfórmula $\neg\neg\alpha$, la sustituimos por $\alpha$.

Para cada transformación que tenemos que hacer para reducir la fórmula, he implementado código en Lex para reconocer la expresión regular y en C para transformar la expresión reconocida. Por tanto, tenemos cuatro arhivos:

- `equivalencia.l`: Comprueba si hay o no equivalencias en las fórmulas dadas. Si encuentra una equivalencia en alguna fórmula la sustituye aplicando la primera regla mencionada haciendo una llamada a la función `sustituir_equivalencia` y muestra el resultado en pantalla. En caso de no encontrar ninguna equivalencia, muestra las fórmulas en pantalla sin ninguna modificación mediante la función `imprime`. Para controlar si se ha realizado la sustitución de las equivalencias o no, utilizamos una variable entera `equivalencia` inicializada por defecto a 0. Si su valor es 0, significa que la fórmula no tenía equivalencias que sustituir por lo que la función `imprime` imprimirá las fórmulas sin ninguna modificación. Si el valor de `equivalencia` es 1, significa que las equivalencias de la fórmula han sido sustituidas. La función `sustituir_equivalencia` recibe como parámetro la fórmula en

la que queremos sustituir las equivalencias y calcula mediante una función auxiliar (`subcadena`) la posición del signo de equivalencia en la cadena. A partir de la posición del signo de equivalencia, calculamos las expresiones $\alpha$ y $\beta$ y sus posiciones de inicio y fin. Una vez tenemos todo esto calculado ya podemos sustituir la equivalencia por dos implicaciones. El proceso se repite hasta que no queda ninguna equivalencia en la fórmula. Por último, modifica el valor de la variable `equivalencia` a 1 y se imprime el resultado final.

- `implicacion.l`: Comprueba si hay o no implicaciones en las fórmulas dadas. Si encuentra alguna implicación, la sustituye aplicando la segunda regla mediante la función `sustituir_implicacion` y muestra el resultado en pantalla. En caso de no encontrar ninguna implicación, muestra las fórmulas en pantalla sin ninguna modificación mediante la función `imprime`. Para controlar si se ha realizado la sustitución de las implicaciones o no, utilizamos una variable entera `implicacion` inicializada por defecto a 0. Si su valor es 0, significa que la fórmula no tenía implicaciones que sustituir por lo que la función `imprime` imprimirá las fórmulas sin ninguna modificación. Si el valor de `implicacion` es 1, significa que las implicaciones de la fórmula han sido sustituidas. La función `sustituir_equivalencia` recibe como parámetro la fórmula en la que queremos sustituir las implicaciones y calcula mediante una función auxiliar (`subcadena`) la posición del signo de implicación en la cadena. A partir de la posición del signo de implicación, calculamos las expresiones $\alpha$ y $\beta$ y sus posiciones de inicio y fin. Una vez tenemos todo esto calculado ya podemos sustituir la implicación. El proceso se repite hasta que no queda ninguna implicación en la fórmula. Por último, modifica el valor de la variable `implicacion` a 1 y se imprime el resultado final.

- `negacion.l`: Comprueba si hay o no dobles negaciones en las fórmulas dadas. Si encuentra alguna doble negación, la sustituye aplicando la tercera regla mediante la función `sustituir_doble_negacion` y muestra el resultado en pantalla. En caso de no encontrar ninguna, muestra las fórmulas en pantalla sin ninguna modificación mediante la función `imprime`. Para controlar si se han eliminado las dobles negaciones o no, utilizamos una variable entera `negacion` inicializada por defecto a 0. Si su valor es 0, significa que la fórmula no tenía dobles equivalencias por lo que la función `imprime` imprimirá las fórmulas sin ninguna modificación. Si el valor de `negacion` es 1, significa que las dobles negaciones de la fórmula han sido eliminadas. La función `sustituir_doble_negacion` recibe como parámetro la fórmula en la que queremos eliminar las dobles negaciones. El funcionamiento es simple, recorre toda la cadena y la va copiando en una variable auxiliar. Si encuentra dos signos de negación seguidos se los salta y sigue copiando. Por último, modifica el valor de la variable `negacion` a 1 y se imprime el resultado final.

- `parentesis-negado.l`: Introduce la conectiva $\neg$ dentro de los paréntesis mediante la función `introducir_negativa` y muestra el resultado en pantalla. En caso de no encontrar ningún caso, muestra las fórmulas en pantalla sin ninguna modificación mediante la función `imprime`. Para controlar si se han introducido las negaciones

en los paréntesis o no, utilizamos una variable entera `neg_parentesis` inicializada por defecto a 0. Si su valor es 0, significa que la fórmula no tenía ninguna negación que introducir por lo que la función `imprime` imprimirá las fórmulas sin ninguna modificación. Si el valor de `neg_parentesis` es 1, significa que las negaciones han sido introducidas en los paréntesis. La función `introducir_negativa` recibe como parámetro la fórmula en la que queremos introducir las negaciones en los paréntesis y calcula mediante una función auxiliar (`subcadena`) la posición del signo de negación que tenemos que introducir en el paréntesis. A partir de la posición de éste, calculamos las expresiones $\alpha$ y $\beta$ y sus posiciones de inicio y fin. Una vez tenemos todo esto calculado ya podemos introducir la negación en el paréntesis. El proceso se repite hasta que no queda ninguna equivalencia en la fórmula. Por último, modifica el valor de la variable `neg_parentesis` a 1 y se imprime el resultado final.

Para conectar las distintas modificaciones que se hacen a las fórmulas, implementé un script `reducir.sh` para conectar los distintos archivos de entrada y salida. La estructura del script es la siguiente:

- Creamos el ejecutable de `equivalencia.l`. Lo ejecutamos paśandole como parámetro el archivo con las fórmulas que queremos reducir e indicamos un archivo para mostrar la salida.
- Creamos el ejecutable de `implicacion.l`. Lo ejecutamos paśandole como paŕametro el archivo creado a partir de ejecutar el código de `equivalencia.l` e indicamos un archivo para mostrar la salida.
- Creamos el ejecutable de `negacion.l`. Lo ejecutamos paśandole como paŕametro el archivo creado a partir de ejecutar el código de `implicacion.l` e indicamos un archivo para mostrar la salida.
- Creamos el ejecutable de `parentesis-negado.l`. Lo ejecutamos paśandole como paŕametro el archivo creado a partir de ejecutar el código de `negacion.l` e indicamos un archivo para mostrar la salida.
- Por último, volvemos a ejecutar el código de `negacion.l` paśandole como entrada el archivo creado a partir de la ejecución de `parentesis-negado` e indicamos un archivo para mostrar la salida.

En el último archivo de salida generado, se encuentra la foŕmula reducida.