3 Parte 3

3.1 Respuesta 1

Sean Π y Π' programas tal que $\Pi \subseteq \Pi'$, y sea M modelo de Π. Tenemos que para cada regla $r_i \in \Pi$ de la forma $Head_i \leftarrow Body_i$, donde $Body_i$ no contiene ninguna negación, y $|Head_i| = 1$, si $Body_i \subseteq M$, entonces $Head_i \cap M \neq \emptyset$. Dado que $|Head_i| = 1$, también se cumple que $Head_i \subseteq M$. También, sea M' modelo de Π'. Para cada regla $r_j \in \Pi'$ de la forma $Head_j \leftarrow Body_j$, si $Body_j \subseteq M'$, entonces $Head_j \subseteq M'$.

Queremos demostrar que $M \subseteq M'$. Es decir, para cada átomo x, si $x \in M$, entonces $x \in M'$.

Para cada regla $r_i \in \Pi$, se cumple que $Head_i \subseteq M$ sí, y solo sí, $Head_i$ es la cabeza de alguna regla r_i tal que $Body_i \subseteq M$. Para aquellas reglas en las cuales se cumple que $|Body_i| = 0$, siempre se cumple que $Head_i \subseteq M$.

Tenemos que por definición de Π', si la regla r pertenece a Π, entonces también pertenece a Π'. Entonces, para alguna regla $r_i \in \Pi$ de la forma $Head_i \leftarrow Body_i$, si $Body_i \subseteq M$, entonces $Body_i \subseteq M'$, y por lo tanto $Head_i \subseteq M$ y $Head_i \subseteq M'$. De esta manera, se cumple el principio de monotonía para programas de lógica sin negación con $|Head_i| = 1$ para toda r_i, y $M \subseteq M'$. $\square$

3.2 Respuesta 2

Queremos demostrar que todo programa que tiene reglas de la forma $Head \leftarrow Body$, sin negación, y con $|Head| \leq 1$, tiene a lo más un modelo. Por inducción, supongamos que un programa Π con reglas de la forma $Head \leftarrow Body$, sin negación, y con $|Head| \leq 1$, y una cantidad de reglas igual a n, tiene a lo más un modelo.

Caso base: Para un programa que tiene 0 reglas, se tiene que no hay ninguna regla de la forma $Head$ (con $|Body| = 0$), por lo que inicialmente no se puede añadir ningún átomo al modelo M del programa. Por lo tanto un programa con $n = 0$ reglas tiene un único modelo, el modelo vacío $M = \emptyset$, y se cumple que tiene a lo más un modelo. Para un programa que solamente una regla r de la forma $Head \leftarrow Body$ (es decir, $n = 1$), se pueden dar los siguientes casos:

1. $|Head| = 1$, y $|Body| = 0$: En este caso, la regla constituye a declarar que un átomo es verdadero. Dado que r es la única regla, $Head$ sería el único modelo del programa.

2. $|Head| = 0$, y $|Body| > 0$: En este caso, $Head$ está vacío, por lo que esta regla excluye átomos del modelo en los cuales se cumple la condición de $Body$. Dado que es la única regla, no se puede cumplir $Body$, por lo que el modelo es vacío.

3. $|Head| = 1$, y $|Body| > 0$: En este caso, $Head$ es parte del modelo $Body$. Dado que es la única regla del programa, no se puede cumplir $Body$, por lo que el modelo es vacío.

Notar que el caso $|Body| = 0$, y $|Head| = 0$, no constituye una regla válida. De esta manera, los casos anteriores cubren todas las posibilidades, y siempre se cumple que un programa con $n = 1$ reglas tiene un modelo.

Por hipótesis inductiva, sabemos que se cumple que un programa con $n = k$ reglas las cuales cumplen las condiciones establecidas tiene a lo más un modelo.

Debemos demostrar que esto se cumple para programa con $n = k+1$ reglas. Sea Π' un programa con $n = k+1$ reglas. Π' corresponde a la unión de un programa Π con $n = k$ reglas, y un programa

$\{r\}$ con una regla ($n = 1$). Es decir $\Pi' = \Pi \cup \{r\}$. Sabemos que tanto Π como $\{r\}$ tienen a lo más un modelo. Para Π, se pueden dar dos casos.

1. Que Π no tenga modelo: En este caso, se deduce que entre las reglas de Π existe una contradicción. Si añade la regla r a Π, esta contradicción no se eliminaría, por lo que Π' tampoco tendría modelo.

2. Que Π tenga un modelo M: En este caso, añadir r no aumentaría la cantidad de modelos. En el caso de que r genere una contradicción, Π' no tendría ningún modelo. Para el caso en que no genere una contradicción, si no se cumple $Body_r$ de r, entonces el modelo de Π se mantendría en M. Por otro lado, si se cumple $Body_r$, entonces el nuevo modelo de Π' crecería a $M \cup Head_r$, y según la definición de modelo habría que iterar sobre las reglas de Π para determinar si alguna otra regla se cumple. Aquí, solo cambiaría el tamaño del modelo de Π', pero no aumentaría su cantidad de modelos.

Dado que en ninguno de los casos anteriores la cantidad de modelos resulta ser mayor a 1, se cumple que un programa con $n = k + 1$ reglas tiene a lo más un modelo.

De esta manera, demostramos por inducción que todos los programas que cumplen las condiciones del enunciado siempre tienen a lo más un modelo. □

3.3 Respuesta 3

Sea `calculate_models` la rutina que ejecuta el algoritmo descrito en el enunciado, y recibe como parámetro un programa (set de reglas).

```
function calculate_models_neg(program)
    // Obtener atomos posibles que pueden tener los modelos
    let atoms = set()
    for each (rule in program) do
        atoms = atoms + rule.Head
    end for each
    // Definir parte recursiva del algoritmo
    function recursive_part(program_models, program, possible_model)
        let new_program = copy(program)
        // reducir (quitar los not del programa según el modelo posible)
        for each (rule in new_program) do
            for each (atom in rule.Body.Neg) do
                if (atom in possible_model) then
                    // quitar la regla de la copia del programa
                    program.remove(rule)
                    continue
                else then
                    // borrar el átomo de la regla (se asume not atom = true)
                    rule.Body.Neg.remove(atom)
                end if
            end foreach
        end for each
        // Calcular el modelo del programa reducido
        calculated_model = calculate_models(new_program)
        if (calculated_model == possible_model) then
            program_models.add(possible_model)
        end if
        // volver a ejecutar el algoritmo con todos los submodelos posibles
        for each (atom in possible_model) do
            recursive_part(program_models, program, possible_model - {atom})
        end for each
    end function
    // Ejecutar rutina anterior (parte recursiva)
    let program_models = set()
    recursive_part(program_models, program, atoms)
    // retornar set de modelos del programa
    return program_models
end function
```

Ejemplos significativos:

1. Para un programa vacío (sin reglas), el flujo no entra a ningún ciclo, y simplemente llega a `calculate_models`, pasando como parámetro el programa vacío, y la subrutina entrega como resultado un modelo vacío, el cual se añade a `program_models`. La función finalmente retorna solo un modelo vacío, lo cual es correcto.

2. Para un programa con tres reglas: `a.`, `b.`, y `c.`, se comienza con el modelo posible: $\{a, b, c\}$. Debido a que no hay ningún átomo negado, simplemente se pasa el programa original a `calculate_models`, el cual da como resultado el modelo $\{a, b, c\}$, y dado que es igual al modelo posible, se agrega a `program_models`. Posteriormente, se hace recursión con los siguientes submodelos: $\{b, c\}, \{a, c\}, \{a, b\}$. Nuevamente no se modifica el programa (la reducción no tiene efecto, dado que no hay átomos negados), por lo que `calculate_models` da la misma respuesta. A continuación la recursión calcula los siguientes modelos (notar que se repiten): $\{c\}, \{b\}, \{c\}, \{a\}, \{b\}, \{a\}$. En estos casos sucede lo mismo que en las iteraciones anteriores. Finalmente al llegar a la recursión, se vuelve a ejecutar la rutina con el modelo vacío {} como modelo posible 6 veces. Finalmente, el programa solo retorna un modelo, $\{a, b, c\}$, el cual es correcto.

3. Para un programa con las reglas `a :- not b` y `b :- not a`, se comienza con el modelo $\{a, b\}$ como modelo posible. Dado que a está en el modelo, se quita la regla `b :- not a` del programa, según la línea 15 del pseudocódigo. Sucede lo mismo para la primera regla, quedando el programa vacío. `calculate_models` retorna vacío, y dado que es distinto a $\{a, b\}$, no es válido como modelo. Para la recursión, se forman los modelos posibles $\{a\}$ y $\{b\}$. Para $\{a\}$, se elimina la regla `b :- not a`, y se elimina el átomo `not b` de la regla `a :- not b`, quedando como `a.`. De esta manera `calculate_models` retorna $\{a\}$ como resultado, y dado que es igual al modelo posible, se acepta como modelo del programa original. Análogamente para $\{b\}$, se acepta a $\{b\}$ como modelo posible. En la recursión siguiente, se toma como modelo posible al modelo vacío. Dado que no tiene ningún átomo, se eliminan los átomos negados de las reglas del programa según la línea 19 del pseudocódigo, quedando las reglas `a.` y `b.`, y la subrutina retorna como modelo a $\{a, b\}$, lo cual es distinto de vacío, por lo que no se acepta como modelo posible. Al final, la función retorna los modelos $\{a\}$ y $\{b\}$, lo que es correcto.

4. Para un programa con las reglas `a :- b` y `b :- not a`, se comienza con el modelo $\{a, b\}$ como modelo posible. Dado a está en el modelo, se quita la regla `b :- not a`, quedando solo la primera. `calculate_models` retorna vacío, y dado que este es distinto a $\{a, b\}$, no es válido como modelo. Para la recursión, se forman los modelos posibles $\{a\}$ y $\{b\}$. Para $\{a\}$, se elimina la regla `b :- not a`, quedando solamente la primera, por lo que `calculate_models` retorna $\{\} \neq \{a\}$ como resultado. Por otro lado, para $\{b\}$, se elimina el átomo `not a` de la segunda regla, quedando las reglas `a :- b` y `b.`. `calculate_models` retorna $\{a, b\} \neq \{b\}$, por lo que no se acepta el modelo. Finalmente, se toma como modelo posible al modelo vacío, y ocurre un caso similar al del modelo $\{b\}$, retornando la subrutina $\{a, b\} \neq \{\}$. La función termina sin ningún modelo posible, por lo que el programa es insatisfacible, lo cual es correcto.