Departamento de Matemática y Ciencia de la Computación

Tarea 1 - SAT Lineal

Miguel Olivares Morales
miguel.olivares@usach.cl

Benjamín Riveros Landeros
benjamin.riveros.l@usach.cl

Lógica Computacional - 22625
Licenciatura en Ciencia de la Computación

Semestre Otoño 2025

1 Introducción

El problema de determinar si las variables de una fórmula booleana pueden ser reemplazadas con valores **T** o **F** de tal forma que la fórmula de como resultado **T** se denomina problema de satisfacibilidad booleana o SAT. Si al evaluar la fórmula esta da como resultado **T**, entonces se dice que es satisfactoria.

2 Procedimiento

Las fórmulas que serán analizadas primero tendrán que ser codificadas según la siguiente gramatica:

$$\phi ::= p \mid (\neg\phi) \mid (\phi \wedge \phi)$$

Para esto usamos el siguiente esquema de traducción:

$T(p) = p$

$T(\neg\phi) = \neg T(\phi)$

$T(\phi_1 \wedge \phi_2) = T(\phi_1) \wedge T(\phi_2)$

$T(\phi_1 \vee \phi_2) = \neg(\neg T(\phi_1) \wedge \neg T(\phi_2))$

$T(\phi_1 \to \phi_2) = \neg(T(\phi_1) \wedge \neg T(\phi_2))$

Esto quiere decir que se analizarán fórmulas compuestas por proposiciones atómicas, negaciones de otras fórmulas y conjunciones de dos fórmulas.

Luego de codificar se tiene que transformar a su notación postfix o también llamada *notación polaca inversa* con la cual facilitará la creación de un *parse tree* para asignar valores **T** o **F** a cada nodo. Al tener el parse tree correspondiente a la fórmula que se evalúa asignamos **T** al nodo que encabeza el árbol. Esto implica asumir que la fórmula completa es verdadera y a partir de ello se puede extender esta asignación hacia los nodos hijos del árbol aplicando reglas semánticas de los conectores lógicos.

Si el nodo principal es una conjunción $\phi \wedge \psi$ entonces ϕ y ψ deben ser verdaderas. Por el contrario, si el nodo es una negación $\neg\phi$ quiere decir que la subfórmula ϕ es falsa. Este procedimiento se aplica recursivamente hasta llegar a los nodos hoja, los cuales corresponden a átomos proposicionales.

De esta forma se obtiene una asignación de valores de verdad que satisface la fórmula. En caso que las asignaciones conduzcan a una contradicción (por ejemplo, se tiene $p \equiv \mathbf{T}$ y $\neg p \equiv \mathbf{T}$) se descarta el camino recorrido o incluso puede significar que la fórmula es *insatisfacible*.

Adicionalmente para una mayor eficiencia en espacio y tiempo, detectar y reutilizar átomos proposicionales podemos construir en cambio un DAG (Directed Acyclic Graph).

2.1 Ejemplo

Dada la siguiente fórmula:

$$((p \to q) \wedge (\neg r \vee p))$$

El primer paso es aplicar la codificación mencionada anteriormente

$$\begin{aligned}\phi &= ((p \to q) \land (\neg r \lor p))\\ T(\phi) &= T(((p \to q) \land (\neg r \lor p)))\\ &= T(p \to q) \land T(\neg r \lor p)\\ &= \neg(T(p) \land \neg T(q)) \land \neg(\neg T(\neg r) \land \neg T(p))\\ T(\phi) &= \neg(p \land \neg q) \land \neg(\neg\neg r \land \neg p)\end{aligned}$$

El siguiente paso es transformar la fórmula codificada a su notación postfix, para esto hay que descomponer la fórmula en *tokens* de la siguiente manera:

$$[\neg, (, p, \land, \neg, q,), \land, \neg, (, \neg, \neg, r, \land, \neg, p,)]$$

Para transformas a su notación postfix se tiene que saber que se evalúan los operadores según la precedencia, en donde la negación tiene la mayor precedencia por lo que se evalúa primero y despúes la conjunción.

Token	**Acción**	**Salida**	**Stack**
$\neg$	Apilar operador		$\neg$
(	Apilar paréntesis		$\neg$, (
p	Agregar a salida	p	$\neg$, (
$\land$	Apilar operador	p	$\neg$, (, $\land$
$\neg$	Apilar operador	p	$\neg$, (, $\land$, $\neg$
q	Agregar a salida	$p\ q$	$\neg$, (, $\land$, $\neg$
)	Desapilar hasta (	$p\ q\ \neg\ \land$	$\neg$
$\land$	Apilar operador	$p\ q\ \neg\ \land$	$\neg$, $\land$
$\neg$	Apilar operador	$p\ q\ \neg\ \land$	$\neg$, $\land$, $\neg$
(	Apilar paréntesis	$p\ q\ \neg\ \land$	$\neg$, $\land$, $\neg$, (
$\neg$	Apilar operador	$p\ q\ \neg\ \land$	$\neg$, $\land$, $\neg$, (, $\neg$
$\neg$	Apilar operador	$p\ q\ \neg\ \land$	$\neg$, $\land$, $\neg$, (, $\neg$, $\neg$
r	Agregar a salida	$p\ q\ \neg\ \land\ r$	$\neg$, $\land$, $\neg$, (, $\neg$, $\neg$
$\land$	Apilar operador	$p\ q\ \neg\ \land\ r$	$\neg$, $\land$, $\neg$, (, $\neg$, $\neg$, $\land$
$\neg$	Apilar operador	$p\ q\ \neg\ \land\ r$	$\neg$, $\land$, $\neg$, (, $\neg$, $\neg$, $\land$, $\neg$
p	Agregar a salida	$p\ q\ \neg\ \land\ r\ p$	$\neg$, $\land$, $\neg$, (, $\neg$, $\neg$, $\land$, $\neg$
)	Desapilar hasta (	$p\ q\ \neg\ \land\ r\ p\ \neg\ \land\neg\neg$	$\neg$, $\land$, $\neg$
	Vaciar pila	$p\ q\ \neg\ \land\ r\ p\ \neg\ \land\neg\ \neg\ \neg\ \land\ \neg$	

3 Algoritmo

$(p_1 \to p_2) \lor (p_3 \land p_4) \to (p_5 \to p_6) \lor (p_7 \land p_8) \to (p_9 \to p_{10}) \lor (p_{11} \land p_{12}) \to (p_{13} \to p_{14}) \lor (p_{15} \land p_{16}) \to (p_{17} \to p_{18}) \lor (p_{19} \land p_{20})$

4 Implementación

5 Conclusiones