Facsímil 02

Inteligencia clásica

Capítulo 05

SAT y CSP: la IA como restricciones

686f6c61 · 2026

Capítulo 05: SAT y CSP: la IA como restricciones

Entrando en el tema

Imagina que tienes que organizar una semana de reuniones. Hay salas, franjas horarias, personas que no pueden coincidir, permisos, duraciones distintas y una regla sencilla: la agenda final no puede tener solapes. Un modelo generativo puede proponer una agenda bonita. Pero bonita no significa válida.

Aquí aparece una idea muy antigua y muy viva de la inteligencia artificial: no todo consiste en generar una respuesta; a veces consiste en encontrar una asignación que cumpla reglas exactas.[1] Si una sala no puede estar en dos reuniones a la vez, esa regla no es una sugerencia. Es una restricción.

SAT y CSP son dos formas clásicas de expresar ese tipo de problema. SAT trabaja con variables booleanas: verdadero o falso. CSP trabaja con variables que pueden tomar valores de dominios más ricos: horas, salas, personas, rutas, configuraciones. En ambos casos, la pregunta central es la misma: ¿existe una solución que cumpla todas las reglas?

No estamos buscando texto plausible

El malentendido habitual es pensar que SAT y CSP son algoritmos viejos para problemas académicos. En realidad, son una forma de disciplina. Te obligan a separar tres cosas que conviene no mezclar:

CAPA	QUÉ HACE	EJEMPLO
Interpretar	Entender una petición ambigua.	“Busca una agenda para el equipo esta semana”.
Proponer	Construir candidatos posibles.	Tres horarios alternativos.
Verificar	Aceptar solo lo que cumple reglas.	Sin solapes, sala disponible, permisos correctos.

Un LLM puede ayudar en las dos primeras capas. Puede leer lenguaje natural, resumir preferencias y proponer candidatos. Pero la tercera capa debería ser verificable: un solver, un validador, un esquema JSON, una política de permisos, una regla de negocio o una combinación de todo eso.

La lección del capítulo es esta: cuando hay reglas duras, la aceptación no debe depender de si la respuesta suena convincente.

Antes de entrar en la notación, quédate con dos imágenes sencillas:

PROBLEMA COTIDIANO	CÓMO LO VE SAT O CSP	PREGUNTA QUE RESPONDE
Activar o no activar opciones de una campaña.	SAT lo ve como interruptores: email sí/no, banner sí/no, aprobación legal sí/no.	¿Hay alguna combinación de interruptores compatible con todas las reglas?
Colocar reuniones en una agenda.	CSP lo ve como huecos que hay que rellenar: reunión 1 a las 9:00 o 10:00, reunión 2 a las 9:00 o 10:00.	¿Qué valor pongo en cada hueco para que no haya conflictos?
Aceptar una acción de un agente.	Un validador lo ve como una puerta: pasa si cumple permisos, formato, coste y estado.	¿Esta acción se puede ejecutar o debe rechazarse?

La matemática que viene ahora formaliza esa idea. SAT habla de interruptores. CSP habla de huecos con valores posibles. Los validadores hablan de puertas que se abren o se cierran.

SAT: verdadero, falso y contradicción

SAT, abreviatura de satisfacibilidad booleana, pregunta si existe una asignación de valores verdadero/falso que hace verdadera una fórmula lógica. Cook demostró en 1971 que SAT es NP-completo, convirtiéndolo en una piedra angular de la teoría de la complejidad.[2] Hoy SAT sigue siendo práctico porque los solvers modernos explotan estructura, propagación y aprendizaje de cláusulas.[3]

Piensa en una campaña muy simple. El equipo quiere publicar una oferta, pero hay reglas:

VARIABLE	SIGNIFICADO COTIDIANO	VALOR POSIBLE
A	Enviar la oferta por email.	Sí o no.
B	Mostrar la oferta como banner en la app.	Sí o no.
C	Tener el texto aprobado por legal.	Sí o no.

Las reglas son igual de simples:

1. La oferta debe salir por al menos un canal: email o banner.

2. Si sale por email, el texto debe estar aprobado.
3. Si sale por banner, el texto debe estar aprobado.

SAT convierte esas frases en lógica booleana. No está escribiendo la campaña. Solo comprueba si existe una combinación que respete las reglas.

La forma canónica se llama CNF: una conjunción de cláusulas. Cada cláusula es una disyunción de literales.

$$\varphi = \bigwedge_{j=1}^{m} C_j, \qquad C_j = \bigvee_{\ell \in L_j} \ell, \qquad \ell \in \{x_i, \neg x_i\}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
φ	Fórmula booleana completa que queremos satisfacer.	$(A \vee B) \wedge (\neg A \vee C)$.
C_j	Cláusula número j. Debe ser verdadera.	$C_1 = (A \vee B)$.
m	Número total de cláusulas.	$m = 3$.
L_j	Conjunto de literales dentro de la cláusula j.	$L_1 = \{A, B\}$.
ℓ	Literal: variable afirmada o negada.	A o $\neg A$.
x_i	Variable booleana.	$A = \text{verdadero}$: se envía email.

Una asignación es una función que da valor a cada variable:

$$\alpha : X \to \{0, 1\}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
α	Asignación concreta de valores.	$\alpha(A) = 1, \alpha(B) = 0, \alpha(C) = 1$.
X	Conjunto de variables booleanas.	$X = \{A, B, C\}$.
$\{0, 1\}$	Dominio booleano: falso o verdadero.	$0 = \text{falso}, 1 = \text{verdadero}$.

La fórmula es satisfacible si existe al menos una asignación que hace verdaderas todas las cláusulas:

$$\exists \alpha \; \forall j \in \{1, \ldots, m\} : C_j(\alpha) = 1$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$\exists \alpha$	Existe una asignación.	Probar $A = 1, B = 0, C = 1$.
$\forall j$	Para toda cláusula.	C_1, C_2, C_3 deben cumplirse.
$C_j(\alpha)$	Valor de la cláusula j bajo la asignación α.	$C_2(\alpha) = 1$.
1	Verdadero.	La cláusula queda satisfecha.

Veámoslo con la campaña anterior. Tomemos:

$$\varphi = (A \vee B) \wedge (\neg A \vee C) \wedge (\neg B \vee C)$$

La primera cláusula dice “usa email o banner”. La segunda dice “si usas email, legal debe estar aprobado”. La tercera dice “si usas banner, legal debe estar aprobado”.

Probamos la asignación $\alpha(A) = 1$, $\alpha(B) = 0$, $\alpha(C) = 1$:

CLÁUSULA	SUSTITUCIÓN	RESULTADO
$A \vee B$	$1 \vee 0$	1
$\neg A \vee C$	$0 \vee 1$	1
$\neg B \vee C$	$1 \vee 1$	1

Leído en castellano: enviamos email, no mostramos banner y legal sí ha aprobado el texto. Todas las cláusulas valen 1. Por tanto, la fórmula es SAT y α es un modelo. Si ninguna asignación de las $2^3 = 8$ posibles funcionara, la fórmula sería UNSAT.

Cómo se pasa de frase a cláusula

La parte que más se suele subestimar no es ejecutar un solver, sino **codificar bien el problema**. Un solver SAT no entiende “legal debe aprobar si hay campaña”. Entiende literales y cláusulas. El trabajo de ingeniería está en traducir sin perder significado.

FRASE DE NEGOCIO	LÓGICA	CNF
“Debe haber email o banner.”	$A \lor B$	$A \lor B$
“Si hay email, legal aprueba.”	$A \to C$	$\neg A \lor C$
“Si hay banner, legal aprueba.”	$B \to C$	$\neg B \lor C$
“No pueden estar email y banner a la vez.”	$\neg(A \land B)$	$\neg A \lor \neg B$

La equivalencia más útil es esta:

$$p \to q \equiv \neg p \lor q$$

SÍMBOLO	LECTURA	POR QUÉ IMPORTA
$p \to q$	Si p, entonces q.	Muchas reglas de negocio tienen esta forma.
$\neg p \lor q$	O no ocurre p, o sí ocurre q.	Es la forma que un solver SAT puede consumir en CNF.

Otra familia de reglas aparece continuamente: **exactamente uno**. Por ejemplo, “elige exactamente un plan”. Se suele partir en dos piezas:

$$\text{al menos uno: } x_1 \lor x_2 \lor \cdots \lor x_n$$

$$\text{a lo sumo uno: } \bigwedge_{i<j} (\neg x_i \lor \neg x_j)$$

Esto enseña una lección práctica: a veces una frase sencilla genera muchas cláusulas. Si tienes 100 opciones y codificas “a lo sumo una” por pares, salen $100 \cdot 99/2 = 4950$ cláusulas. No es malo por sí mismo, pero conviene saber qué estás creando.

Cómo decide un solver: DPLL

Hasta aquí hemos comprobado una asignación a mano. Pero, ¿cómo encuentra un solver la asignación, o decide que no existe, sin probar las 2^n combinaciones? El algoritmo clásico es DPLL (Davis, Putnam, Logemann y Loveland), la base de casi todos los solvers modernos.[4] Es una búsqueda en profundidad (la del capítulo 2) sobre las variables, pero con una idea que lo cambia todo: la **propagación unitaria**.

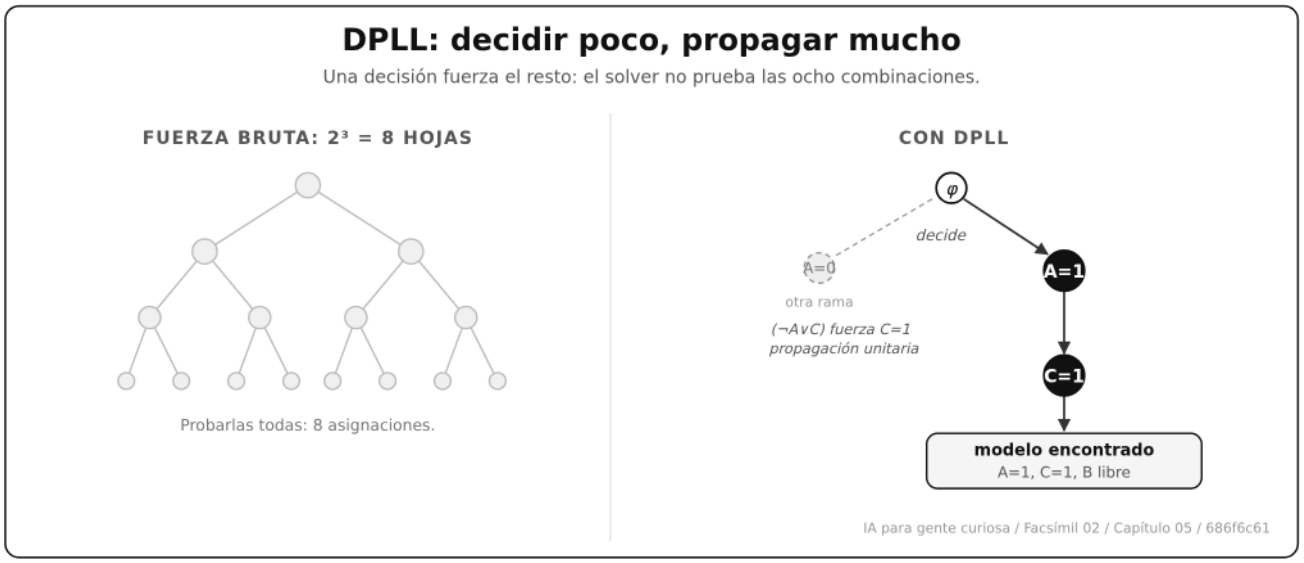
Una cláusula es *unitaria* cuando todos sus literales menos uno ya son falsos. Entonces ese único literal está **forzado**: si la cláusula tiene que ser verdadera, ese literal tiene que ser verdadero. No hay nada que elegir. DPLL aplica esta regla en cadena: cada valor forzado

puede dejar otra cláusula unitaria, que fuerza otro valor, y así sucesivamente, podando el árbol antes de ramificar. El bucle es: propagar las cláusulas unitarias, comprobar si alguna cláusula ha quedado con todos sus literales en falso (conflicto, entonces retrocede), y si no, decidir un valor para una variable libre y volver a propagar.

Veámoslo en la campaña, $\varphi = (A \vee B) \wedge (\neg A \vee C) \wedge (\neg B \vee C)$. Supongamos que el solver decide $A = 1$:

PASO	QUÉ OCURRE	ESTADO
Decisión	$A = 1$	$(A \vee B)$ ya es verdadera; en $(\neg A \vee C)$, $\neg A$ es falso
Propagación	$(\neg A \vee C)$ queda unitaria y fuerza $C = 1$	$C = 1$
Comprobación	con $C = 1$, $(\neg B \vee C)$ ya es verdadera	sin conflicto
Resultado	todas las cláusulas satisfechas	modelo $A = 1, C = 1$, B queda libre

El solver no probó las ocho combinaciones: una decisión y una propagación bastaron. Los solvers reales van más allá con CDCL (*conflict-driven clause learning*): cuando llegan a un conflicto, **aprenden** una cláusula nueva que resume por qué esa rama falló, para no repetir el error. Ese aprendizaje de cláusulas es lo que permite hoy resolver fórmulas con millones de variables. El *backtracking* completo, la maquinaria que comparten DPLL y los CSP, se desarrolla en el capítulo 7.



Dónde está la frontera de dificultad

Cook demostró que SAT es NP-completo, pero la dificultad no se reparte por igual. Si cada cláusula tiene como mucho dos literales (2-SAT), el problema se resuelve en tiempo polinómico, rápido y predecible. En cuanto las cláusulas pueden tener tres literales (3-SAT), vuelve a ser NP-completo, tan difícil como el SAT general.[5] Esa frontera entre dos y tres literales por cláusula es una de las más estudiadas de la informática teórica. La lección práctica es directa: la forma en que codificas un problema, cuántos literales metes en cada cláusula, decide si caes en la zona fácil o en la difícil.

CSP: variables, dominios y restricciones

Un CSP, problema de satisfacción de restricciones, generaliza la idea. Ya no trabajamos solo con verdadero/falso. Una variable puede ser una sala, una hora, una persona, una ruta, un plan o una configuración. Los primeros trabajos sobre redes de restricciones formalizaron esta forma de representar problemas combinatorios como relaciones entre variables.[6] Mackworth popularizó la consistencia de redes de relaciones como herramienta para reducir búsqueda antes de probar soluciones completas.[7]

CSP es parecido, pero ya no todo cabe en interruptores. Piensa en una agenda: una reunión no es “verdadera” o “falsa”; hay que ponerla a una hora. Una sala no es “verdadera” o “falsa”; hay que elegir cuál. Por eso CSP habla de variables con dominios.

VARIABLE	PREGUNTA COTIDIANA	DOMINIO POSIBLE
R_1	¿A qué hora va la reunión de producto?	9:00 o 10:00.
R_2	¿A qué hora va la reunión con cliente?	9:00 o 10:00.
R_3	¿A qué hora va la revisión técnica?	9:00 o 10:00.

Formalmente, un CSP se puede escribir así:

$$\mathcal{P} = (X, D, C)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$\mathcal{P}$	Problema completo de satisfacción de restricciones.	Agenda semanal del equipo.
X	Conjunto de variables que hay que asignar.	$X = \{R_1, R_2, R_3\}$, tres reuniones.
D	Dominios permitidos para esas variables.	Cada reunión puede ir a las 9:00 o 10:00.
C	Conjunto de restricciones que deben cumplirse.	Sin solapes para la misma persona.

Cada variable X_i tiene su dominio:

$$X = \{X_1, \ldots, X_n\}, \qquad X_i \in D_i$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
X_i	Variable individual.	R_1: reunión de producto.
n	Número de variables.	$n = 3$.
D_i	Dominio de valores posibles para X_i.	$D_1 = \{9, 10\}$.
$X_i \in D_i$	La variable debe tomar un valor permitido.	$R_1 = 9$.

Una restricción es una relación sobre una o varias variables:

$$C_k = (S_k, R_k)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
C_k	Restricción número k.	“Ana no puede estar en dos reuniones a la vez”.
S_k	Alcance de la restricción: variables afectadas.	$S_k = (R_1, R_2)$.
R_k	Relación permitida entre los valores de esas variables.	$R_1 \neq R_2$.

Una asignación a es solución si asigna un valor permitido a cada variable y todas las restricciones se cumplen:

$$\forall X_i \in X:\ a(X_i) \in D_i \qquad \text{y} \qquad \forall C_k \in C:\ C_k(a) = \text{verdadero}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
a	Asignación de valores a variables.	$a(R_1) = 9, a(R_2) = 10, a(R_3) = 9$.
$a(X_i)$	Valor elegido para la variable X_i.	$a(R_2) = 10$.
$C_k(a)$	Evaluación de la restricción bajo la asignación a.	$R_1 \neq R_2$ es verdadero.
verdadero	La restricción queda satisfecha.	No hay conflicto.

Ejemplo mínimo:

VARIABLE	DOMINIO	SIGNIFICADO
R_1	$\{9, 10\}$	Reunión de producto.
R_2	$\{9, 10\}$	Reunión con cliente.
R_3	$\{9, 10\}$	Revisión técnica.

Restricciones:

1. $R_1 \neq R_2$, porque Ana participa en ambas.
2. $R_2 = 10$, porque el cliente solo puede a las 10:00.
3. $R_3 \neq R_2$, porque comparten sala.

Sin restricciones hay $2^3 = 8$ asignaciones. Con restricciones, una solución válida es:

$$a(R_1) = 9, \qquad a(R_2) = 10, \qquad a(R_3) = 9$$

Comprueba:

RESTRICCIÓN	SUSTITUCIÓN	RESULTADO
$R_1 \neq R_2$	$9 \neq 10$	Verdadero
$R_2 = 10$	$10 = 10$	Verdadero
$R_3 \neq R_2$	$9 \neq 10$	Verdadero

Esta asignación no es “plausible”. Es válida. Esa diferencia es el corazón del capítulo.

Modelar CSP como ingeniería, no como decoración matemática

Un CSP bien modelado empieza con decisiones de representación. Si eliges mal las variables, el problema explota. Si eliges dominios demasiado grandes, el solver prueba demasiado. Si escondes restricciones globales como muchas restricciones pequeñas sin necesidad, pierdes estructura.

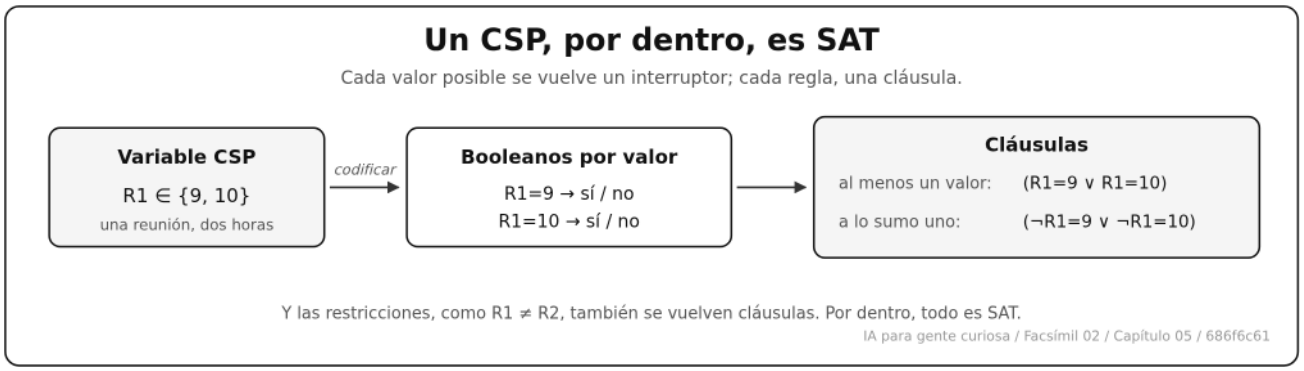
DECISIÓN DE MODELADO	PREGUNTA DE INGENIERÍA	CONSECUENCIA
Variables	¿Qué estoy asignando realmente?	Reunión-franja, persona-turno, tarea-máquina.
Dominio	¿Qué valores son posibles antes de buscar?	Cuanto más limpio el dominio, menos combinatoria.
Restricciones duras	¿Qué no puede violarse nunca?	Filtran candidatos inválidos.
Restricciones blandas	¿Qué preferimos si se puede?	Definen coste entre soluciones válidas.
Granularidad	¿Una variable enorme o varias pequeñas?	Afecta a propagación, explicación y depuración.

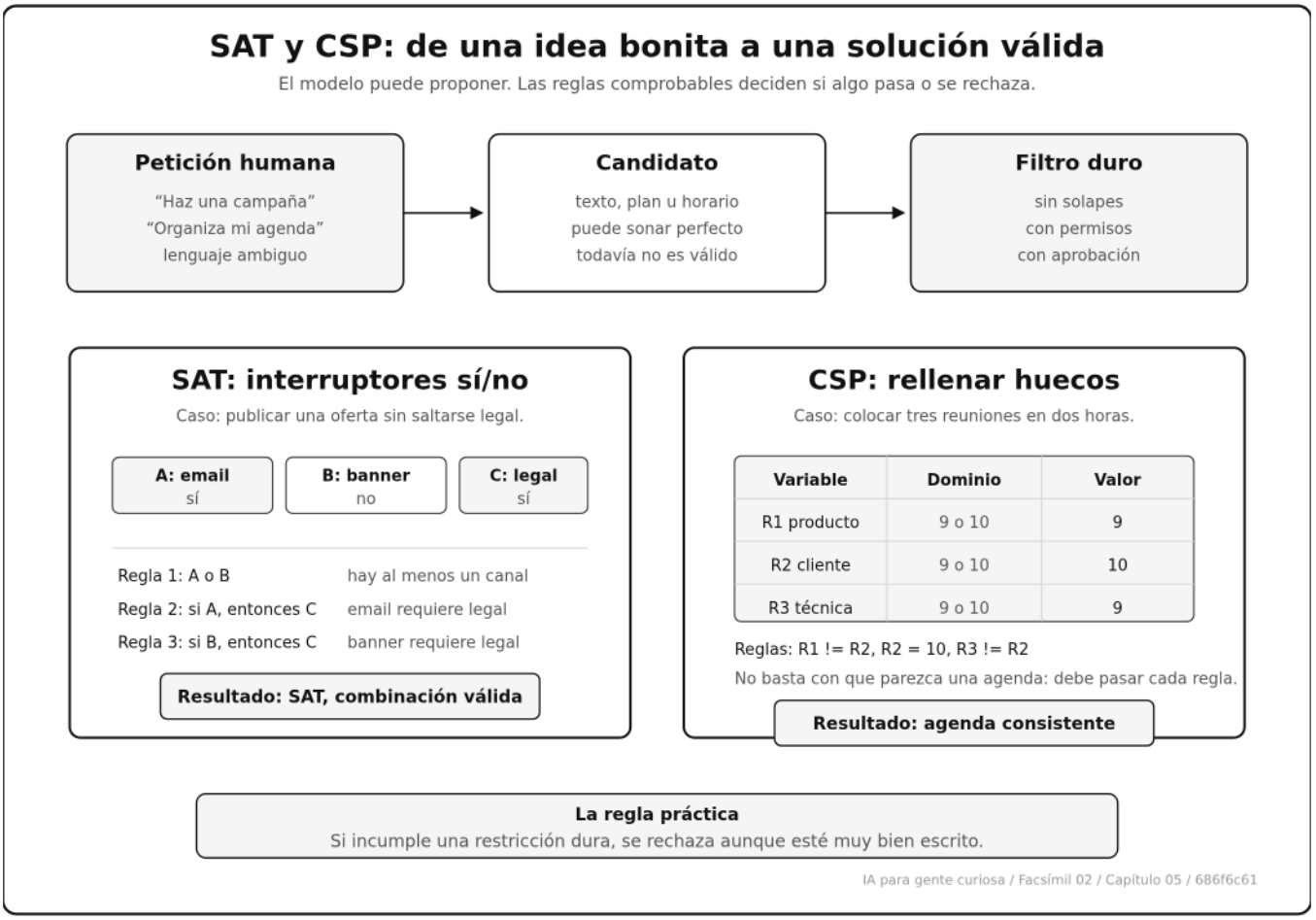
En un sistema real conviene registrar también por qué falla una asignación. “No hay solución” puede ser correcto, pero no siempre es suficiente para operar. Si un horario no existe porque Ana, la sala 2 y el cliente tienen ventanas incompatibles, esa explicación permite corregir datos, relajar preferencias o pedir una decisión humana.

SAT y CSP son la misma familia

SAT y CSP no son dos mundos separados: son el mismo problema visto con distinto grano. SAT es el caso particular de CSP donde todas las variables son booleanas, con dominio $\{0, 1\}$, y las restricciones son cláusulas. Y al revés: todo CSP de dominios finitos se puede **codificar** como SAT, representando cada par variable-valor con un booleano (R_1 vale 9, sí o no) y

traduciendo las restricciones a cláusulas.[8] Esa equivalencia tiene una consecuencia útil: un buen solver SAT puede resolver problemas que ni siquiera parecen booleanos, porque por dentro todo acaba siendo interruptores y cláusulas. En la práctica se elige el formalismo que hace el modelo más natural y más fácil de explicar, sabiendo que por debajo son la misma maquinaria.





De validez a optimización

Hasta ahora solo hemos preguntado si existe una solución válida. Muchas veces queremos algo más: la mejor solución válida según un coste. Eso nos lleva a la optimización con restricciones, que se puede escribir así:

$$a^* = \arg\min_{a \in \mathcal{A}} J(a) \quad \text{sujeto a} \quad \forall C_k \in C:\ C_k(a) = \text{verdadero}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
a^*	Mejor asignación válida encontrada.	Agenda con menos cambios respecto a la semana anterior.
$\mathcal{A}$	Conjunto de asignaciones candidatas.	Todas las agendas posibles.
$J(a)$	Función de coste o penalización.	Número de cambios de sala + preferencias incumplidas.
C_k	Restricción dura.	Nadie puede estar en dos reuniones a la vez.
$C_k(a) = \text{verdadero}$	La asignación cumple la regla k.	El calendario no tiene solapes.

La parte importante es el orden mental: primero validez, después preferencia. Si mezclas reglas duras y preferencias blandas, puedes convertir un problema resoluble en uno imposible.

Ejemplo:

AGENDA	RESTRICCIONES DURAS	COSTE $J(A)$	DECISIÓN
a_1	Cumple todas	5	Válida, pero mejorable
a_2	Incumple un solape	1	Rechazada aunque sea barata
a_3	Cumple todas	2	Mejor válida

La agenda a_2 no gana porque su coste sea menor. Si viola una restricción dura, queda fuera. Entre las válidas, elegimos la de menor coste: a_3.

En el día a día

SAT y CSP aparecen cada vez que un sistema debe decir “esto se puede” o “esto no se puede” de forma verificable.

En configuración de producto, un cliente puede activar módulos, planes, complementos y permisos. Algunas combinaciones son incompatibles: si `plan_basic=true`, quizá `soporte_dedicado=false`. Eso se parece mucho a SAT.

En planificación de turnos, cada persona tiene disponibilidad, descansos mínimos, habilidades, límites legales y preferencias. Eso se parece mucho a CSP: variables con dominios y restricciones entre ellas. Dechter lo trata precisamente como procesamiento de restricciones: reducir dominios, propagar consecuencias y buscar solo donde todavía puede existir solución.[9]

En sistemas con LLMs, la conexión es todavía más práctica. El modelo puede redactar un plan, pero el sistema debería validar permisos, formato, estados permitidos y acciones peligrosas antes de ejecutar. Ahí SAT y CSP se convierten en arquitectura: no son solo algoritmos, son una forma de separar creatividad y garantía.

Por qué debería importarte

Porque los LLMs son buenos generando candidatos, pero no son garantías. Si pides “haz un horario sin conflictos”, puede devolver algo que parece correcto y contiene un solape escondido. Si pides “crea una configuración compatible”, puede inventar una combinación que viola una regla comercial.

SAT y CSP te enseñan a diseñar sistemas donde la respuesta final no se acepta por estilo, sino por verificación. Esta idea conecta directamente con *guardrails*, validadores, permisos, planificación, agentes y evaluación. Poole, Mackworth y Goebel presentan esta separación entre representación, inferencia y búsqueda como una de las bases de la IA computacional.[10]

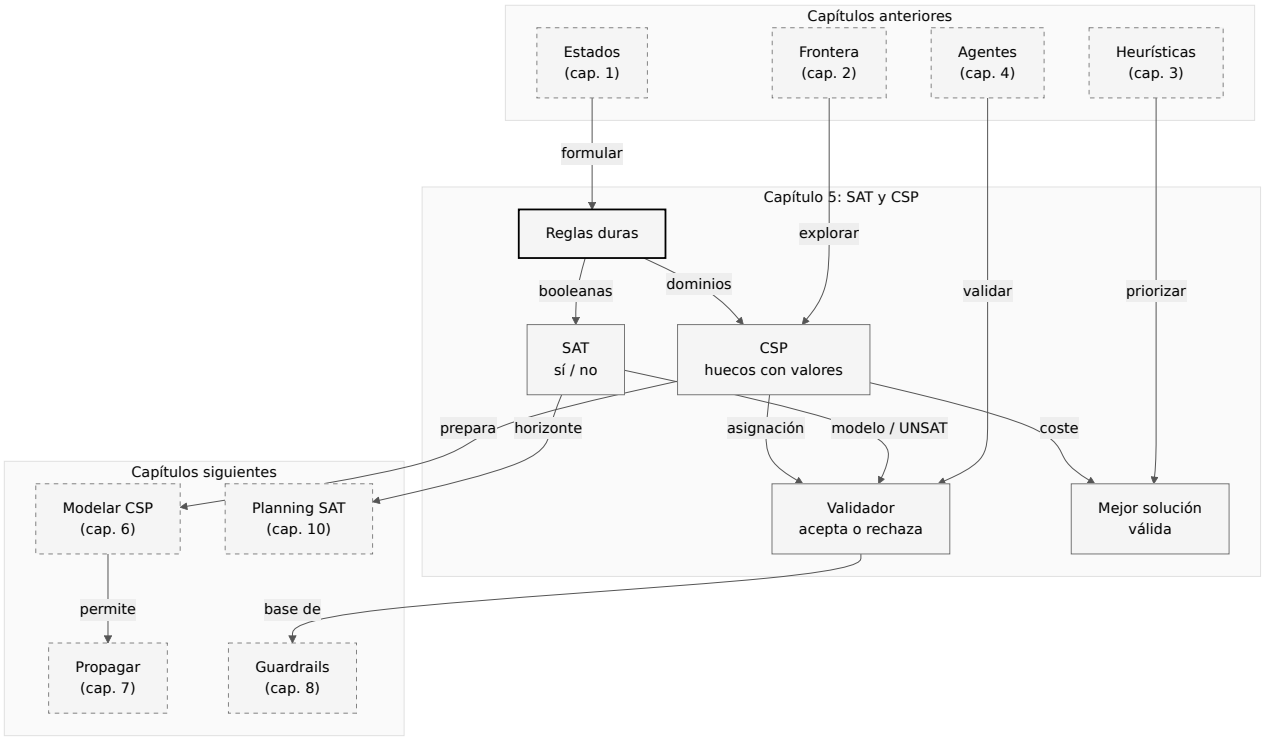
Dónde solía tropezar yo

ERROR	POR QUÉ ES UN ERROR	ANTÍDOTO
Tratar una preferencia como restricción dura	“Ana prefiere mañana” no es lo mismo que “Ana solo puede mañana”. Si lo endureces todo, el problema puede volverse imposible.	Marca cada regla como dura o blanda antes de modelar. Lo duro filtra; lo blando puntúa.
Pensar que SAT y CSP generan explicaciones	Un solver puede decir SAT, UNSAT o devolver una asignación. La explicación pedagógica es otra capa.	Usa el solver para validez y una capa aparte para explicar por qué una solución cumple o falla.
Validar después de actuar	Si ejecutas una acción y luego descubres que violaba una restricción, ya has convertido un problema lógico en un incidente operativo.	Valida antes de hacer *commit*: antes de enviar, reservar, desplegar o cobrar.
Modelar variables enormes	Una variable gigante tiene un dominio inmenso y restricciones difíciles de expresar.	Divide el problema en decisiones pequeñas: persona-día, reunión-franja, permiso-acción.

Cómo encaja todo

Este mapa marca el cambio de fase del facsímil: dejamos de pensar solo en caminos y empezamos a pensar en reglas que una solución debe satisfacer. La búsqueda sigue estando debajo, pero ahora el criterio principal no es “qué nodo miro primero”, sino “qué combinaciones quedan permitidas”.

La decisión aprendida aquí es convertir frases del mundo en restricciones verificables. Esa idea se reutiliza en CSP, guardrails, planificación y sistemas con permisos.


Capítulos anteriores
Estados (cap. 1)
Frontera (cap. 2)
Agentes (cap. 4)
Heurísticas (cap. 3)
formular
Capítulo 5: SAT y CSP
Reglas duras
explorar
booleanas
dominios
validar
priorizar
SAT sí / no
CSP huecos con valores
prepara
horizonte
asignación
modelo / UNSAT
coste
Capítulos siguientes
Modelar CSP (cap. 6)
Planning SAT (cap. 10)
Validador acepta o rechaza
Mejor solución válida
permite
base de
Propagar (cap. 7)
Guardrails (cap. 8)

Vocabulario aprendido

TÉRMINO	DEFINICIÓN
SAT	Problema de decidir si una fórmula booleana tiene alguna asignación que la haga verdadera.
UNSAT	Resultado que indica que ninguna asignación cumple todas las cláusulas.
Modelo SAT	Asignación concreta que satisface la fórmula.
CNF	Forma normal conjuntiva: una conjunción (AND) de cláusulas, cada una disyunción (OR) de literales.
NP-completo	Clase de los problemas más difíciles de NP; SAT fue el primero que se demostró NP-completo.
DPLL	Algoritmo base de los solvers SAT: búsqueda en profundidad con propagación unitaria y backtracking.
Propagación unitaria	Regla que fuerza el valor del único literal libre de una cláusula cuando los demás ya son falsos.
CSP	Problema definido por variables, dominios y restricciones.
Dominio	Conjunto de valores permitidos para una variable.
Restricción dura	Regla que no se puede violar. Si se viola, la solución se rechaza.
Restricción blanda	Preferencia que puede incumplirse pagando un coste o penalización.
Validador determinista	Componente que decide validez aplicando reglas comprobables.

Antes de pasar página

- [ ] ¿Puedo explicar la diferencia entre SAT y CSP sin usar jerga? (Si no, vuelve a «SAT: verdadero, falso y contradicción» y «CSP: variables, dominios y restricciones».)
- [ ] ¿Sé escribir una fórmula CNF pequeña y probar una asignación? (Si no, vuelve a «SAT: verdadero, falso y contradicción».)
- [ ] ¿Entiendo cómo un solver decide con propagación unitaria y backtracking? (Si no, vuelve a «Cómo decide un solver: DPLL».)

- [ ] ¿Puedo formular un CSP como $\mathcal{P} = (X, D, C)$? (Si no, vuelve a «CSP: variables, dominios y restricciones».)
- [ ] ¿Veo por qué SAT y CSP son la misma familia? (Si no, vuelve a «SAT y CSP son la misma familia».)
- [ ] ¿Distingo una restricción dura de una preferencia blanda? (Si no, vuelve a «De validez a optimización».)
- [ ] ¿Sé explicar por qué un candidato válido no tiene por qué ser el mejor? (Si no, vuelve a «De validez a optimización».)
- [ ] ¿Entiendo por qué un LLM puede proponer, pero no debería ser quien garantice la validez final? (Si no, vuelve a «Por qué debería importarte».)

En resumen

IDEA FUERZA	DETALLE
SAT pregunta por verdad booleana.	Busca una asignación de verdadero/falso que satisfaga todas las cláusulas.
CSP amplía la idea a dominios ricos.	Variables, valores permitidos y restricciones describen horarios, permisos, recursos y configuraciones.
Validez y preferencia no son lo mismo.	Primero se descartan soluciones inválidas; después se optimiza entre las válidas.
La IA moderna necesita restricciones clásicas.	Un LLM puede generar candidatos, pero la aceptación debe pasar por reglas verificables.

Para saber más

Biere, A., Heule, M., van Maaren, H. y Walsh, T. (Eds.). (2009). *Handbook of satisfiability*. IOS Press.

Cook, S. A. (1971). The complexity of theorem-proving procedures. En *Proceedings of the Third Annual ACM Symposium on Theory of Computing* (pp. 151-158). ACM. https://doi.org/10.1145/800157.805047

Davis, M., Logemann, G. y Loveland, D. (1962). A machine program for theorem-proving. *Communications of the ACM*, 5(7), 394-397. https://doi.org/10.1145/368273.368557

Dechter, R. (2003). *Constraint processing*. Morgan Kaufmann.

Garey, M. R. y Johnson, D. S. (1979). *Computers and intractability: a guide to the theory of NP-completeness*. W. H. Freeman.

Mackworth, A. K. (1977). Consistency in networks of relations. *Artificial Intelligence*, 8(1), 99-118. https://doi.org/10.1016/0004-3702(77)90007-8

Montanari, U. (1974). Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7, 95-132. https://doi.org/10.1016/0020-0255(74)90008-5

Poole, D., Mackworth, A. y Goebel, R. (1998). *Computational intelligence: a logical approach*. Oxford University Press.

Rossi, F., van Beek, P. y Walsh, T. (Eds.). (2006). *Handbook of constraint programming*. Elsevier.

Russell, S. y Norvig, P. (2021). *Artificial intelligence: a modern approach* (4.ª ed.). Pearson.

Notas

1. Russell, S. y Norvig, P. (2021). *Artificial intelligence: a modern approach* (4.ª ed.). Pearson. Los capítulos sobre búsqueda y satisfacción de restricciones presentan los problemas como asignaciones sometidas a condiciones verificables. ↩

2. Cook, S. A. (1971). The complexity of theorem-proving procedures. En *Proceedings of the Third Annual ACM Symposium on Theory of Computing* (pp. 151-158). ACM. https://doi.org/10.1145/800157.805047 ↩

3. Biere, A., Heule, M., van Maaren, H. y Walsh, T. (Eds.). (2009). *Handbook of satisfiability*. IOS Press. ↩

4. Davis, M., Logemann, G. y Loveland, D. (1962). A machine program for theorem-proving. *Communications of the ACM*, 5(7), 394-397. https://doi.org/10.1145/368273.368557 ↩

5. Garey, M. R. y Johnson, D. S. (1979). *Computers and intractability: a guide to the theory of NP-completeness*. W. H. Freeman. El libro de referencia sobre NP-completitud, que sitúa 3-SAT entre los problemas centrales de la clase y demuestra la separación con 2-SAT. ↩

6. Montanari, U. (1974). Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7, 95-132. https://doi.org/10.1016/0020-0255(74)90008-5 ↩

7. Mackworth, A. K. (1977). Consistency in networks of relations. *Artificial Intelligence*, 8(1), 99-118. https://doi.org/10.1016/0004-3702(77)90007-8 ↩

8. Russell, S. y Norvig, P. (2021). *Artificial intelligence: a modern approach* (4.ª ed.). Pearson. El tratamiento de la satisfacción de restricciones muestra cómo un CSP de dominios finitos se reduce a SAT mediante una codificación de variables indicadoras. ↩

9. Dechter, R. (2003). *Constraint processing*. Morgan Kaufmann. ↩

10. Poole, D., Mackworth, A. y Goebel, R. (1998). *Computational intelligence: a logical approach*. Oxford University Press. ↩