Facsímil 02

Inteligencia clásica

Capítulo 10

Planificación heurística, con SAT y agentes LLM

686f6c61 · 2026

Capítulo 10: Planificación heurística, con SAT y agentes LLM

Entrando en el tema

En el capítulo anterior construimos un plan pequeño: validar una factura, enviarla y registrar el envío. Tres acciones. Mundo amable. Todo cabía en la cabeza.

Ahora subamos un poco la temperatura. Imagina un agente que prepara una release de software: ejecutar tests, revisar migraciones, comprobar permisos, generar changelog, pedir aprobación si hay riesgo, desplegar en staging, observar logs, desplegar en producción y dejar trazas. Además, cualquier paso puede fallar.

Ahí una lista de tareas ya no basta. Necesitamos decidir qué probar primero, cómo evitar combinaciones absurdas, cuándo preguntar a un solver y cuándo dejar que un LLM proponga el siguiente paso sin convertirlo en autoridad absoluta.

Tres maneras de no perderse

La planificación clásica enseña una idea sencilla: un plan es una secuencia de acciones aplicables que llega al objetivo. El problema es que las secuencias posibles crecen muy deprisa. Bylander mostró que incluso versiones proposicionales de STRIPS tienen complejidad computacional dura.[1]

En la práctica, se suelen combinar tres estrategias:

ESTRATEGIA	QUÉ HACE	CUÁNDO AYUDA
Heurística	Ordena la búsqueda por promesa.	Cuando hay muchas acciones posibles.
SAT	Pregunta si existe un plan de longitud k.	Cuando queremos una prueba lógica de factibilidad.
Agente LLM	Propone pasos y explica decisiones.	Cuando el entorno es abierto o lingüístico.

La clave es no confundir sus papeles. Una heurística orienta, pero puede equivocarse. SAT verifica una codificación, pero necesita un horizonte y un modelo. Un LLM propone y adapta lenguaje, pero sus pasos deben validarse.

Antes de elegir motor, conviene recuperar la definición clásica del capítulo anterior, la de Ghallab, Nau y Traverso. Un plan π no es una intención ni una explicación bonita: es una secuencia de acciones que transforma estados sin saltarse precondiciones:[2]

$$\pi = \langle a_0, a_1, \ldots, a_{n-1} \rangle, \quad s_{t+1} = \gamma(s_t, a_t), \quad G \subseteq s_n$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
π	Plan completo.	Validar, enviar, registrar.
a_t	Acción elegida en el paso t.	`enviar_factura`.
$\gamma(s_t, a_t)$	Función de transición: aplica una acción a un estado.	Si la factura está validada, pasa a enviada.
$G \subseteq s_n$	El estado final contiene todos los objetivos.	Hay email enviado y log creado.

Si además hay costes, no basta con llegar. También importa cómo llegamos. El coste de un plan es la suma de los costes de sus acciones, el mismo coste de camino acumulado (la g de A*) que vimos en búsqueda:

$$C(\pi) = \sum_{t=0}^{n-1} c(a_t)$$

Un plan de tres pasos puede ser peor que uno de cinco si el tercero manda un email irreversible sin revisión humana. Por eso en sistemas reales solemos mezclar longitud, coste, riesgo, permisos y confianza en la observación.

La misma tarea se ve distinta según la lente:

LENTE	PREGUNTA QUE HACE	RESPUESTA ÚTIL
Heurística	¿Qué estado parece más cerca del objetivo?	“Prueba primero el camino que ya tiene tests y changelog”.
SAT	¿Existe algún plan de k pasos que cumpla todas las reglas?	“Con $k = 2$ no; con $k = 3$ sí”.
Agente LLM	¿Qué paso tiene sentido proponer con este contexto textual?	“Pide aprobación antes de desplegar porque hay una migración”.

Planificación como búsqueda heurística

Podemos leer un planner como un buscador en estados. Desde un estado s, aplicamos acciones válidas y avanzamos. Para no probar todo a ciegas, usamos una función de evaluación. No es nueva: es exactamente la de A*, el algoritmo de Hart, Nilsson y Raphael que vimos en el capítulo 3 de búsqueda.[3]

$$f(s) = g(s) + h(s)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
s	Estado candidato durante la búsqueda.	Tests pasados, changelog pendiente.
$g(s)$	Coste acumulado desde el inicio.	Dos acciones ejecutadas.
$h(s)$	Estimación del coste restante hasta el objetivo.	Faltan deploy y verificación.
$f(s)$	Prioridad total del estado.	$2 + 2 = 4$.

El significado es el mismo que en búsqueda: $g(s)$ es el coste real ya pagado para llegar a s, $h(s)$ es la estimación de lo que falta, y la suma $f(s)$ marca qué estado mirar antes. Y vale la misma garantía: si h nunca sobrestima el coste que de verdad queda, es decir, si es **admisible**, A* encuentra el plan óptimo. Lo que cambia en planificación es de dónde sale h: en vez de una distancia geométrica, suele estimar cuántas acciones faltan o cuántos hechos objetivo siguen sin cumplirse. Bonet y Geffner formularon la planificación como búsqueda heurística y mostraron cómo estimaciones relativamente simples podían guiar planners hacia soluciones útiles.[4]

La heurística más simple de todas es real, aunque poco informativa: contar cuántos hechos del objetivo faltan por cumplir.[5]

$$h(s) = |G \setminus s|$$

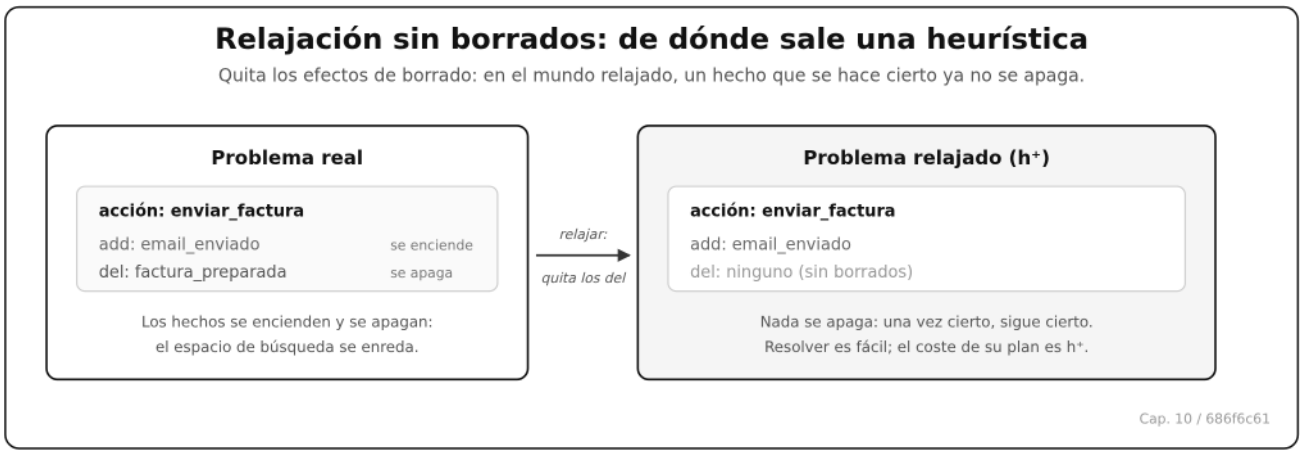
SÍMBOLO	SIGNIFICADO	EJEMPLO
G	Conjunto de hechos objetivo.	`{tests_ok, deploy_hecho, logs_ok}`.
s	Hechos verdaderos ahora.	`{tests_ok}`.
$G \setminus s$	Objetivos que todavía faltan.	`{deploy_hecho, logs_ok}`.
$	G\setminus s	$

Es barata de calcular, pero pobre: solo cuenta cuántos objetivos quedan, sin tener en cuenta mutex, recursos, precondiciones ocultas ni el coste de las acciones; por eso puede ordenar mal. Los planners reales usan heurísticas más informativas, como las que derivan de planes relajados o grafos de planificación. Graphplan introdujo una forma influyente de razonar con grafos de niveles y exclusiones mutuas.[6] FF popularizó heurísticas basadas en planes relajados: ignorar ciertos efectos negativos para estimar una ruta optimista hacia el objetivo.[7]

Heurísticas que sí informan: relajar el problema

¿De dónde sale una heurística buena, y no inventada? El truco más fértil de la planificación es **relajar** el problema: resolver una versión más fácil y usar su coste como estimación. La relajación estrella es la **relajación sin borrados** (*delete-relaxation*): tomas el dominio y borras todos los `del(a)`, de modo que una vez un hecho se hace verdadero, ya nunca deja de serlo. En ese mundo simplificado nada se «estropea», así que resolver es mucho más fácil, y el coste de su plan, llamado h^+, es una estimación informada de lo lejos que estás en el problema real.

Calcular h^+ exacto sigue siendo difícil, así que en la práctica se aproxima. La heurística de FF, una de las más usadas durante años, extrae un *plan relajado* concreto y cuenta sus acciones. La idea es siempre la misma: una heurística no se saca de la manga; se **deriva** de una versión relajada del problema, lo que la hace barata de calcular y, a la vez, conectada con la estructura real del dominio. Esa es la diferencia entre contar objetivos a ciegas y estimar con criterio.



Planificación con SAT

Otra forma de plantear el problema es fijar un horizonte k: "¿existe un plan de k pasos o menos?". En vez de recorrer estados, construimos una fórmula booleana. Si la fórmula es SAT, el modelo nos dice qué acciones ocurren en cada tiempo. Si es UNSAT, no existe plan bajo esa codificación y ese horizonte.

Kautz y Selman hicieron célebre esta idea al formular planificación como satisfacibilidad.[8]

Podemos resumirlo así:

$$\Phi_k = I_0 \wedge T_0 \wedge T_1 \wedge \cdots \wedge T_{k-1} \wedge G_k$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
Φ_k	Fórmula SAT del problema con horizonte k.	"¿Hay plan de 3 pasos?".
I_0	Estado inicial codificado en tiempo 0.	`tests_pendientes@0`.
T_t	Restricciones de transición entre t y $t+1$.	Si `deploy@1`, entonces `build_ok@1`.
G_k	Objetivo exigido en el tiempo final.	`produccion_actualizada@3`.

Las restricciones típicas son:

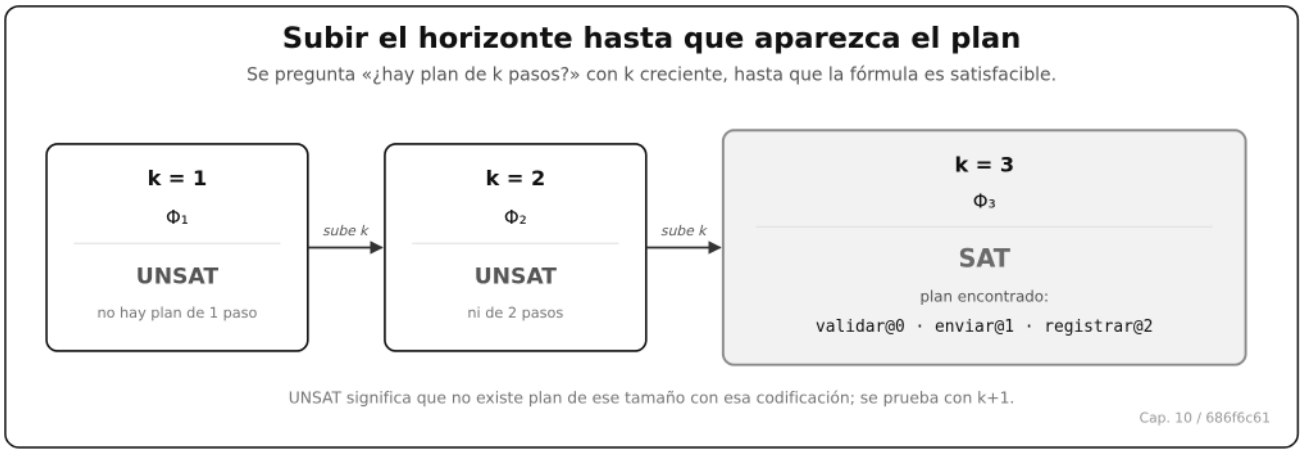
RESTRICCIÓN	FORMA MENTAL	QUÉ IMPIDE
Precondición	$a_t \Rightarrow pre(a)_t$	Ejecutar acciones sin requisitos.
Efecto positivo	$a_t \Rightarrow add(a)_{t+1}$	Acciones que no cambian nada.
Efecto negativo	$a_t \Rightarrow \neg del(a)_{t+1}$	Hechos que sobreviven aunque fueron eliminados.
Persistencia	Si nadie cambia p, p persiste.	Mundos que olvidan hechos arbitrariamente.
Mutex	Acciones incompatibles no ocurren juntas.	Dos acciones compiten por el mismo recurso.

Cómo se construye la fórmula

Esto es más concreto de lo que parece. Cada hecho y cada acción se convierten en variables booleanas **indexadas por tiempo**: `deploy@2` significa «la acción deploy se ejecuta en el paso 2», y `build_ok@2` significa «el hecho build_ok es cierto en el paso 2». Con esas variables, las restricciones de arriba son cláusulas concretas:

REGLA	CLÁUSULA (FORMA)	LECTURA
Estado inicial	`build_ok@0` , `¬deploy_hecho@0`	lo cierto y lo falso en el tiempo 0
Precondición	`deploy@2 → build_ok@2`	si deploy ocurre en 2, su precondición es cierta en 2
Efecto	`deploy@2 → deploy_hecho@3`	si deploy ocurre en 2, su efecto aparece en 3
Persistencia	`(p@t ∧ nadie_lo_borra@t) → p@t+1`	un hecho dura si nadie lo cambia
Mutex	`¬(deploy@2 ∧ rollback@2)`	dos acciones en conflicto no coinciden
Objetivo	`produccion_actualizada@k`	la meta es cierta en el último paso

Júntalas todas con un `∧` gigante y obtienes Φ_k. Si un solver SAT, de los del capítulo 5, encuentra una asignación que la satisface, lees qué variables `accion@t` valen verdadero y ya tienes el plan, paso a paso. Si no la hay, sabes con certeza lógica que **no existe** plan de k pasos con esa codificación, y pruebas con $k+1$. Así, planificar se vuelve una sucesión de preguntas SAT con horizonte creciente, justo lo que compara el lab de este capítulo.



SAT no “entiende” el mundo. Solo decide si la fórmula tiene una asignación que cumple todo. La potencia está en que esa asignación es verificable.

Agentes LLM: propuesta no es permiso

Los agentes LLM reabren la planificación desde otro ángulo. No siempre tenemos un dominio PDDL completo. A veces el mundo es texto, páginas web, APIs, tickets, ficheros y decisiones humanas. Ahí el LLM es útil para proponer pasos, interpretar observaciones y decidir qué información falta.

ReAct mostró una forma influyente de intercalar razonamiento y actuación: el modelo razona, actúa sobre un entorno, observa y continúa.[9] Toolformer exploró cómo un modelo puede aprender a invocar herramientas externas mediante APIs.[10]

Pero aquí conviene ser muy estrictos: un agente LLM no sustituye el modelo de planificación. Lo complementa.

PIEZA	EN PLANNING CLÁSICO	EN AGENTE LLM
Estado	Hechos explícitos.	Contexto, memoria, ficheros, logs.
Acción	Operador formal.	Tool call propuesta.
Precondición	Fórmula verificable.	Validador antes de ejecutar.
Efecto	Add/delete list.	Observación estructurada tras tool.
Replanificación	Nueva búsqueda.	Nuevo paso tras observar realidad.

El patrón sano es:

> **EJEMPLO, NO NOTACIÓN OFICIAL.**
>
> $$s_{t+1} = \text{observe}(\text{exec}(a_t, s_t))$$
>
> *Esta notación es nuestra, una forma compacta de decir que el estado siguiente no es el que el plan esperaba, sino el que de verdad se lee tras ejecutar. La idea de fondo sí es clásica: es el ciclo de percepción y acción de un agente, que en sistemas con observación parcial se formaliza con modelos como los POMDP.*

SÍMBOLO	SIGNIFICADO	EJEMPLO
s_t	Estado antes del paso.	Build verde, deploy pendiente.
a_t	Acción elegida para el paso t.	Ejecutar deploy en staging.
exec	Ejecución real de la herramienta.	Llamada a CI/CD.
observe	Lectura verificable del resultado.	Logs, código de salida, métricas.
s_{t+1}	Estado actualizado tras observar.	Staging desplegado o fallo registrado.

Si s_{t+1} contradice lo esperado, no seguimos “porque el plan lo decía”. Replanificamos.

Modo ingeniero: el bucle ReAct

ReAct (razonar y actuar) se implementa como un bucle muy concreto con el SDK de Anthropic: el modelo razona y propone una herramienta, el código la valida y la ejecuta, la observación vuelve al modelo, y se repite. La pieza que no se delega es la validación: el modelo propone, el código autoriza.

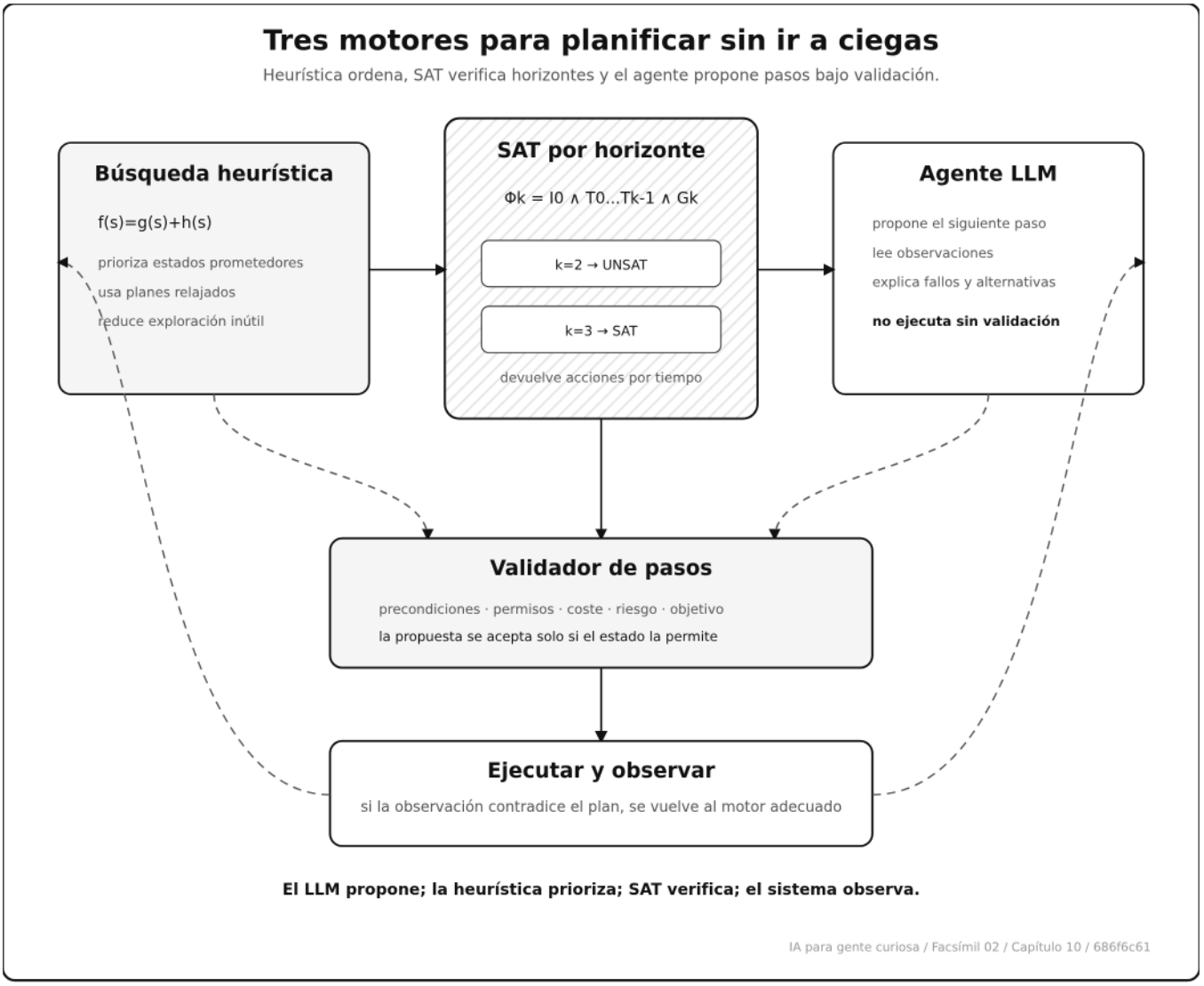
```
import anthropic

client = anthropic.Anthropic()
mensajes = [{"role": "user",
             "content": "Prepara la release. Estado inicial: tests_pendientes."}]

for paso in range(MAX_PASOS):
    # 1. El modelo razona y propone la siguiente accion como tool call.
    respuesta = client.messages.create(
        model="claude-opus-4-8",
        max_tokens=1024,
        tools=TOOLS,
        messages=mensajes,
    )
    mensajes.append({"role": "assistant", "content": respuesta.content})
    if respuesta.stop_reason != "tool_use":
        break                                  # el modelo cree que ha terminado

    for bloque in respuesta.content:
        if bloque.type == "tool_use":
            # 2. El codigo valida precondiciones ANTES de ejecutar.
            if not precondiciones_ok(bloque.name, estado):
                resultado = {"error": "precondicion ausente, no ejecutado"}
            else:
                # 3. Ejecuta y OBSERVA el estado real, no el esperado.
                resultado = observar(ejecutar(bloque.name, bloque.input))
                estado = resultado["estado"]
            # 4. La observacion vuelve al modelo como tool_result.
            mensajes.append({"role": "user", "content": [{
                "type": "tool_result",
                "tool_use_id": bloque.id,
                "content": str(resultado),
            }]})
```

Cada vuelta del bucle es un razonar, actuar y observar. Lo que vuelve al modelo en el `tool_result` no es lo que el plan «esperaba», sino lo que de verdad pasó: si `observar` reporta que el correo dejó de estar confirmado, el modelo lo lee en la siguiente vuelta y replanifica solo. Aquí los tres motores conviven: el modelo propone (el agente LLM), `precondiciones_ok` y los guardrails autorizan (la herencia del capítulo 8), y nada se da por hecho hasta observarlo. La heurística y SAT entran cuando el espacio de pasos es grande: en vez de pedir al modelo que adivine, se le ofrece el plan que el planner ya validó.



En el día a día

En un equipo que construye agentes, este capítulo se traduce en decisiones muy prácticas. No basta con preguntar al modelo “qué harías ahora”. Hay que decidir qué propuestas pasan a ejecución y cuáles se descartan.

SITUACIÓN	LECTURA DE PLANIFICACIÓN	CONTROL ÚTIL
Muchas tools posibles	Alto factor de ramificación.	Heurística por coste, riesgo o progreso.
Duda sobre si existe plan corto	Horizonte k.	Codificación SAT o búsqueda acotada.
Tool crítica	Acción con precondiciones duras.	Guardrails antes de ejecutar.
Resultado inesperado	Estado observado distinto.	Replanificación, no insistencia ciega.

En producción, una buena arquitectura suele separar cuatro capas: el LLM propone, el planificador o política ordena, los validadores autorizan y el monitor observa. Si esas capas se mezclan en un prompt enorme, el sistema puede parecer inteligente en la demo y volverse frágil con usuarios reales.

Por qué debería importarte

Porque los costes de un mal plan no son solo tokens. Un agente que reintenta sin nueva evidencia consume tiempo, dinero y confianza. Un agente que ejecuta pasos en el orden equivocado puede romper datos. Y un agente que no observa efectos reales vive en una ficción: cree que hizo algo porque lo escribió.

La planificación avanzada no consiste en meter más matemática por gusto. Consiste en elegir qué parte del sistema debe decidir qué. Las heurísticas reducen búsqueda, SAT da verificaciones fuertes para horizontes concretos, y los LLMs aportan flexibilidad lingüística. Juntos funcionan mejor cuando cada uno tiene límites claros.

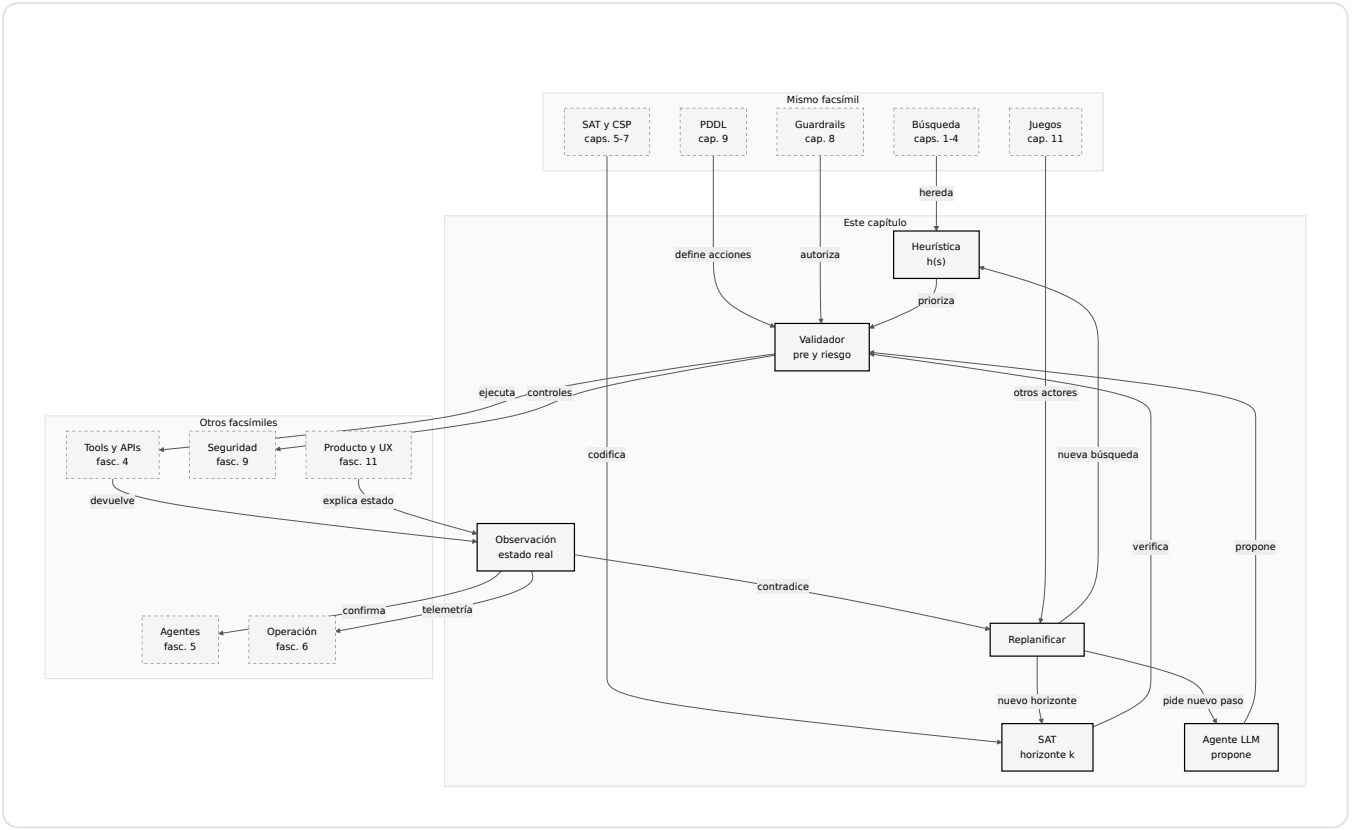
Dónde solía tropezar yo

ERROR	POR QUÉ ES UN ERROR	ANTÍDOTO
Creer que la heurística es una garantía	Una heurística ordena la búsqueda, no demuestra que un paso sea válido.	Validar precondiciones y efectos después de elegir.
Usar SAT sin mirar la codificación	SAT solo verifica lo que hemos escrito en la fórmula.	Revisar transiciones, persistencia y mutex.
Pedir al LLM un plan largo y ejecutarlo entero	El mundo puede cambiar tras el primer paso.	Ejecutar un paso, observar y replanificar.
No contar el coste de las acciones	El plan más corto puede ser caro o peligroso.	Añadir coste, riesgo y aprobación humana.
Reintentar sin nueva información	Un bucle no es planificación; es ausencia de criterio de parada.	Registrar observación nueva o escalar.

Cómo encaja todo

Este mapa compara tres formas de no perderse al planificar: una heurística ordena búsqueda, SAT verifica si existe plan para un horizonte concreto y el LLM propone pasos en entornos abiertos. Ninguna pieza debería ocupar el lugar de las otras.

La decisión aprendida es diseñar un bucle donde proponer, validar, ejecutar, observar y replanificar sean fases separadas. Esa arquitectura se reutiliza directamente en agentes y operación.



Vocabulario aprendido

TÉRMINO	DEFINICIÓN
Heurística de planificación	Estimación del coste restante hasta el objetivo.
Horizonte	Número máximo de pasos permitido en el plan.
Función de transición	Regla que calcula el siguiente estado tras aplicar una acción válida.
Planificación con SAT	Traducción de planning a una fórmula booleana.
Mutex	Relación de incompatibilidad entre acciones o hechos simultáneos.
Plan relajado	Estimación optimista que ignora algunos efectos negativos.
Relajación sin borrados	Problema simplificado que ignora los efectos de borrado; su coste (h+) da una heurística informada.
ReAct	Patrón de agente que intercala razonar, actuar y observar, repitiendo el ciclo.
Bucle agente	Ciclo de proponer, validar, ejecutar, observar y replanificar.

Antes de pasar página

- [ ] ¿Puedo explicar para qué sirve una heurística en planificación? (Si no, vuelve a «Planificación como búsqueda heurística».)
- [ ] ¿Distingo una heurística de una garantía lógica? (Si no, vuelve a «Tres maneras de no perderse».)
- [ ] ¿Entiendo de dónde sale una heurística informada, relajando el problema sin borrados? (Si no, vuelve a «Heurísticas que sí informan: relajar el problema».)
- [ ] ¿Entiendo qué significa preguntar si Φ_k es SAT? (Si no, vuelve a «Planificación con SAT».)
- [ ] ¿Sé escribir una cláusula con variables indexadas por tiempo, como `deploy@2`, y nombrar tres restricciones? (Si no, vuelve a «Cómo se construye la fórmula».)
- [ ] ¿Puedo explicar por qué un agente LLM debe observar antes de seguir? (Si no, vuelve a «Agentes LLM: propuesta no es permiso».)
- [ ] ¿Sé montar el bucle ReAct, razonar, actuar y observar, con validación en código? (Si no, vuelve a «Modo ingeniero: el bucle ReAct».)
- [ ] ¿Entiendo por qué un horizonte $k = 2$ puede ser UNSAT? (Si no, vuelve a «Planificación con SAT».)

En resumen

IDEA FUERZA	DETALLE
La heurística ordena la búsqueda.	Ayuda a mirar primero los estados prometedores, pero no garantiza validez.
SAT verifica horizontes.	Si Φ_k es SAT, tenemos una asignación coherente de acciones y hechos.
El LLM propone, no autoriza.	Sus pasos deben pasar por precondiciones, permisos, coste y observación.
La observación manda.	Si el mundo contradice el plan, el sistema debe replanificar.

Para saber más

Blum, A. L. y Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2), 281-300. https://doi.org/10.1016/S0004-3702(96)00047-1

Bonet, B. y Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, 129(1-2), 5-33. https://doi.org/10.1016/S0004-3702(01)00108-4

Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1-2), 165-204. https://doi.org/10.1016/0004-3702(94)90081-7

Ghallab, M., Nau, D. y Traverso, P. (2004). *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Hoffmann, J. y Nebel, B. (2001). The FF planning system: fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14, 253-302. https://doi.org/10.1613/jair.855

Kautz, H. A. y Selman, B. (1992). Planning as satisfiability. En *Proceedings of the 10th European Conference on Artificial Intelligence* (pp. 359-363). John Wiley and Sons.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N. y Scialom, T. (2023). Toolformer: Language Models Can Teach Themselves to Use Tools. https://doi.org/10.48550/arXiv.2302.04761

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. y Cao, Y. (2023). *ReAct: Synergizing Reasoning and Acting in Language Models*. International Conference on Learning Representations. https://arxiv.org/abs/2210.03629

Notas

1. Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1-2), 165-204. https://doi.org/10.1016/0004-3702(94)90081-7 ↩

2. Ghallab, M., Nau, D. y Traverso, P. (2004). *Automated Planning: Theory and Practice*. Morgan Kaufmann. ↩

3. Hart, P. E., Nilsson, N. J. y Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 100-107. https://doi.org/10.1109/TSSC.1968.300136 ↩

4. Bonet, B. y Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, 129(1-2), 5-33. https://doi.org/10.1016/S0004-3702(01)00108-4 ↩

5. Russell, S. y Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (4.ª ed.). Pearson. La heurística de contar objetivos no satisfechos es el ejemplo más básico de heurística de planificación. ↩

6. Blum, A. L. y Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2), 281-300. https://doi.org/10.1016/S0004-3702(96)00047-1 ↩

7. Hoffmann, J. y Nebel, B. (2001). The FF planning system: fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14, 253-302. https://doi.org/10.1613/jair.855 ↩

8. Kautz, H. A. y Selman, B. (1992). Planning as satisfiability. En *Proceedings of the 10th European Conference on Artificial Intelligence* (pp. 359-363). John Wiley and Sons. ↩

9. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. y Cao, Y. (2023). *ReAct: Synergizing Reasoning and Acting in Language Models*. International Conference on Learning Representations. https://arxiv.org/abs/2210.03629 ↩

.0. Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N. y Scialom, T. (2023). Toolformer: Language Models Can Teach Themselves to Use Tools. https://doi.org/10.48550/arXiv.2302.04761 ↩